CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de

Ingenieros Informáticos

TRABAJO FIN DE GRADO

Testing automatizado y calidad del software

Autor: Jorge Aparicio Rabadán

Director: Oscar Dieste Tubío

MADRID, JULIO 2019

1. Resumen del proyecto

Este proyecto habla acerca del testing web, aportando una introducción sobre la actualidad del testing en general. Seguidamente, se dan a conocer los objetivos a conseguir y la importancia de este trabajo. Más adelante, se describen los antecedentes del testing manual y automatizado y de la librería Selenium, explicando las ventajas y las desventajas de su uso, tratando el framework específicamente, dando a conocer los tipos que existen y sus características. En los siguientes apartados, se presenta la finalidad del proyecto y su estructura a seguir para realizarlo. La sección siguiente, tiene como objeto identificar al lector las situaciones concretas que suelen dar problemas a la hora de automatizar con la librería Selenium, explicando las razones del problema con su respectiva posible solución. Así se tratará de conseguir hacer uso de buenas prácticas que tendrán como consecuencia una correcta programación de las pruebas.

This project talks about web testing, providing an introduction about the current testing in general. Next, the objectives to be achieved and the importance of this work are announced. Later, we describe the background of manual and automated testing and the Selenium library, explaining the advantages and disadvantages of its use, treating the framework specifically, showing the types which exist and their characteristics. In the following sections, the purpose of the project and its structure to be followed is presented. The next section, aims to identify the reader the specific situations that usually give problems when it comes to automate with the Selenium library, explaining the reasons for the problem with their respective possible solution. This will try to make use of good practices that will result in a correct programming of the tests.

2. Introducción

2.1. Contexto

Hoy en día, muchas empresas software, por su negocio, quieren ser ágiles. Quieren crear productos más rápido para venderlos en el mercado y adelantarse a la competencia, mejorando para ello la calidad de su proceso, producto y equipos software.

Una de las claves para agilizar ese proceso es detectar errores en los puntos del desarrollo donde cueste menos solucionarlos y así desarrollar con más seguridad. La clave para conseguir esto es la automatización de ciertos procesos, en particular las pruebas.

El concepto de automatización se entiende como la utilización de la tecnología para la realización de tareas sin necesidad de la intervención humana. La automatización se usa en múltiples empresas, por ejemplo, en las empresas software que tienen un gran volumen de procesos web como las tiendas online (amazon), consumen mucho tiempo en probar su funcionamiento manualmente.

La principal razón por la que se hace uso de la automatización es el tiempo. Gracias a las pruebas automatizadas se aumenta la posibilidad testear rápidamente y sin esfuerzo varias veces. Además, al automatizar pruebas sencillas, los testers reales (manuales) pueden dedicar mayor tiempo a pruebas más críticas y caminos más elaborados dejando los caminos básicos a las pruebas automatizadas.

Uno de los conceptos importantes dentro del campo del testing es la integración continua. Consiste en realizar automatizaciones de manera continuada y así mantener actualizada la misma. Es un modelo muy buscado por las empresas de hoy en día, porque cuando es introducida una nueva funcionalidad o un cambio en el software, que debe ser probado, se evitará la repetición de la automatización completa y por otro lado, siempre se tendrá probado el software.

Podemos llegar a la conclusión que la automatización es una gran vía para destacar entre la competencia y ser capaces de llegar y complacer al cliente de una manera más rápida e innovadora.

Una de las herramientas más utilizadas es Selenium, que es un entorno de pruebas de software para aplicaciones basadas en la web, que proporciona la capacidad de automatizar pruebas web funcionales. Actualmente, es el framework que más se usa a nivel empresarial.

2.2. Objetivos del proyecto

Selenium es una herramienta muy amplia y no es fácil de usar porque conlleva un conocimiento amplio del código fuente de la web y de la misma librería. Al ser tan completa y pudiendo hacer uso de varios de lenguajes de programación para programar los planes de prueba, nos da la opción de poder abarcar sin problema cualquier prueba a realizar. Suele provocar problemas sobretodo si se hace uso del IDE, ya que es el más fácil de usar, porque con activar el modo grabación, solamente hay que ir clicando en los lugares donde se quiere automatizar la prueba y seguidamente, exportar el código. Esto da lugar a errores en la automatización, ya que existen páginas web que están programadas con un código fuente no limpio y dinámico.

2.3. Objetivos del proyecto

A lo largo de mi experiencia laboral, me he encontrado con situaciones en las que he precisado ayuda de una o varias personas para avanzar, por ejemplo, debido al desconocimiento de problemas típicos del uso de la herramienta Selenium o la mala programación de la web. Esto provoca una pérdida de tiempo tanto por parte del automatizador (en este caso yo) y por supuesto, por parte de los demás compañeros que dejan de hacer su trabajo para centrarse en el problema. A la hora de reportar resultados de equipo, observamos un claro descenso del rendimiento de los trabajadores dando lugar a un retraso de las tareas conllevando incluso, una pérdida económica para la empresa.
Existe mucha información sobre cómo hacer uso de esta herramienta, pero es mucho más complicado encontrar información sobre cómo resolver casos puntuales complicados. Es por ello que es necesaria la descripción de estos casos y cómo solucionarlos.

2.4. Contribuciones del proyecto

Este proyecto presenta una guía acerca del uso de las librerías selenium, identificando los problemas y aportando los detalles suficientes para evitar situaciones que conllevan a un uso incorrecto de la librería, consiguiendo que las automatizaciones sean exitosas ahora y en un futuro. También se pretende obtener el mínimo conocimiento para no quedarse estancado en problemas a la hora de automatizar, es decir, dar facilidades al programador en los distintos casos de prueba.

En la siguiente tabla se describen las compatibilidades de los distintos tipos de Selenium, con la finalidad de tener con buena y rápida visibilidad de los tipos que ofrece el framework, describiendo las características principales que determinarán que tipo de Selenium tiene a su disposición el automatizador:

2.5. Estructura del proyecto

Se comenzará con los antecedentes, que expondrán la situación del testing explicando por qué se debe automatizar y cuáles son sus ventajas respecto a lo manual. También se describirán qué tipos de Selenium existen con sus características principales de cada uno.
En la sección objetivos del proyecto, se describirá la finalidad de este proyecto y lo que se quiere conseguir.
La estrategia de la realización del proyecto define los pasos que se han seguido para desarrollar este proyecto de principio a fin.
En el apartado realización del proyecto, se ha realizado la guía explicando en detalle los problemas identificados con sus respectivas soluciones.
En las Conclusiones, se expone la opinión personal y los aprendizajes que he obtenido al realizar el proyecto.
Y para terminar, referencias bibliográficas describen los lugares donde se ha obtenido parte de la información para realizar el proyecto

2. Antecedentes

2.1. Pruebas software

La calidad del software tiene muchos ámbitos (Calidad del software es mucho más que el testing).
Dentro del campo testing, existen distintos tipos de pruebas, cada una orientada a detectar y prevenir ciertos tipos de errores en el software.

2.2. Automatización de pruebas

En la siguiente tabla, se describen los tipos de Selenium que existen para automatizar:

Selenium IDE	**Selenium RC**	**Selenium Webdriver**	**Selenium Grid**
Mozilla, Chrome	Todos los navegadores, excepto la última versión de mozilla	Todos los navegadores con todas sus versiones	Todos los navegadores con todas sus versiones
Grabación y ejecución	Ejecución	Ejecución	Ejecución
No se requiere arrancar el servidor	Requiere arrancar el servidor	No se requiere arrancar el servidor	No se requiere arrancar el servidor
Su lenguaje core es javaScript	Su lenguaje core es javaScript	Interactúa de manera nativa con el navegador	Interactúa de manera nativa con el navegador
Fácil uso. Requiere poco conocimiento de javascript	Fácil de usar porque es una pequeña API	Uso complejo dadas sus dimensiones y su potencia	Uso complejo dadas sus dimensiones y su potencia
No es del todo orientado a objetos	Casi todo está orientado a objetos	Puramente orientado a objetos	Puramente orientado a objetos
No soportado en Android/iphone	No soportado en Android/iphone	Soportado en Android/iphone	Soportado en Android/iphone

Es cierto que las automatizaciones de la interfaz son imprescindibles, pero no son las que necesariamente más retorno de inversión aporta, ni las que deberían automatizarse en mayor cantidad.
La interfaz de usuario es la parte más propensa a cambios de toda la aplicación, y para automatizar y tener fiabilidad sobre lo que estamos ejecutando necesitamos cierta estabilidad en las pruebas. Un cambio en la interfaz podría hacer fallar la prueba automática, y en ese caso, tendríamos que volver a adaptarla para que volviera a funcionar.
Para decidir si automatizar o no, debemos tener en cuenta varios factores:

- Coste de la automatización: Debemos tener en cuenta el trabajo que supone la automatización y si va a proporcionar beneficios en el futuro ya que, si conlleva mucho tiempo automatizar una prueba que dura 2 minutos y que solo se va a ejecutar unas 10 veces en el futuro, puede que no sea una buena idea. Generalmente, las pruebas que se pueden automatizar de forma rápida y fácil son las mejores candidatas, mientras que las pruebas complejas cuya automatización lleva mucho tiempo no son generalmente una buena elección.

- Cambio de la funcionalidad en un futuro: Si hay una alta probabilidad de que la funcionalidad a probar sufra muchos cambios, no tiene mucho sentido realizar el trabajo de automatizar ya que no servirá para el futuro.

- Frecuencia de Uso: En general, las partes más utilizadas de una aplicación no son rentables para la automatización debido a los costes de mantenimiento que pueden suponer si los usuarios quieren introducir cambios en ellas. No obstante, si la funcionalidad se usa a menudo y se mantiene estable, sí puede convertirse en una buena candidata para las pruebas automatizadas.

- Pruebas de regresión: Detectar errores en partes de la aplicación que no han sido modificadas es una 9de las razones principales para la creación de pruebas automatizadas. Naturalmente, los errores de regresión pueden encontrarse en cualquier parte, pero es más probable que se detecten en las partes centrales de la aplicación que son compartidas por distintas funcionalidades que tienen la misma lógica de negocio. En este caso, la mejor forma de actuar es crear un conjunto de pruebas unitarias automatizadas.

- Versionado del software: Esto está ligado directamente con la posibilidad de cambios futuros en la aplicación. Si solo se mantienen una o dos versiones de la aplicación en producción, la automatización es recomendable. Si se mantienen más, es conveniente centrarse más en las pruebas manuales, debido a los costes de mantenimiento ya que pueden llegar a ser bastante altos.

- Funcionalidad crítica dentro de la aplicación: Las partes críticas de la aplicación deberían probarse cada vez que se realiza cualquier cambio en la misma. Por tanto, son unas estupendas deben ser automatizadas, ya que se ejecutarán muy a menudo.

2.3. Ventajas de la automatización

1. Los costes operativos disminuyen considerablemente. Los procesos no requieren de personal que realice tareas mecánicas.
2. Se eliminan los errores humanos. Como las tareas se automatizan y se programan para determinados patrones, despistes o fallos provenientes de las personas desaparecen totalmente de la ecuación, dando más eficiencia y fiabilidad a los resultados.
3. Los empleados se centran en lo importante. Con más tiempo "libre", el personal de la empresa se puede dedicar a explotar sus capacidades intelectuales, a aportar conocimiento para buscar nuevas estrategias de mejora y crecimiento, lo que hará al negocio mucho más competente de cara al mercado.
4. Los sistemas trabajan sin horario e interrupciones y su tráfico de datos ayuda a disponer de mejores análisis, que beneficiarán la comprensión e interpretación de la realidad con respecto a la oferta y demanda de productos y servicios.
5. El control sobre los procesos que maneja la automatización es mayor y los fallos que detecta el sistema se dan a conocer instantáneamente.

2.4. Desventajas de la automatización

1. Incertidumbre laboral. Es una preocupación para la plantilla si la automatización eliminará ciertos trabajos. Contrariamente a esto, automatizar procesos aporta la capacidad de crear nuevos puestos, ya que los empleados pueden abordar muchos más proyectos que antes no alcanzaban a llegar por falta de tiempo.
2. Alta inversión inicial. Antes de que, a nivel económico, esta tecnología comience a ser rentable, hay que realizar un primer esfuerzo económico. Si la implementación se desenvuelve correctamente, se amortizará la inversión inicial, pero antes de lanzarse hay que analizar el retorno de la inversión (ROI) del proyecto y la gestión de presupuesto del que se dispone.
3. Gestión de cambios. Contar con esta herramienta supone integrar procesos de control de calidad, que suponen cierto grado de esfuerzo y tiempo. Esto implica estar pendiente de actualizaciones y análisis y retroalimentación de lo que aporta este nuevo sistema, para que su utilidad y beneficios no se queden estancados y evolucionen en cuanto a funcionalidades en base a las necesidades de la compañía.

2.5. Automatización de pruebas de la interfaz

Probablemente lo que primero suele venirnos a la cabeza cuando oímos hablar de automatización de pruebas son automaciones de “record & play”, por ejemplo con herramientas tipo Selenium IDE, que te permiten simular y grabar tus interacciones con la interfaz de la plataforma, con el navegador web etc.
Depende de qué plataforma, a primera vista pueden resultar las más sencillas de automatizar.
También podemos hacer uso de otros tipos de Selenium como Webdriver, Grid o RC, aunque estas necesitan de un conocimiento mayor de programación y de la librería, pero que conlleva a una automatización más precisa y sin fallas.

2.6. El framework Selenium

2.6.1. Selenium IDE 

Selenium IDE es un plugin que tiene opciones de grabación y reproducción con una interfaz de usuario fácil de entender. Su core está programado en JavaScript y admite diferentes lenguajes de programación.
Esta herramienta fue creada para Firefox pero también existe en otros navegadores como Google Chrome.
En la actualidad, ha dejado de actualizarse a partir de la versión 55 de Mozilla por lo que habría que hacer uso de versiones más antiguas aunque ya existe para Chrome (véase en la siguiente imagen).

Una de las razones por las que un automatizador hace uso de este complemento es la facilidad de obtención del código, ya que permite la exportación del código a varios lenguajes como Java. No obstante, puede que el uso de las distintas funciones resulte inservible para una futura automatización, ya que se usan métodos sin tener en cuenta, por ejemplo, si está oculto el elemento o no.
Tambien existen otras herramientas que hacen uso de este tipo de Selenium, cómo por ejemplo Katalon, que también permite la grabación y la ejecución de los tests.

Definitivamente, no es una buena opción para crear una versión final de la automatización, pero si para empezar a crear la prueba, aunque requiere de modificaciones.
Podemos hallar información en la web oficial de Selenium o existen cursos muy completos en las webs como Udemy o devacademy.

2.6.2. Selenium RC

Selenium RC (Selenium Remote Control) se utiliza para ejecutar los distintos scripts en diferentes navegadores. Su compatibilidad se extiende a diferentes browsers como IE, Firefox, Chrome, Safari, Opera, etc. También es compatible con múltiples lenguajes como Java, Ruby, C #, Perl, Python, etc. La aplicación está desarrollada en C # pero su uso puede automatizarse con java por ejemplo. Al igual que el lenguaje independiente, también es una plataforma independiente, el mismo código funcionará en el sistema operativo Windows, Linux, Mac y Solaris. La extensión más común utilizada en Selenium RC es la extensión de Java, porque Java es un lenguaje independiente de la plataforma. Similar al Selenium IDE, el RC también tiene sus limitaciones. Antes de comenzar la prueba, tenemos que iniciar y detener el servidor para ejecutar su prueba.

Podemos hallar información en la web oficial de Selenium o existen cursos muy completos en las webs como Udemy o devacademy.

2.6.3. Selenium Webdriver

Para superar todos los problemas y aumentar el alcance de Selenium IDE y RC, introdujo una nueva versión de SE llamada Selenium WebDriver. Soporta los múltiples lenguajes, no tenemos que iniciar el servidor, es compatible con las Pruebas de Android y las pruebas de iPhone.
El código de WebDriver tiene un aspecto diferente al de RC & IDE, le permite convertir el código IDE a código de WD & RC. Como IDE es compatible con la interfaz de usuario, pero WebDriver & RC no tienen una interfaz de usuario, tenemos que usar un lenguaje de programación central en ella.
Por ello, esta forma de automatizar es la mejor, ya que podemos solventar los problemas de los objetos ocultos, pero eso sí, precisamos de un conocimiento profundo del framework selenium.

Podemos hallar información en la web oficial de Selenium o existen cursos muy completos en las webs como Udemy o devacademy.

2.6.4. Selenium Grid:

Es una herramienta para ejecutar test de Selenium de forma distribuida. Utiliza internamente Selenium Remote Control para ejecutar los test de integración con distintos navegadores y plataformas.
Se especializa en ejecutar múltiples pruebas a través de diferentes navegadores, sistemas operativo y máquinas. Puede conectarse con Selenium Remote especificando el navegador, la versión y el sistema operativo que desee. Hay dos elementos principales: hub y nodos.
Dentro de Selenium Grid se encuentra un módulo importante, el Hub. Este módulo es la parte central de Selenium Grid ya que nos permitirá controlar los distintos Servidores Remote Control encargados de lanzar los test y distribuir los test en los mismos.
Una vez montada la infraestructura para los test crearemos un test con JUnit 4 y Selenium con la particularidad de que podrán ser lanzados de forma paralela, es decir completamente simultánea en varios hilos. Con esto conseguiremos test de carga con el

que poder medir los tiempos de respuesta de nuestro sistema. También conseguiremos distribuir nuestros test de Selenium en varias máquinas y navegadores ganando en eficiencia con test demasiado pesados.

Podemos hallar información en la web oficial de Selenium o existen cursos muy completos en las webs como Udemy o devacademy.

2.7. Pros y contras de la automatización con selenium

2.7.1. Pros

Hay que tener en cuenta una serie de consideraciones que sirven de precedente a la hora de automatizar, como el coste que conlleva la automatización:

Selenium está escrito en su mayoría sobre un lenguaje que nativo de los browsers, y las pruebas se automatizan en los distintos lenguajes de programación como puede ser Java, javascript o python, ofrecen una gran portabilidad.
Respecto a la velocidad de ejecución y de realización de pruebas, tenemos que, como testers, plantearnos si es necesaria una automatización o no, ya que dependiendo de:

- Posibles regresivos debidos a cambios futuros.
- Dificultad de realización de las pruebas.

Testing automatizado	**Testing manual**
Si se quiere ejecutar un plan de pruebas repetidamente, la automatización será de muchísima ayuda.	Si el caso de prueba sólo se va a ejecutar dos veces ante un cambio en la codificación, probablemente debería ser una prueba manual antes que automatizada ya que el costo es mayor que su beneficio.
Nos permite lanzar las pruebas automatizadas sobre un código que cambia frecuentemente reduciendo el tiempo en las pruebas de regresión.	Permite al tester ampliar más las pruebas durante la ejecución del caso de prueba. Encontrando más bugs ya que éste puede inventar combinaciones impensadas mientras navega la aplicación y ante diversas situaciones.
Permite realizar matrices de pruebas combinando diferentes lenguajes con diferentes SO.	Imposible la realización de matrices de pruebas.
Permite realizar pruebas en paralelo en una o varias máquinas.	No permite realizar las pruebas en paralelo, depende del número de testers que haya, por lo que aumenta el gasto.

CONTRAS:
Hay que tener en cuenta una serie de consideraciones que sirven de precedente a la hora de automatizar, como el coste que conlleva la automatización, que depende del conocimiento que se tenga sobre la herramienta que se utilizará, la experiencia en programación, tiempo de las pruebas, cantidad de ciclos de testing, etc.

Si son pruebas en las que el gasto de tiempo es mayor automatizando que ejecutando manualmente o se van a ejecutar pocas veces, no merece la pena el gasto.
En la siguiente tabla, se describen los desventajas de los dos tipos de testing.

Testing automatizado	Testing manual
El esfuerzo inicial es mayor. La creación del script automatizado puede ser más costoso que la creación del caso de prueba para ejecución manual.	No precisa de costes para crear los casos de prueba automatizados, por lo que el coste es menor.
No se pueden (o puede ser muy costoso) automatizar referencias visuales como colores o ubicación en pantalla de objetos.	Las pruebas manuales pueden consumir demasiado tiempo.
No existe desventaja porque ya están previamente automatizadas, por lo que el regresivo se lanza sin problemas.	Cada vez que hay un cambio en la aplicación el tester debe lanzar algunas pruebas de regresión. Ante reiterados cambios, el tester puede llegar a ejecutar demasiadas veces estas pruebas, reduciendo su capacidad de encontrar errores.

3. Objetivos del proyecto

El objetivo general de este proyecto es la identificación de casos problemáticos reales a la hora de automatizar con la librería Selenium haciendo uso de buenas prácticas para así encontrar una solución rápida que faciliten la realización de testing automatizado, tanto desde la perspectiva del desarrollador como del tester.

Objetivo 1: Identificar situaciones habituales que inducen a error o a mala práctica.
Objetivo 2: Proponer recomendaciones para solucionar estas situaciones.
Objetivo 3: Crear una guía que ayude a desarrolladores y testers en el uso del framework.

4. Estrategia de realización del proyecto

La estrategia a seguir ha sido la siguiente:

1. Iniciación: Búsqueda extensa de información para conocer el estado actual de la herramienta, explorando a fondo las posibilidades que tiene la librería, buscando en experiencias de otros usuarios a través de foros, entendiendo el funcionamiento de la herramienta.
2. Planificación: Una vez conocido el estado de la herramienta, se procede a organizar el proyecto con los casos que provocan problemas para así dar soluciones para que la persona que use esta guía pueda adquirir el conocimiento suficiente y así afrontarlos con facilidad. La previa identificación de los problemas haciendo uso de la observación del código fuente de la página web para así encontrar los casos problemáticos que he considerado "peligrosos" a la hora automatizar. Leyendo experiencias a través de internet, he concretado las situaciones problemáticas que suele tener el desarrollador de manera cotidiana.

   La problemática se centra en el uso incorrecto y el desconocimiento de la librería principalmente, que muchas veces es causado por una mala programación de la página web.
   Una vez identificados los problemas, los he categorizado (tipo 1, 2, 3, etc.) y posteriormente, he comenzado a realizar la ejecución con sus respectivos resultados que he reflejado a modo de guía en el siguiente punto.

3. Ejecución: Se ejecutan los casos problemáticos para así poder mostrar con snippets de código las posibles soluciones. También se ha hecho uso de automatizaciones obtenidas en el de automatizadores expertos con conocimientos exhaustivos de la herramienta.

5. Estudio del framework Selenium

5.1. Problemas en el uso de Selenium

Siguiendo la estrategia descrita en el anterior punto, se ha realizado una identificación de los problemas principales que suele provocar el uso del framework Selenium. Los problemas se han categorizado en varios tipos:

- Identificadores de elementos: Los métodos de búsqueda de los elementos en el código fuente, siendo según el caso, se hará uso de unos u otros.
- Objectos ocultos: Los métodos para la obtención de los objetos no presentes en el código fuente.
- Mantenibilidad y reusabilidad: Descripción de la manera correcta de realizar una automatización que perdure en el tiempo.
- Tiempos de espera: Explicación de los problemas que provocan un mal uso de los tiempos de carga de los objetos.

En las siguientes secciones se describen los problemas concretos dentro de cada categoría, y se propones soluciones a los mismos.

5.2. Identificadores de elementos

En el código fuente de las webs, nos encontramos con varias formas de identificar los elementos.

- Id
- Name
- DOM
- XPath
- Link
- CSS

Debemos prestar especial cuidado a que el identificador elegido sea reutilizable en el futuro: por ejemplo, debe evitarse la elección de un identificador que pueda cambiar con cada nueva versión de la aplicación. A continuación, repasamos los tipos de identificadores existentes, y como deben ser utilizados.

Es muy importante elegir de manera adecuada los identificadores que vamos a usar en cada caso:

5.2.1. Id

5.2.1.1 Uso adecuado

Muchos elementos en el código fuente de la página web están identificados con una variable (id=comp-ih3hrpn1label).

A la hora de elegir este método, tenemos que asegurarnos que la aplicación tiene bien definidos estos identificadores, es decir, que sean únicos e invariantes en el tiempo, para así poder de ellos para buscar el elemento deseado. En este caso, el mantenimiento de las pruebas en el tiempo no existiría, porque añadir o quitar elementos a la página no afectaría al 'id' del resto ya que siempre debería ser el mismo. Esta es la mejor opción siempre que se dé este escenario.

Un ejemplo de uso de Id podría ser el siguiente:

Código fuente HTML:

```
<span id="comp-ih3hrpn1label" class="b1label">COMPRAR</span>
```

Dependiendo de la variante de Selenium usado, el Id anterior puede invocarse de distintas formas. A continuación, se muestra un posible uso mediante Webdriver:

Código Selenium Java

```
driver.findElement(By.id("comp-ih3hrpn1link")).click();
```

5.2.1.2 Uso problemático

En una página web, pueden existir objetos con ids dinámicos porque estos cambian de lugar o son eliminados. Por ejemplo, en una tienda online los productos cambian continuamente. Esto es problemático ya que si en un primer momento tenemos el valor del Id, cuando volvamos a ejecutar esa prueba en un regresivo, nos encontraremos con otro valor en el id, por lo que la prueba no funcionará al no encontrar el elemento (¿Cómo hacer una actualización forzada de la página usando Selenium Webdriver?, 2019). Un ejemplo de esta problemática es el siguiente:

Primera vez que automatizamos:

Código fuente HTML:

```
<span id="comp-ih3hrpn1label" class="b1label">COMPRAR</span>
```

Código Selenium Java

```
driver.findElement(By.id("comp-ih3hrpn1label")).click();
```

Código fuente HTML con el Id Cambiado:

```
<span id="comp-ivxhrpn2label" class="b1label">COMPRAR</span>
```

Como se puede observar en el cuadro anterior, al recargar la página, el id del elemento sería otro. Por lo que la prueba no funciona. Este escenario no sería adecuado para hacer uso del id (Selenium WebDriver ¿Cómo resolver la excepción de referencia de elementos obsoletos?, 2018).

5.2.2. Name

El método name es otro atributo que referencia a objetos de manera distinta al Id, siendo normalmente un texto que representa al nombre del objeto a buscar.

5.2.2.1 Uso adecuado

El uso de este método se reduce a conocer el nombre del objeto a buscar. A continuación, se describe un ejemplo:

Código fuente HTML:

```
<input aria-label="Dirección de email" class="field__input ng-pristine ng-untouched ng-invalid ng-invalid-required ng-valid-pattern" name="subscriberEmail" ng-model="main.subscriberEmail" ng-pattern="/^\b[a-zA-Z0-9.!#$%&amp;amp;'*+\/=?^_`{|}~-]{1,50}@[a-zA-Z0-9-]{1,50}(?:\.[a-zA-Z0-9-]{2,20})+$/" placeholder="Dirección de email" required="" size="1">
```

Código Selenium Java

```
driver.findElement(By.name("subscriberEmail")).sendKeys("hola@gmail.com");
```

5.2.2.2 Uso problemático

Generalmente, el atributo 'name' de un elemento HTML no suele ser único, de hecho suele ser variante por lo que el uso de este método no garantiza que la prueba se ejecute de la manera deseada. De hecho, en funciones como “getText”, “click” o “type” que sólo referencian a un elemento, la acción se realizaría siempre sobre el primer elemento encontrado. A continuación, se detalla un ejemplo:

Código fuente HTML:

```
<input aria-label="Dirección de email" class="field__input ng-pristine ng-untouched ng-invalid ng-invalid-required ng-valid-pattern" name="subscriberEmail" ng-model="main.subscriberEmail" ng-pattern="/^\b[a-zA-Z0-9.!#$%&amp;’*+\/=?^_`{|}~-]{1,50}@[a-zA-Z0-9-]{1,50}(?:\.[a-zA-Z0-9-]{2,20})+$/" placeholder="Dirección de email" required="" size="1">

<input aria-label="Dirección de email" class="field__input ng-pristine ng-untouched ng-invalid ng-invalid-required ng-valid-pattern" name="subscriberEmail" ng-model="main.subscriberEmail" ng-pattern="/^\b[a-zA-Z0-9.!#$%&amp;’*+\/=?^_`{|}~-]{1,50}@[a-zA-Z0-9-]{1,50}(?:\.[a-zA-Z0-9-]{2,20})+$/" placeholder="Dirección de email" required="" size="1">
```

Código Selenium Java

```
driver.findElement(By.name("subscriberEmail")).sendKeys("hola@gmail.com");
```

Si quisiéramos obtener el segundo “subscriberEmail”, no lo obtendríamos de esta forma ya que siempre devolvería el primero.

Esta opción es muy poco recomendable y no sería válida para hacer click sobre un conjunto de elementos con el mismo nombre (Selenium WebDriver ¿Cómo resolver la excepción de referencia de elementos obsoletos?, 2018).

5.2.3. DOM

5.2.3.1 Uso adecuado

DOM (Document Object Model) es un modelo de datos que sirve para hacer referencia a varios elementos del código fuente. Seguidamente, se describe un ejemplo (Page Object Model (POM) & Page Factory: Selenium WebDriver Tutorial, 2019):

Código fuente HTML:

```
<h2 class="_2Znzd" data-hook="cart-title">Mi carrito</h2>
//Estos son los elementos
<h3 id="item_sku_1" data-hook="product-name">EZ 00001</h3>
<h3 id="item_sku_1" data-hook="product-name">EZ 00002</h3>
```

Código Selenium Java

```
//Obtenemos el primer elemento
document.getElementsById("Mi carrito")[1]
```

5.2.3.2 Uso problemático

El problema reside en que, al introducir un nuevo elemento, o reorganizar los existentes, da lugar a que las referencias cambien, lo que puede invalidar las pruebas automatizadas anteriormente (Selenium WebDriver ¿Cómo resolver la excepción de referencia de elementos obsoletos?, 2018). A continuación, se muestra un ejemplo:

Código fuente HTML:

```
<h2 class="_2Znzd" data-hook="cart-title">Mi carrito</h2>
//Estos son los elementos
<h3 id="item_sko_1" data-hook="product-name">EZ 00001</h3>
<h3 id="item_sko_1" data-hook="product-name">EZ 00002</h3>
<h3 id="item_sko_1" data-hook="product-name">EZ 00003</h3>
```

5.2.4. XPath

5.2.4.1 Uso adecuado

Esta forma de identificación de elementos hace uso de la estructura XML que posee todo documento HTML, haciendo referencia a los elementos mediante una ruta. existen dos formas de uso: ruta absoluta (partiendo desde el elemento padre '/') o ruta relativa (partiendo de un elemento hijo). A continuación, se detalla un ejemplo:

Código fuente HTML:

```
//Path absoluto
<a aria-label="Carrito con 2 items" class="_1gJ4T _3UFsv _3tSTQ" data-hook="cart-icon-button" href="https://jorx35.wixsite.com/misitio/cart" role="button"><div class="kOc8C" style="padding-bottom:20.833333333333336%;visibility:visible" data-hook="svg-icon-wrapper"><svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="100%" height="100%" viewBox="0 0 504 105" preserveAspectRatio="xMinYMax meet" data-hook="svg-icon-3"><g data-hook="movable" transform="translate(384)"><circle class="m8g_G" cx="70" cy="50" r="50"></circle><text x="70" y="50%" dy=".32em" text-anchor="middle" class="_1INsc" data-hook="items-count">2</text></g><text x="0" y="50%" dy=".35em" data-hook="free-text">CARRITO</text></svg></div></a>
```

Código Selenium Java

```
//Path absoluto
driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Carrito'])[1]/preceding::span[1]")).click();

//Path relativo

xpath=//span[@id='comp-ih3hrpn1label']
```

5.2.4.2 Uso problemático

En este caso, nos encontramos el mismo problema que el localizador por DOM ya que dependen de la estructura del documento. Además con Xpath se recomienda hacer uso de la ruta absoluta ya que devolverá un elemento único, mientras que en el caso de rutas relativas no se cumple ya que una misma ruta puede ser válida para varios elementos. A continuación, se expone un ejemplo:

Código fuente HTML:

```
//Path absoluto
<a aria-label="Carrito con 2 items" class="_1gJ4T _3UFsv _3tSTQ" data-hook="cart-icon-button" href="https://jorx35.wixsite.com/misitio/cart" role="button"><div class="kOc8C" style="padding-bottom:20.833333333333336%;visibility:visible" data-hook="svg-icon-wrapper"><svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="100%" height="100%" viewBox="0 0 504 105" preserveAspectRatio="xMinYMax meet" data-hook="svg-icon-3"><g data-hook="movable" transform="translate(384)"><circle class="m8g_G" cx="70" cy="50" r="50"></circle><text x="70" y="50%" dy=".32em" text-anchor="middle" class="_1INsc" data-hook="items-count">2</text></g><text x="0" y="50%" dy=".35em" data-hook="free-text">CARRITO</text></svg></div></a>
```

Código Selenium Java

```
//Path absoluto
driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Carrito'])[1]/preceding::span[1]")).click();
//Path relativo
xpath=//span[@id='comp-ih3hrpn1label']
```

Si cambiara la localización del elemento, la automatización quedaría inservible al no poder encontrarlo.

5.2.5. Link

5.2.5.1 Uso adecuado

Este método se usa normalmente para localizar un enlace. Su uso requiere conocer el texto que va a mostrar dicho enlace en la página HTML

A continuación, se expone un ejemplo:

Código fuente HTML:

```
<a class="button-learn-more" data-hook="learn-more-button" translate="LEARN_MORE_BUTTON" section-link="moreInfoVM.getProductSectionId()" section-params="moreInfoVM.productUrlPart" section-post-click="moreInfoVM.closeWindow()" href="https://jorx35.wixsite.com/misitio/product-page/ez-00001">Ver todos los
```

```
detalles</a>
```

Código Selenium Java

```
driver.findElement(By.linkText("Ver todos los detalles")).click();
```

5.2.5.2 Uso problemático

Puede no ser útil en caso de existir enlaces con el mismo texto ya que devolvería el primero de ellos.

Se desaconseja el uso de este método.

5.2.6. CSS

5.2.6.1 Uso adecuado

Este localizador consiste en identificar los elementos por sus propiedades de CSS.

Código fuente HTML:

```
<a aria-label="Carrito con 2 items" class="_1gJ4T _3UFsv _3tSTQ" data-hook="cart-icon-button" href="https://jorx35.wixsite.com/misitio/cart" role="button"><div class="kOc8C" style="padding-bottom:20.833333333333336%;visibility:visible" data-hook="svg-icon-wrapper"><svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="100%" height="100%" viewBox="0 0 504 105" preserveAspectRatio="xMinYMax meet" data-hook="svg-icon-3"><g data-hook="movable" transform="translate(384)"><circle class="m8g_G" cx="70" cy="50" r="50"></circle><text x="70" y="50%" dy=".32em" text-anchor="middle" class="_1INsc" data-hook="items-count">2</text></g><text x="0" y="50%" dy=".35em" data-hook="free-text">CARRITO</text></svg></div></a>
```

Código Selenium Java

```
//Hacemos click sobre el botón CSS carrito
driver.findElement(By.cssSelector("svg > text")).click();
```

5.2.6.2 Uso problemático

El uso de este método no es aconsejable ya que la misma propiedad normalmente referencia a varios elementos, por lo que no garantiza la unicidad del elemento referenciado. A continuación, se detalla un ejemplo:

Código fuente HTML:

```
<a aria-label="Carrito con 2 items" class="_1gJ4T _3UFsv _3tSTQ" data-hook="cart-icon-button" href="https://jorx35.wixsite.com/misitio/cart" role="button"><div class="kOc8C" style="padding-bottom:20.833333333333336%;visibility:visible" data-hook="svg-icon-wrapper"><svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="100%" height="100%" viewBox="0 0 504 105" preserveAspectRatio="xMinYMax meet" data-hook="svg-icon-3"><g data-hook="movable" transform="translate(384)"><circle class="m8g_G" cx="70" cy="50" r="50"></circle><text x="70" y="50%" dy=".32em" text-anchor="middle" class="_1INsc" data-hook="items-count">2</text></g><text x="0" y="50%" dy=".35em" data-hook="free-text">COMPRAR</text></svg></div></a>
```

Código Selenium Java

```
driver.findElement(By.cssSelector("svg > text")).click();
```

En el ejemplo mostrado, podemos observar que ‘COMPRAR’ tiene las mismas propiedas que ‘CARRITO’ que es el elemento buscado, por lo que como resultado obtendremos el elemento no deseado.

5.3. Objetos Ocultos

Unos de los problemas principales del Selenium es que esta librería no contempla de manera predeterminada la obtención de objetos ocultos.

5.3.1 Uso adecuado

El objetivo es hacer visible el objeto a ejecutar para poder automatizar la prueba sin ningún problema, ya que siempre que se ejecute el elemento estará visible y disponible.
Es por esto por lo que necesitamos conocimientos en programación para poder realizar una automatización correcta y que luego nos pueda servir en el futuro. A continuación, se describe un ejemplo:

Código fuente HTML:

```
<div class="_3uack __web-inspector-hide-shortcut__"><span class="kpb5P" data-hook="sr-product-item-price-to-pay">Precio</span><span data-hook="product-item-price-to-pay" class="_23ArP">200,00 €</span></div>
```

Código Selenium Java

```
JavascriptExecutor js = (JavascriptExecutor)driver;
js.executeScript("arguments[0].click();", element);
driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Precio'])[1]/following::a[1]")).click()
```

5.3.2 Uso problemático

Cuando exportamos el código desde la aplicación IDE Selenium y lo importamos en nuestro proyecto, no tiene en cuenta estos elementos porque el objeto no está visible. Para solucionar este problema, debemos hacer uso de métodos javascript para hacerlos visibles y para poder hacer click en ellos o encontrarlos.
Este es uno de los casos que los automatizadores sin experiencia no controlan y se atascan, ya que no conocen el uso de javascript dentro de java o directamente no usan java ni js.
Para clarificar, el siguiente fragmento de código ha sido extraído de la automatización con selenium IDE donde el objeto está oculto. A continuación, se expone un ejemplo:

Código fuente HTML:

```
<div class="_3uack"><span class="kpb5P" data-hook="sr-product-item-price-to-pay">Precio</span><span data-hook="product-item-price-to-pay" class="_23ArP">200,00 €</span></div>
```

Código Selenium Java

```
driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Precio'])[1]/following::a[1]")).click()
```

5.4. Tiempos de espera

Otro de los problemas que nos solemos encontrar a la hora de automatizar con selenium, es el uso los tiempos de espera.
El uso incorrecto de éstos, puede provocar que la velocidad de ejecución aumente en caso de establecer un tiempo muy alto e innecesario o también nos podemos encontrar con que su valor es menor al tiempo de carga, por lo que fallará debido a que el elemento no ha

sido cargado y se producirá una excepción «ElementNotVisibleException» (Waiting after web page interactions | QA Automation, 2012).

5.4.1. Espera de transiciones de la página

Este caso es usado cuando se necesita hacer uso de la "redirección", es decir, cuando hacemos click en una imagen y esperamos a que se cargue para comprobar que la nueva web se abre correctamente y es la página buscada.

5.4.1.1. Uso adecuado

Para solucionar este problema, podemos hacer uso de este tipo de espera, ya que sólo avanzará la ejecución cuando el elemento se haya cargado. A continuación, se detalla un ejemplo:

Código Selenium Java

```
WebDriverWait wait = new WebDriverWait(webDriver, 10);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("Mi carrito")));
```

5.4.1.2. Uso problemático

Puede darse el caso que el elemento no cargue, por fallo de conexión o por no existencia del objeto a buscar.

5.4.2. Espera a la carga con sleep

Este método es una forma de esperar a que el elemento se cargue parando el hilo de ejecución (Webdriver Tiempos de espera de carga, 2011).

5.4.2.1. Uso adecuado

Se descarta el uso de este método por ineficiencia del resultado y por no asegurar nunca la carga del elemento.

5.4.2.2. Uso problemático

Este método es una mala práctica ya que la espera no está dirigida directamente al elemento, porque el programador fija un tiempo customizado sin conocer realmente lo que debemos esperar. A continuación, se detalla un ejemplo:

Código Selenium Java

```
Thread.sleep(2000);
driver.findElement(By.id("comp-ih3hrpn1link")).click();
```

5.5. Mantenibilidad y reusabilidad

Para hacer uso de buenas prácticas teniendo el código limpio con facilidad de encontrar errores y entendible, podemos hacer uso del Page Object Model(POM):
En pocas palabras, POM es una forma hacer más mantenible el código de dos formas:

1. Creando objetos con los datos que más suelen cambiar.
2. Creando métodos que realicen acciones determinadas.

En el primer caso, el objetivo es estructurar el código focalizando los datos cambiantes del código fuente para que en un futuro podamos realizar el mantenimiento actualizando los mismo. Este método de automatización requiere un previo estudio del código fuente por lo que se necesita más tiempo, pero en un futuro, esta pérdida se recuperará.

```
public class MisitioWeb{

 WebDriver driver;

 String homePageUserName =“driver.get("https://jorx35.wixsite.com/misitio")”;

   public MisitioWeb(WebDriver driver){
      this.driver = driver;
   }
   //Get the User name from Home Page
      public String getHomePageUserName(){
       return    driver.findElement(homePageUserName).getText();
       }
}
```

En este snippet de código podemos observar un método que realiza una acción determinada. Imaginemos que hubiera mil líneas de código, y en él no existiera este método. El código sería inmantenible y perderíamos tiempo en la ejecución.
El fragmento de código es usado en todas las pruebas, porque abre el navegador y se conecta a la web deseada, pero si creamos una función como la siguiente, que haga un lo mencionado, no haría falta repetir código :

```
 @Before
 public void setUp() throws Exception {
```

```
            System.setProperty("webdriver.chrome.driver",
                                    "D:\\TFG\\chromedriver.exe");
            driver = new ChromeDriver();
            baseUrl = "https://jorx35.wixsite.com/misitio";
            driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}
```

Si esto fallara, se focalizaría el fallo dentro del mismo y sería fácil de encontrar y solucionar el error, convirtiéndo así el código en mantenible y reusable.
Por supuesto, esta forma es la mejor, pero también requiere de un gran gasto, ya que se precisa de profesionales con conocimientos en java (Tutorial esperas (Waits) en Selenium WebDriver - Tutorial Selenium, 2018)
.

6. Buenas prácticas

Para obtener buenos resultados de la automatización, el programador debe seguir una serie de recomendaciones:
Hacer el código legible, utilizando métodos claros, devuelve siempre “this”, usa métodos “void”. Devuelve argumentos genéricos y con nombres auto-descriptivos y claros.
Haz que el código sea robusto y portable haciendo uso de la búsqueda de los elementos de manera correcta y precisa.
Evita hacer uso de Thread.Sleep, para no esperar más de lo preciso provocando pérdidas de tiempo o para no esperar menos dando lugar a no encontrar el elemento.
Usando URL’s relativas, evitamos las URL’s concretas, ya que normalmente cambian. Por ejemplo, la ruta concreta de una web “https://jorx35.wixsite.com/misitio/buy-now/carrito&200;” provocará problemas en el futuro ya que el carrito es cambiante.
Mantén tu proyecto actualizado, manteniendo la versión de Selenium actualizada y el navegador.
El uso de métodos para realizar tareas sencillas mejora la localización de los posibles errores en la automatización (Martin, 2009).

7. Conclusiones

El objetivo de la automatización de pruebas no es eliminar por completo el testing manual, ni suplantar a los testers manuales.
Debemos tener en cuenta, que las pruebas que se automatizan provienen de un chequeo previo por parte de los testers manuales.
Incluso puede haber ocasiones en las que automatizar alguna prueba o generar las condiciones necesarias para ello sea tan costoso, que sea más rentable mantenerla manual.
A modo personal, el proyecto que he realizado me ha aportado un gran conocimiento del testing de software y me ha hecho entender que no sólo vale con programar una prueba, si no que hay que automatizarla teniendo en cuenta muchos factores que antes no conocía.

8. Referencias bibliográficas

¿Cómo hacer una actualización forzada de la página usando Selenium Webdriver? (2019, 8 junio). Lugar de publicación: https://codeday.me/es/qa/20190608/831900.html

Martin, R. C. (2009). Clean Code: A Handbook of Agile Software Craftsmanship (Ed.

rev.). New York, EE UU: Prentice Hall.

Page Object Model (POM) & Page Factory: Selenium WebDriver Tutorial. (2019, 30 mayo). Lugar de publicación: https://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html

Selenium WebDriver ¿Cómo resolver la excepción de referencia de elementos obsoletos? (2018, 25 junio). Lugar de publicación: https://es.switch-case.com/55861164

Tutorial esperas (Waits) en Selenium WebDriver - Tutorial Selenium. (2018, 14 enero). Lugar de publicación: https://www.tutorialselenium.com/2018/01/14/waits-en-selenium-webdriver/

Waiting after web page interactions | QA Automation. (2012, 16 marzo). Lugar de publicación: http://www.qaautomation.net/?p=490

Webdriver. (2011, 30 mayo). Lugar de publicación: https://rstopup.com/como-puedo-esperar-para-que-una-pagina-se-carga-despues-de-un-formulario-se-envia-en-webdriver-estoy-usando-firefox-conductor.html

9. Anexo

```
1. package tfg;
2. import java.util.regex.Pattern;
3. import java.util.concurrent.TimeUnit;
4. import org.junit.*;
5. import static org.junit.Assert.*;
6. import static org.hamcrest.CoreMatchers.*;
7. import org.openqa.selenium.*;
8. import org.openqa.selenium.chrome.ChromeDriver;
9. import org.openqa.selenium.firefox.FirefoxDriver;
10.     import org.openqa.selenium.support.ui.Select;
11.
12.     public class CarritoVacio {
13.             private WebDriver driver;
14.             private String baseUrl;
15.             private boolean acceptNextAlert = true;
16.             private StringBuffer verificationErrors = new StringBuf
    fer();
17.
18.             @Before
19.             public void setUp() throws Exception {
20.                     System.setProperty("webdriver.chrome.driver",
21.                                     "D:\\TFG\\chromedriver.exe");
22.                     driver = new ChromeDriver();
23.                     baseUrl = "https://www.katalon.com/";
24.                     driver.manage().timeouts().implicitlyWait(30,
    TimeUnit.SECONDS);
25.             }
26.             @Test
27.             public void testCarritoVacio() throws Exception {
28.                     driver.get("https://jorx35.wixsite.com/misitio/
    buy-now");
29.                     driver.findElement(By.cssSelector("svg >
    text")).click();
30.                     driver.findElement(By.xpath("(.//*[normalize-
    space(text()) and normalize-
    space(.)='Carrito'])[1]/following::div[1]")).click();
31.                     try {
32.                             assertEquals("El carrito está vacío",
33.     driver.findElement(By.xpath("(.//*[normalize-space(text()) and
    normalize-space(.)='Carrito'])[1]/following::span[1]")).getText());
34.                     } catch (Error e) {
35.                             verificationErrors.append(e.toString())
    ;
36.                     }
37.             }
38.
39.             @After
40.             public void tearDown() throws Exception {
41.                     driver.quit();
42.                     String verificationErrorString = verificationEr
    rors.toString();
43.                     if (!"".equals(verificationErrorString)) {
44.                             fail(verificationErrorString);
45.                     }
46.             }
47.
```

```
48.             private boolean isElementPresent(By by) {
49.                     try {
50.                             driver.findElement(by);
51.                             return true;
52.                     } catch (NoSuchElementException e) {
53.                             return false;
54.                     }
55.             }
56.
57.             private boolean isAlertPresent() {
58.                     try {
59.                             driver.switchTo().alert();
60.                             return true;
61.                     } catch (NoAlertPresentException e) {
62.                             return false;
63.                     }
64.             }
65.
66.             private String closeAlertAndGetItsText() {
67.                     try {
68.                             Alert
   alert = driver.switchTo().alert();
69.                             String alertText = alert.getText();
70.                             if (acceptNextAlert) {
71.                                     alert.accept();
72.                             } else {
73.                                     alert.dismiss();
74.                             }
75.                             return alertText;
76.                     } finally {
77.                             acceptNextAlert = true;
78.                     }
79.             }
80.     }
81.
82.     package tfg;
83.
84.     import java.util.concurrent.TimeUnit;
85.     import org.junit.*;
86.     import static org.junit.Assert.*;
87.     import org.openqa.selenium.*;
88.     import org.openqa.selenium.chrome.ChromeDriver;
89.     import org.openqa.selenium.firefox.FirefoxDriver;
90.
91.     public class AsistenciaPreguntas {
92.       private WebDriver driver;
93.       private String baseUrl;
94.       private boolean acceptNextAlert = true;
95.       private StringBuffer verificationErrors = new StringBuffer();
96.
97.       @Before
98.       public void setUp() throws Exception {
99.               System.setProperty("webdriver.chrome.driver",
100.                                 "D:\\TFG\\chromedriver.exe");
101.             driver = new ChromeDriver();
102.             baseUrl = "https://www.katalon.com/";
103.             driver.manage().timeouts().implicitlyWait(30,
   TimeUnit.SECONDS);
104.     }
105.
```

```
106.      @Test
107.      public void testAsistenciaPreguntas() throws Exception {
108.        driver.get("https://jorx35.wixsite.com/misitio");
109.        driver.findElement(By.id("comp-igwc2nn32label")).click();
110.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='¿Cómo puedo agregar una nueva
   pregunta?'])[1]/following::div[8]")).click();
111.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='¿Puedo agregar un video en mi
   FAQ?'])[1]/following::symbol[1]")).click();
112.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='¡Eso es! Una miniatura de tu video
   aparecerá en la casilla del texto de la
   respuesta'])[1]/following::div[3]")).click();
113.      }
114.
115.      @After
116.      public void tearDown() throws Exception {
117.        driver.quit();
118.        String verificationErrorString = verificationErrors.toStrin
   g();
119.        if (!"".equals(verificationErrorString)) {
120.          fail(verificationErrorString);
121.        }
122.      }
123.
124.      private boolean isElementPresent(By by) {
125.        try {
126.          driver.findElement(by);
127.          return true;
128.        } catch (NoSuchElementException e) {
129.          return false;
130.        }
131.      }
132.
133.      private boolean isAlertPresent() {
134.        try {
135.          driver.switchTo().alert();
136.          return true;
137.        } catch (NoAlertPresentException e) {
138.          return false;
139.        }
140.      }
141.
142.      private String closeAlertAndGetItsText() {
143.        try {
144.          Alert alert = driver.switchTo().alert();
145.          String alertText = alert.getText();
146.          if (acceptNextAlert) {
147.            alert.accept();
148.          } else {
149.            alert.dismiss();
150.          }
151.          return alertText;
152.        } finally {
153.          acceptNextAlert = true;
154.        }
155.      }
156.    }
157.    package tfg;
```

```
158.
159.    import java.util.regex.Pattern;
160.    import java.util.concurrent.TimeUnit;
161.    import org.junit.*;
162.    import static org.junit.Assert.*;
163.    import static org.hamcrest.CoreMatchers.*;
164.    import org.openqa.selenium.*;
165.    import org.openqa.selenium.chrome.ChromeDriver;
166.    import org.openqa.selenium.firefox.FirefoxDriver;
167.    import org.openqa.selenium.support.ui.Select;
168.
169.    public class AsistenciaContacto {
170.      private WebDriver driver;
171.      private String baseUrl;
172.      private boolean acceptNextAlert = true;
173.      private StringBuffer verificationErrors = new StringBuffer();
174.
175.      @Before
176.      public void setUp() throws Exception {
177.              System.setProperty("webdriver.chrome.driver",
178.                                 "D:\\TFG\\chromedriver.exe");
179.              driver = new ChromeDriver();
180.              baseUrl = "https://www.katalon.com/";
181.              driver.manage().timeouts().implicitlyWait(30,
   TimeUnit.SECONDS);
182.      }
183.      @Test
184.      public void testAsistenciaContacto() throws Exception {
185.        driver.get("https://jorx35.wixsite.com/misitio");
186.        driver.findElement(By.id("comp-igwc2nn32label")).click();
187.        driver.findElement(By.id("field1")).click();
188.        driver.findElement(By.id("field1")).click();
189.        driver.findElement(By.id("field1")).clear();
190.        driver.findElement(By.id("field1")).sendKeys("Jorge
   Aparicio Rabadan");
191.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='PONTE EN
   CONTACTO'])[1]/following::div[4]")).click();
192.        driver.findElement(By.id("field2")).click();
193.        driver.findElement(By.id("field2")).clear();
194.        driver.findElement(By.id("field2")).sendKeys("jorch35@gmail
   .com");
195.        driver.findElement(By.id("comp-
   j6vtbp2hfieldMessage")).click();
196.        driver.findElement(By.id("comp-
   j6vtbp2hfieldMessage")).clear();
197.        driver.findElement(By.id("comp-
   j6vtbp2hfieldMessage")).sendKeys("hola, necesito ayuda");
198.        driver.findElement(By.id("comp-j6vtbp2hsubmit")).click();
199.      }
200.      @After
201.      public void tearDown() throws Exception {
202.        driver.quit();
203.        String verificationErrorString = verificationErrors.toStrin
   g();
204.        if (!"".equals(verificationErrorString)) {
205.          fail(verificationErrorString);
206.        }
207.      }
208.
```

```
209.      private boolean isElementPresent(By by) {
210.        try {
211.          driver.findElement(by);
212.          return true;
213.        } catch (NoSuchElementException e) {
214.          return false;
215.        }
216.      }
217.
218.      private boolean isAlertPresent() {
219.        try {
220.          driver.switchTo().alert();
221.          return true;
222.        } catch (NoAlertPresentException e) {
223.          return false;
224.        }
225.      }
226.
227.      private String closeAlertAndGetItsText() {
228.        try {
229.          Alert alert = driver.switchTo().alert();
230.          String alertText = alert.getText();
231.          if (acceptNextAlert) {
232.            alert.accept();
233.          } else {
234.            alert.dismiss();
235.          }
236.          return alertText;
237.        } finally {
238.          acceptNextAlert = true;
239.        }
240.      }
241.    }
242.    package tfg;
243.
244.    import java.util.regex.Pattern;
245.    import java.util.concurrent.TimeUnit;
246.    import org.junit.*;
247.    import static org.junit.Assert.*;
248.    import static org.hamcrest.CoreMatchers.*;
249.    import org.openqa.selenium.*;
250.    import org.openqa.selenium.chrome.ChromeDriver;
251.    import org.openqa.selenium.firefox.FirefoxDriver;
252.    import org.openqa.selenium.support.ui.Select;
253.
254.    public class AltavozEZ00001 {
255.      private WebDriver driver;
256.      private String baseUrl;
257.      private boolean acceptNextAlert = true;
258.      private StringBuffer verificationErrors = new StringBuffer();
259.
260.      @Before
261.      public void setUp() throws Exception {
262.              System.setProperty("webdriver.chrome.driver",
263.                                 "D:\\TFG\\chromedriver.exe");
264.              driver = new ChromeDriver();
265.              baseUrl = "https://www.katalon.com/";
266.              driver.manage().timeouts().implicitlyWait(30,
   TimeUnit.SECONDS);
267.      }
```

```
268.
269.      @Test
270.      public void testComprarAltavozEZ00001() throws Exception {
271.        driver.get("https://jorx35.wixsite.com/misitio");
272.        driver.findElement(By.id("comp-ih3hrpn1link")).click();
273.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='EZ
   00001'])[1]/preceding::button[1]")).click();
274.
275.        driver.findElement(By.linkText("Ver todos los
   detalles")).click();
276.
277.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Cantidad'])[1]/following::button[1]")).click();
278.
279.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Precio'])[1]/following::a[1]")).click();
280.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Total'])[3]/following::span[1]")).click();
281.
282.        driver.findElement(By.cssSelector("svg")).click();
283.      }
284.
285.      @After
286.      public void tearDown() throws Exception {
287.        driver.quit();
288.        String verificationErrorString = verificationErrors.toStrin
   g();
289.        if (!"".equals(verificationErrorString)) {
290.          fail(verificationErrorString);
291.        }
292.      }
293.
294.      private boolean isElementPresent(By by) {
295.        try {
296.          driver.findElement(by);
297.          return true;
298.        } catch (NoSuchElementException e) {
299.          return false;
300.        }
301.      }
302.
303.      private boolean isAlertPresent() {
304.        try {
305.          driver.switchTo().alert();
306.          return true;
307.        } catch (NoAlertPresentException e) {
308.          return false;
309.        }
310.      }
311.
312.      private String closeAlertAndGetItsText() {
313.        try {
314.          Alert alert = driver.switchTo().alert();
315.          String alertText = alert.getText();
316.          if (acceptNextAlert) {
317.            alert.accept();
318.          } else {
319.            alert.dismiss();
```

```
320.          }
321.          return alertText;
322.        } finally {
323.          acceptNextAlert = true;
324.        }
325.      }
326.    }
327.
328.    package tfg;
329.
330.    import java.util.concurrent.TimeUnit;
331.    import org.junit.*;
332.    import static org.junit.Assert.*;
333.    import org.openqa.selenium.*;
334.    import org.openqa.selenium.chrome.ChromeDriver;
335.
336.    import com.sun.media.jfxmedia.logging.Logger;
337.
338.    public class Compra1 {
339.      private WebDriver driver;
340.      private String baseUrl;
341.      private boolean acceptNextAlert = true;
342.      private StringBuffer verificationErrors = new StringBuffer();
343.
344.      @Before
345.      public void setUp() throws Exception {
346.              System.setProperty("webdriver.chrome.driver",
347.                      "D:\\TFG\\chromedriver.exe");
348.        driver = new ChromeDriver();
349.        baseUrl = "https://www.katalon.com/";
350.        driver.manage().timeouts().implicitlyWait(30,
   TimeUnit.SECONDS);
351.
352.      }
353.
354.      @Test
355.      public void Compra1() throws Exception {
356.        driver.get("https://jorx35.wixsite.com/misitio");
357.        driver.findElement(By.id("comp-ih3hrpn1label")).click();
358.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='EZ
   00001'])[1]/preceding::img[1]")).click();
359.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Cantidad'])[1]/following::button[1]")).click();
360.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Precio'])[1]/following::a[1]")).click();
361.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Seguir
   comprando'])[1]/following::button[1]")).click();
362.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Conectar Métodos de
   Pago'])[1]/following::span[1]")).click();
363.      }
364.
365.      @After
366.      public void tearDown() throws Exception {
367.        driver.quit();
368.        String verificationErrorString = verificationErrors.toStrin
   g();
```

```
369.       if (!"".equals(verificationErrorString)) {
370.         fail(verificationErrorString);
371.       }
372.     }
373.
374.     private boolean isElementPresent(By by) {
375.       try {
376.         driver.findElement(by);
377.         return true;
378.       } catch (NoSuchElementException e) {
379.         return false;
380.       }
381.     }
382.
383.     private boolean isAlertPresent() {
384.       try {
385.         driver.switchTo().alert();
386.         return true;
387.       } catch (NoAlertPresentException e) {
388.         return false;
389.       }
390.     }
391.
392.     private String closeAlertAndGetItsText() {
393.       try {
394.         Alert alert = driver.switchTo().alert();
395.         String alertText = alert.getText();
396.         if (acceptNextAlert) {
397.           alert.accept();
398.         } else {
399.           alert.dismiss();
400.         }
401.         return alertText;
402.       } finally {
403.         acceptNextAlert = true;
404.       }
405.     }
406.   }
407.   package tfg;
408.
409.   import java.util.regex.Pattern;
410.   import java.util.concurrent.TimeUnit;
411.   import org.junit.*;
412.   import static org.junit.Assert.*;
413.   import static org.hamcrest.CoreMatchers.*;
414.   import org.openqa.selenium.*;
415.   import org.openqa.selenium.chrome.ChromeDriver;
416.   import org.openqa.selenium.firefox.FirefoxDriver;
417.   import org.openqa.selenium.support.ui.Select;
418.
419.   public class CompraVariosAltavoces {
420.     private WebDriver driver;
421.     private String baseUrl;
422.     private boolean acceptNextAlert = true;
423.     private StringBuffer verificationErrors = new StringBuffer();
424.
425.     @Before
426.     public void setUp() throws Exception {
427.             System.setProperty("webdriver.chrome.driver",
428.                                "D:\\TFG\\chromedriver.exe");
```

```
429.              driver = new ChromeDriver();
430.        baseUrl = "https://www.katalon.com/";
431.        driver.manage().timeouts().implicitlyWait(30,
   TimeUnit.SECONDS);
432.      }
433.
434.      @Test
435.      public void testCompraVariosAltavoces() throws Exception {
436.        driver.get("https://jorx35.wixsite.com/misitio");
437.        driver.findElement(By.id("comp-ih21w39elabel")).click();
438.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Precio'])[2]/following::button[1]")).click();
439.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Cantidad'])[1]/following::button[1]")).click();
440.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Carrito'])[1]/preceding::span[1]")).click();
441.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Precio'])[3]/following::button[1]")).click();
442.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Cantidad'])[1]/following::button[1]")).click();
443.        driver.findElement(By.id("minicart-title")).click();
444.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Carrito'])[1]/preceding::span[1]")).click();
445.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Precio'])[4]/following::button[1]")).click();
446.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Cantidad'])[1]/following::button[1]")).click();
447.        driver.findElement(By.id("minicart-title")).click();
448.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Carrito'])[1]/preceding::span[1]")).click();
449.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Precio'])[5]/following::button[1]")).click();
450.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Cantidad'])[1]/following::button[1]")).click();
451.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-
   space(.)='Carrito'])[1]/preceding::span[1]")).click();
452.        driver.findElement(By.cssSelector("svg > text")).click();
453.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Precio'])[4]/following::a[1]")).click();
454.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Seguir
   comprando'])[1]/following::span[1]")).click();
455.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Abre tu tienda y empieza a
   vender'])[1]/preceding::span[1]")).click();
456.      }
457.
458.      @After
```

```
459.      public void tearDown() throws Exception {
460.        driver.quit();
461.        String verificationErrorString = verificationErrors.toStrin
   g();
462.        if (!"".equals(verificationErrorString)) {
463.          fail(verificationErrorString);
464.        }
465.      }
466.
467.      private boolean isElementPresent(By by) {
468.        try {
469.          driver.findElement(by);
470.          return true;
471.        } catch (NoSuchElementException e) {
472.          return false;
473.        }
474.      }
475.
476.      private boolean isAlertPresent() {
477.        try {
478.          driver.switchTo().alert();
479.          return true;
480.        } catch (NoAlertPresentException e) {
481.          return false;
482.        }
483.      }
484.
485.      private String closeAlertAndGetItsText() {
486.        try {
487.          Alert alert = driver.switchTo().alert();
488.          String alertText = alert.getText();
489.          if (acceptNextAlert) {
490.            alert.accept();
491.          } else {
492.            alert.dismiss();
493.          }
494.          return alertText;
495.        } finally {
496.          acceptNextAlert = true;
497.        }
498.      }
499.    }
500.
501.    package tfg;
502.
503.    import java.util.concurrent.TimeUnit;
504.    import org.junit.*;
505.    import static org.junit.Assert.*;
506.    import org.openqa.selenium.*;
507.    import org.openqa.selenium.chrome.ChromeDriver;
508.    import org.openqa.selenium.support.ui.WebDriverWait;
509.
510.    public class Newsletter {
511.      private WebDriver driver;
512.      private String baseUrl;
513.      private boolean acceptNextAlert = true;
514.      private StringBuffer verificationErrors = new StringBuffer();
515.
516.      @Before
517.      public void setUp() throws Exception {
```

```
518.              System.setProperty("webdriver.chrome.driver",
519.                      "D:\\TFG\\chromedriver.exe");
520.        driver = new ChromeDriver();
521.        baseUrl = "https://www.katalon.com/";
522.        driver.manage().timeouts().implicitlyWait(30,
   TimeUnit.SECONDS);
523.      }
524.
525.      @Test
526.      public void testNewsletter() throws Exception {
527.        driver.get("https://jorx35.wixsite.com/misitio");
528.        driver.manage().window().maximize();
529.        driver.findElement(By.id("comp-igwc2nn31label")).click();
530.        driver.findElement(By.xpath("/html/body/div/get-
   subscribers/div/div/div[2]/div/input")).click();
531.        driver.findElement(By.name("subscriberEmail")).clear();
532.        driver.findElement(By.name("subscriberEmail")).sendKeys("ho
   la@gmail.com");
533.        driver.findElement(By.xpath("(.//*[normalize-space(text())
   and normalize-space(.)='Dirección de
   email'])[1]/following::span[1]")).click();
534.        driver.findElement(By.xpath("//h3")).click();
535.        try {
536.            assertEquals("Thanks for Subscribing!",
   driver.findElement(By.xpath("/  /h3")).getText());
537.        } catch (Error e) {
538.            verificationErrors.append(e.toString());
539.        }
540.        driver.findElement(By.xpath("//div/div/div/div")).click();
541.        try {
542.            assertEquals("We'll send news to your inbox.",
   driver.findElement(By.xpath("(.//*[normalize-space(text()) and
   normalize-space(.)='Thanks for
   Subscribing!'])[1]/following::p[1]")).getText());
543.        } catch (Error e) {
544.            verificationErrors.append(e.toString());
545.        }
546.
547.      }
548.
549.      @After
550.      public void tearDown() throws Exception {
551.        driver.quit();
552.        String verificationErrorString = verificationErrors.toStrin
   g();
553.        if (!"".equals(verificationErrorString)) {
554.          fail(verificationErrorString);
555.        }
556.      }
557.
558.      private boolean isElementPresent(By by) {
559.        try {
560.          driver.findElement(by);
561.          return true;
562.        } catch (NoSuchElementException e) {
563.          return false;
564.        }
565.      }
566.
567.      private boolean isAlertPresent() {
```

```
568.         try {
569.           driver.switchTo().alert();
570.           return true;
571.         } catch (NoAlertPresentException e) {
572.           return false;
573.         }
574.       }
575.
576.       private String closeAlertAndGetItsText() {
577.         try {
578.           Alert alert = driver.switchTo().alert();
579.           String alertText = alert.getText();
580.           if (acceptNextAlert) {
581.             alert.accept();
582.           } else {
583.             alert.dismiss();
584.           }
585.           return alertText;
586.         } finally {
587.           acceptNextAlert = true;
588.         }
589.       }
590.     }
591.
592.     package tfg;
593.
594.     import org.openqa.selenium.By;
595.     import org.openqa.selenium.NoSuchFrameException;
596.     import org.openqa.selenium.WebDriver;
597.     import org.openqa.selenium.WebElement;
598.     import org.openqa.selenium.chrome.ChromeDriver;
599.
600.     public class Main {
601.
602.             public static void main(String[] args) throws Interrupt
    edException {
603.                     // TODO Auto-generated method stub
604.
605.                     System.setProperty("webdriver.chrome.driver",
606.                                     "D:\\TFG\\chromedriver.exe");
607.                     WebDriver driver = new ChromeDriver();
608.                     driver.get("https://jorx35.wixsite.com/misitio"
    );
609.                 driver.findElement(By.id("comp-
    ih21w39elabel")).click();
610.                     Thread.sleep(5000);  // Let the user actually
    see something!
611.                     WebElement
    searchBox = driver.findElement(By.name("q"));
612.                     searchBox.sendKeys("ChromeDriver");
613.                     searchBox.submit();
614.
615.                     Thread.sleep(5000);
616.                    // driver.quit();
617.             }
618.
619.     }
```

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Tue Jul 02 19:13:43 CEST 2019
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)