SELENIUM

Guillermo Morillo Morais
Proyecto de final de curso
Desarrollo de Aplicaciones Web
IES María de Zayas y Sotomayor
2018

Índice

TECNOLOGÍAS UTILIZADAS

- IntellIJ Idea
- Selenium WebDriver
- Selenium IDE
- JUnit
- Java
- ChromeDriver
- Google Chrome
- Github
- ChroPath

El proyecto se va a desarrollar en el IDE IntellIJ Idea, utilizando el lenguaje Java, el framework de implementación de pruebas sobre navegador Selenium WebDriver y el de pruebas unitarias para java JUnit.

Selenium IDE y ChroPath son extensiones para navegador, en este caso se utiliza el navegador Google Chrome.
La parte de Selenium WebDriver necesita un driver de conexión con el navegador, el cual utilizaremos ChromeDriver.

El código de prueba, así como toda la documentación y archivos del proyecto estarán en Github.

http://github.com/DAWZayas-Projects/MORILLO-MORAIS-GUILLERMO

Las pruebas se realizarán sobre el proyecto WhereToEat
realizado en el IES María de Zayas y Sotomayor, durante el curso 2017-2018 de DAW en las asignaturas de Desarrollo Cliente y Diseño Web.

URL: wheretoeat-ca57a.firebaseapp.com

DESCRIPCIÓN DEL PROYECTO

La intención del proyecto es transmitir la filosofía de selenium, explicar en qué consiste la tecnología de automatización del navegador para pruebas, como podría ser un proyecto de testeo sobre una aplicación, qué metodología se puede utilizar y de qué componentes y cómo se conforman las pruebas y test que se pueden realizar.

Se va a realizar un caso de prueba sobre la aplicación WhereToEat, realizando una estructura de proyecto basado en el patrón page-object, separando, para además explicar mejor cada parte de un caso de prueba, las páginas tomadas como objeto, heredando de una página genérica, el WebDriver como conexión entre la aplicación y las pruebas, y los test con las comprobaciones de los resultados.

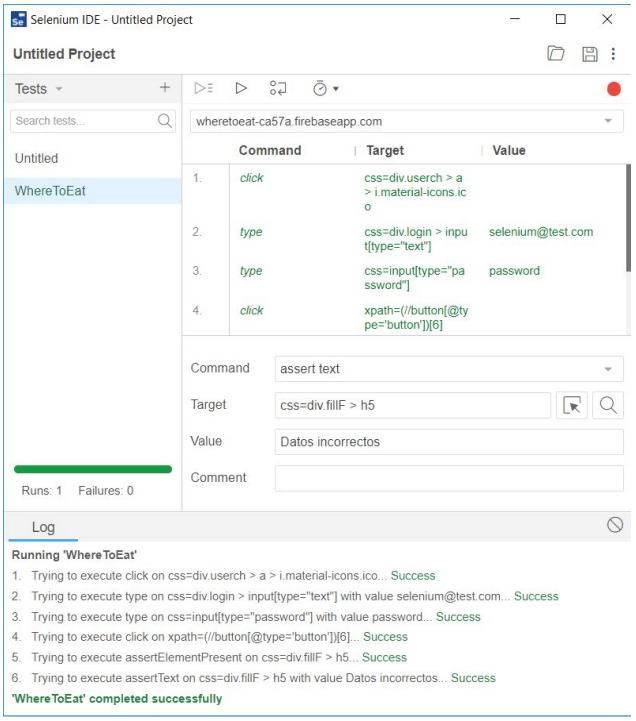
Además se pretende que se conozca también la existencia y utilización básica de la extensión de navegador Selenium IDE, que comparte la filosofía fundamental de Selenium WebDriver.

¿QUÉ ES SELENIUM?

Selenium es un entorno de pruebas, que se realizan mediante la automatización del navegador, a través de WebDriver o de la extensión para navegadores, se toma el control de navegador y se identifican los elementos que intervienen en la aplicación y se les programan ciertas acciones.

De esas acciones se pueden recoger resultados, para apoyándonos en otros sistemas de pruebas unitarias, como puede ser en este caso JUnit, comparamos resultados para obtener resultados positivos o negativos a esos test y determinar si está funcionando correctamente o no.

SELENIUM IDE



Se trata de una extensión para navegadores, la cual despliega una aplicación mediante la cual se pueden cargar proyectos que se tengan ya prediseñados, o diseñar nuevos proyectos de test.

Una de las partes más interesantes de la aplicación es que tiene la opción de grabar las interacciones que vamos teniendo con la página web, guardando cada elemento y cada acción que mantenemos con dichos elementos, y configurando un test que después se puede volver a ejecutar las veces que sean necesarias.

Otra de las maneras de diseñar los test es mediante la identificación de un elemento web, para lo cual también nos proporcionan un puntero por el cual simplemente pinchando el elemento, lo tendríamos identificado. Y a dicho elemento le asignamos una acción a realizar y un valor en el caso de que sea necesario, como una entrada de datos, o una comprobación de que contiene ese texto.

También dispone de un log, o consola. Es la parte donde se transmite qué acción está realizando en cada momento y que resultado devuelve.

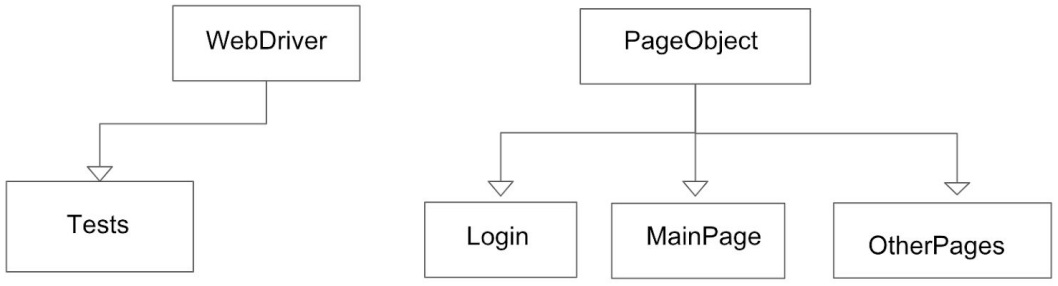
SELENIUM WEBDRIVER

Selenium WebDriver tiene la misma filosofía de conectarse al navegador y controlarlo. Lo efectúa a través del WebDriver, el cual transmite al navegador las instrucciones que programamos mediante código y retorna los resultados.

Las ventajas con respecto al IDE es que mediante el WebDriver no tenemos la necesidad de tener unos comandos ejecutados de manera secuencial, si no que podemos programar arquitecturas de proyectos de test mucho más complejas y reutilizables y podemos hacer uso de él mediante otras herramientas de automatización o de integración continua, sin tener que estar ejecutando la extensión de navegador nosotros mismos.

Selenium actualmente mantiene integración con varios lenguajes de programación, como principales Java, Ruby, Python y C++.
Y tiene compatibilidad con sistemas de pruebas unitarias, como en el caso de Java, JUnit.

ESTRUCTURA DEL PROYECTO



En el caso de este proyecto, se desarrolla manteniendo una estructura basada en el patrón page-object.
Tenemos un WebDriver, que es el que invoca al navegador y define las propiedades de la conexión. De él heredan los test, que es donde se realizarán las comprobaciones de los resultados en sí mismos.

Por otra parte tenemos PageObject que es una clase donde se definen las acciones básicas y comunes que se utilizarán después en las páginas que heredan de ella, que son las páginas propiamente dichas, en las cuales se agrupan elementos y acciones que tienen en común que se realizan en una misma sección de la aplicación.

WEBDRIVER

Para la inicialización del WebDriver, creamos la propiedad del sistema, a la que le indicamos la ruta donde tenemos almacenado el webdriver.exe en nuestro equipo.

También le podemos indicar una serie de propiedades que tiene el chromedriver, tanto preferencias como opciones, en este caso de ejemplo, las preferencias hacen referencia a cuando descargamos archivos, indicando el directorio destino, etc…

Las opciones son las características principales con las que se va a ejecutar el webDriver, como por ejemplo, la opción “headless” la cual indica que se ejecutará un navegador sin cabecera, es decir sin interfaz gráfica, lo cual para algunos casos conviene, ya que se puede ejecutar en un servidor sin interfaz, o en caso de necesitar poca utilizaciones de recursos.

Las opciones se utilizan a la hora de instanciar el objeto driver de la clase ChromeDriver.

Una vez instanciado, se pueden setear varias opciones, como por ejemplo, el tamaño del navegador que se va a automatizar, la maximización de la pantalla, se definen los tamaños de las esperas implícitas, y la página de inicio.

PATRÓN PAGE-OBJECT

La función principal del patrón page-object es la de separar la ejecución de los test, de la identificación y funciones de los elementos que interaccionan en la página.

Haciendo así que la reutilización sea mayor y la agrupación de las acciones tenga mayor presencia, agrupando en sub-páginas que hereden de una página estándar, que tienen una temática común, o pertenecen a una misma sección de la aplicación.

Cada sección o cada página será tratado en los test como si fuera un objeto, y las acciones que puede realizar como los métodos de un objeto.
Facilitando que si en algún momento es necesario cambiar algo de alguna página, no se cambia el test, si no la declaración de la página.

PAGEOBJECT

Se trata de la definición de la página padre, de la cual heredarán las demás páginas, en ella declararemos el constructor al que luego haremos referencia desde las demás páginas, recibiendo un webdriver como objeto, ya que desde esta página es donde declararemos las funciones comunes a todas las páginas, algunas de ellas, que utilizan acciones propias del webdriver tal como las esperas.
También se declara un ejecutor de javaScript y las Actions que es la manera de declarar acciones de tipo DOM, como moveMouse(), etc…

ESPERAS

Las esperas en Selenium se definen como el tiempo que espera el WebDriver para recibir una respuesta del WebElement determinado antes de darlo por fallido.

En Selenium existen tres tipos de esperas:

- Esperas implícitas: Son aquellas que se realizan siempre, se define al principio un tiempo de espera para el webDriver y se ejecutan en todo el proyecto.
- Esperas explícitas: Se declaran para un elemento concreto.
- thread sleep: Se duerme la ejecución de los procesos un tiempo determinado.

WEBELEMENTS

Los WebElements son los elementos HTML que forman la página, a través de los cuales se ejecutan las acciones en Selenium.

Hay varias maneras de identificarlos, las principales son:

- Por el WebDriver: WebElement element = driver.getElement(By.id("id_elemento"));
- A través de anotaciones:
  @FindBy(id = "id_elemento") private WebElement element;

  @FindAll({@FindBy(css = "css_elemento") }) private List<WebElement> element;

Se puede hacer una búsqueda de elementos mediante className, css, how, id, linkText, name, partialLinkText, tagName, using, xpath…

Como hemos dicho las acciones se realizan a través de los webElements y las principales acciones que pueden realizar son click(), clear(), getAttribute(), getClass(), getCssValue(), getSize(), getText(), isDisplayed(), isEnabled(), isSelected(), sendKeys()...

Habrá ocasiones que se necesite esperar a que un webElement esté visible, o clickable, o realice alguna acción de tipo DOM, como moverse hasta un elemento.

En este caso, esas acciones se definen como comunes en PageObject y se realizan pasandole el webElement objetivo como parámetro.

DEFINICIÓN DE PÁGINAS

Una vez tenemos la definición de la página padre, la creación de las demás es simplemente heredar.

Se utiliza la estrategia de agrupar las partes que aparecen en una misma página en la aplicación, en este caso, el header como menú de navegación en el caso del navegador a tamaño máximo, la página de inicio, la de perfil, la de registro …

Ya en las páginas concretas, se realiza la declaración de los elementos que van a intervenir, y los métodos o funciones en los que intervienen.

En caso de este proyecto vamos a utilizar el método de las anotaciones, y la manera de definir los webElements, por lo general utilizaremos xpath, ya que en este proyecto no se definen los elementos con ID y xpath aún no siendo la manera más rápida, es la más cómoda y segura, ya que con la ayuda de la extensión ChroPath se consigue fácilmente el xpath.

Para la definición de los métodos la estrategia que se va a seguir es la de realizar operaciones pequeñas, en las cuales sean necesarias pocas acciones, y en la medida de lo posible buscar la forma de que devuelvan algún resultado, ya sea un booleano, o un String para realizar la comprobación en los test.

DEFINICIÓN DE LOS TEST

Para la implementación de los test, la manera óptima sería realizar una clase de test por cada agrupación o por temática, en el caso de este proyecto, al ser un caso de uso pequeño, solamente se realizará en una única clase.

La clase test, extiende de webDriver teniendo así todo lo definido en ella.

Mediante la anotación @FixMethodOrder le vamos a definir en el orden que queremos que se ejecuten los métodos señalados con la anotación @Test, ya que JUnit que es la tecnología en la que nos apoyamos para realizar los test ya tiene orden intrínseco entre @BeforeTest, que utilizamos para instanciar el webDriver, @Test, para la ejecución de los test, y @AfterTest, que utilizamos para el cierre del webDriver.

Debemos instanciar un objeto por cada clase que hemos definido, al que le adjuntamos el objeto driver:

private static LoginPage loginPage = new LoginPage(driver);

Definimos los test con un nombre identificativo, y utilizamos las aserciones de JUnit, y para poder mantener el orden que le hemos asignado a la realización de los test los nombres empezarán de la A a la Z.

Generalmente no haría falta definir un orden, ya que se realizarán de manera secuencial según se han definido en el código, pero es posible que en tiempo de ejecución no se mantenga ese orden, por ello es mejor definir un orden y evitar posibles fallos que puedan producirse, ya que muchas veces, los resultados de un test, pueden influir en la realización del siguiente.

MANERA ÓPTIMA DE UTILIZAR SELENIUM

La manera más óptima de utilizar Selenium, es definir un proyecto con los test que queremos realizar sobre nuestra aplicación, y apoyarnos en un sistema de integración continua como podría ser Jenkins para el caso de Java y en un constructor de aplicaciones como puede ser Maven.

El flujo sería programar una tarea de Jenkins, para que cada vez que se realice una actualización de nuestra aplicación en nuestro control de versiones, Jenkins lo detecte y apoyándose en las fases de Maven, ejecutase la fase de Test, y una vez se comprueba que los test son correctos, y que la actualización no ha roto nada de lo que ya teníamos y es correcta, dar el visto bueno para la subida a nuestro control de versiones o directamente a la rama que elijamos.

ALTERNATIVAS A SELENIUM

Hay diferentes alternativas para la realización de testing como puede ser Katalon Studio, Testcomplete, Oracle Testing Suite…

O algunas basadas en Selenium pero que incluyen diferentes plataformas e incluso plataformas móviles como puede ser Appium.