Selenium Avanzado

Desafio 1:

Proyecto Java: TestingDesafioSelenium1

Para grabar la prueba utilice el plugin Selenium Launch Builder en el browser de Mozilla Firefox, el cual guarde como json con el nombre “desafio1.json”. Posteriormente reproduje el json para validar que fuera correcto, para reproducirlo tuve que levantar un servidor Selenium en eclipse con la url: http://localhost:4444/ mediante la clase “Servidor.java”. Una vez validado el json exporte el caso de prueba como Junit.

Agregue el Junit exportado al proyecto y los ejecute como Junit Test.

Desafio 2:

Proyecto Java: TestingDesafioSelenium2

Los nombres de los productos los obtengo del archivo: “nombresProductos.txt” y los seteo como parámetros.

Para obtener el elemento html input del buscador de la página, lo busco mediante su nombre: “search” y lo guardo en una variable del tipo WebElement. Luego ingreso el nombre del producto en el buscado mediante la función sendKeys(“nombreProducto”). Una vez ingresado el nombre del producto a buscar, obtengo el botón para seleccionar la búsqueda mediante su xpath, para posteriormente realizar un evento de click asi se ejecuta la búsqueda del producto.

Una vez realizada la búsqueda, obtengo el primer elemento que devolvió la búsqueda mediante el xpath. Este xpath busca un elemento html img que se encuentra dentro de un elemento html a que se encuentra dentro un elemento html div cuya clase es ‘image’. Luego de obtenido el primer elemento se ejecuta el evento click sobre este primer elemento.

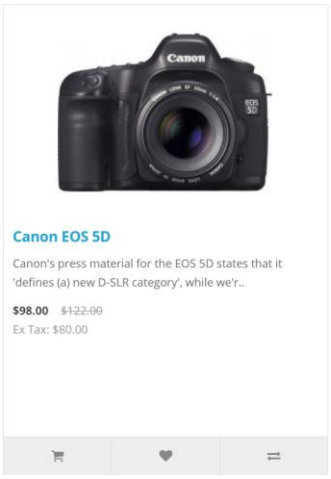
Desafio 3:

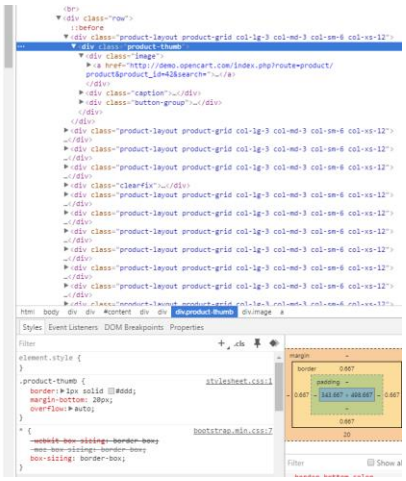
Proyecto Java: TestingDesafioSelenium3

Para obtener todos los productos ingreso el string espacio (“ ”) en el input del buscador y ejecuto la búsqueda. Para asegurarme que se desplieguen la mayor cantidad de productos que haya disponible, obtengo el select que determina cuantos productos se están desplegando. El select lo obtengo mediante su id: “input-limit”. Al obtener el select obtengo sus elementos html option y selecciono el ultimo option en el select ya que generalmente el ultimó option es el de mayor valor. El último option lo selecciono indicándole al select que seleccione la opción en el índice igual a cantidad de option menos uno.

Una vez asegurado que desplegué la mayor cantidad de productos posibles, los obtengo en una lista de WebElement mediante la búsqueda de elementos html que cumplan la condición nombre de la clase: “product-thumb”.

Ejemplo de elemento html con clase “product-thumb”:





Al obtener la lista de WebElement con todos los productos, recorro dicha lista para escribir el nombre de cada producto en el archivo “listaProducto.txt”. El nombre de cada producto lo obtengo al aplicar el método getText() al elemento html a, que se encuentra dentro de un elemento html h4, que se encuentra dentro de un elemento html cuya clase es “caption” para cada elemento en la lista de WebElement.

Desafio 4:

Proyecto Java: TestingDesafioSelenium4

Los nombres de los productos los obtengo del archivo: “listaProducto.txt” y los seteo como parámetros. Dicho txt es una copia del archivo creado en el desafío 3.

Luego en el método testPagina() seteo en el input el nombre del producto y lo busco al realizar un evento de click sobre el botón a la derecha del mencionado input.

Selecciono el primer elemento desplegado como resultado de la búsqueda y valido mediante un assertEquals que tenga el mismo nombre que el producto buscado, de ser así, asumimos que es el producto que estábamos buscando.

Luego de esta validación, seleccionamos el producto. Posteriormente busco el título del producto seleccionado y valido que sea el mismo al nombre del producto que previamente había buscado. El título del producto seleccionado lo obtengo a través de su correspondiente xpath.

Las esperas introducidas son para que al momento de ejecutar las pruebas, no se pase tan rápidamente entre un parámetro y otro así puedo validar visualmente en la interfaz gráfica que el producto buscado es el mismo que el seleccionado.

Page Object

Proyecto Java: TestingDesafioPageObject

Creo la clase PageObject.java que tiene como atributo un WebDriver y al instanciar esta clase se le debe de pasar por parámetro el WebDriver a utilizar.

Los métodos de PageObject son:

- `IngresarNombreProducto(String nombreProducto)`

Recibe como parámetro el nombre de un producto el cual inserta en el input del buscador de la página. El input del buscador lo obtengo mediante su nombre.

- BuscarProducto()

  Realiza el evento de click sobre el botón de buscar. Referencio al botón de buscar mediante su xpath.

- ObtenerNombrePrimerProducto()

  Retorna el nombre del primer producto desplegado luego de una búsqueda. Dicho primer producto lo obtengo al buscar un elemento html cuyo nombre de clase sea “product-thumb”, luego sobre este elemento busco el elemento html cuyo nombre de clase es “caption”, que dentro tiene un elemento html h4, quien tiene un elemento html a cuyo texto es el nombre del primer producto mencionado.

- SeleccionarPrimerProducto()

  Realiza el evento click() sobre el primer elemento desplegado luego de una búsqueda. Dicho primer elemento lo buscamos mediante el elemento html cuyo nombre de clase sea “product-thumb”, para luego realizar el evento click sobre el elemento html image que se encuentra dentro del elemento html a, que está dentro de un elemento html cuyo nombre de clase es “image” que se encuentra dentro del primer elemento mencionado.

- ObtenerTituloProductoSeleccionado()

  Devuelve el del título del producto seleccionado. Este título se obtiene al aplicar la función getText() sobre el elemento html que se obtiene al buscarlo por su correspondiente xpath.

Luego en la clase de test “MainTest.java”, en el método a testear, creo una instancia del PageObject pasandole por parámetro el atributo driver que es un WebDriver. Luego aplico las mismas acciones que realizaba en el desafio 4 solo que ahora llamo a los métodos de PageObject y le paso por parámetro el nombre del producto a buscar en los métodos que lo solicitan. Por otro lado, los asserEquals se siguen realizando en el método a testear.