Creación de testing automatizado para una web usando Selenium

Adrian Soria Bonilla

Resumen—El control de calidad es un elemento clave en el desarrollo y mantenimiento del software. La demanda del testing automatizado está en alza en los últimos años, ya que los tests de regresión automatizados facilitan y agilizan validar el correcto funcionamiento de cada nueva versión desplegada de un software específico, lo cual compagina en especial con el entorno dinámico de las metodologías ágiles, las cuales se han convertido en un estándar en el sector. El propósito de este proyecto consiste en la creación de un test de regresión automatizado para una web agregadora de vuelos utilizando Selenium WebDriver, una de las herramientas más populares.

Palabras clave— QA, Selenium, automatización, regresión, testing, WebDriver.

Abstract— Quality assurance is a key component in software development and maintenance. Demand for automated testing is on the rise as of late, since automated regression tests facilitate and hasten validating the proper functioning of each new deployed version of a specific software, a fact that specially resonates with the dynamic environment of agile methodologies, which have become a standard in the field. The purpose of this project consists on the creation of an automated regression test for a flight aggregator website using Selenium WebDriver, one of the most popular tools.

Index Terms— QA, Selenium, automation, regression, testing, WebDriver.

1 Introducción

TODO software que ha sido creado debe ser validado, tanto en lo que respecta al cumplimiento de todos los requisitos que se establecieron para el proyecto como al correcto funcionamiento de cada componente y su tolerancia a fallos o situaciones inesperadas.

El software está en constante evolución y cambio antes de alcanzar su versión final, y cada nueva versión de éste no debería introducir nuevos errores que se hayan solventado en versiones anteriores, y de ser así, se han de poder detectar mediante lo que se conoce como test de regresión, que consiste precisamente en verificar que no se hayan reintroducido errores.

Esta tarea, que puede llegar a ser ardua, lenta y acabar afectada por error humano (debido a su extensión y su naturaleza repetitiva), es la principal beneficiada de la creación de tests automatizados, suponiendo una inversión inicial de tiempo para la creación de dichos scripts, pero que se rentabiliza rápido ya que se pueden validar nuevas versiones del software con agilidad. En un equipo en el que se use una metodología DevOps, donde son omnipresentes la integración y entrega continuas, la automatización del test de regresión es aún más valiosa.

Sin embargo, pese a la utilidad del testing automatizado y su peso en las metodologías ágiles que se han vuelto un estándar en la mayoría de equipos, suele haber poca oferta de personas con conocimiento de automatización y se está considerando como una habilidad cada vez más valiosa.

E-mail de contacto: adriansoriabonilla@gmail.com
Mención realizada: Ingeniería del Software
Trabajo tutorizado por: Xavier Otazu Porter
Curso 2017/18

Es por ello que se decidió escoger este tema para el proyecto, de cara a poder comprender la utilidad y uso del testing automatizado mientras se intentaba simular los procedimientos que se seguirían en un equipo real.

Para ello se eligió una página agregadora de vuelos, *www.adioso.com*, de cara a construir un test de regresión automatizado que comprobara su funcionamiento, con especial atención a su función principal para la búsqueda de vuelos.

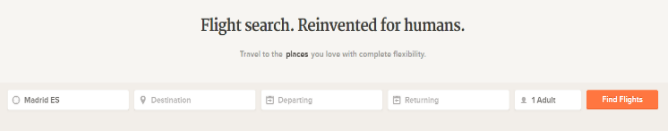

Figura 1: Página principal de adioso

Este documento está estructurado en las siguientes secciones, mencionadas por orden:

- Introducción
- Objetivos
- Estado del arte
- Limitaciones
- Metodología
- Planificación
- Desarrollo e implementación
- Resultados
- Conclusiones
- Agradecimientos
- Bibliografía
- Anexo

2 OBJETIVOS

Los objetivos de este proyecto son los siguientes:

- Aprendizaje y uso efectivo de testing automatizado, poseyendo una base de conocimiento sólida al final del proyecto, con vistas de mejorarla a lo largo del tiempo.
- Testear todas las secciones de la página www.adioso.com, con especial énfasis en la funcionalidad principal: la búsqueda de vuelos.
- Definir, documentar en detalle y ejecutar casos de test para cada una de las funcionalidades principales de la página, de cara a posteriormente automatizarlos.
- Simplificar el lanzamiento de los scripts y que se notifique por e-mail de los resultados a través de Jenkins, algo que facilitaría el mantenimiento y seguimiento de los resultados de dichos tests.

3 ESTADO DEL ARTE

En estos últimos años se ha visto una clara tendencia por parte de los equipos de test a adoptar metodologías ágiles y, algo menos común, DevOps (énfasis en CI/CD) en los últimos 2 años, y se han convertido prácticamente en los estándares en el sector mientras el modelo de desarrollo "Waterfall" empieza a perder popularidad, tal y como afirma el "International 2017 State of Testing" [1].

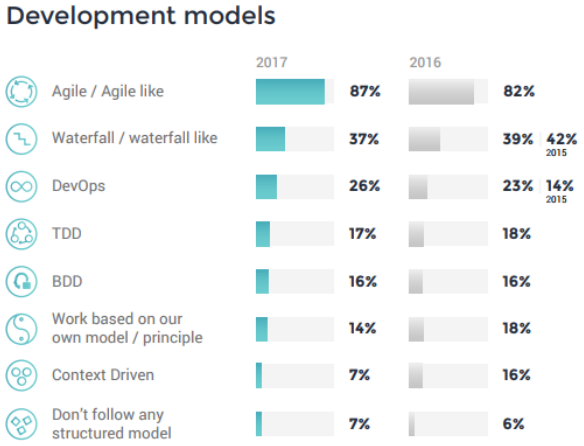

Figura 2: Extracto del International 2017 State of Testing

También se ha comprobado que la presencia de la automatización es considerable, pero parece ser que, pese a ser una habilidad altamente valorada y considerada importante, todavía no se ha asentado del todo y puede ser que hagan falta unos cuantos más años para que surjan más profesionales en automatización y que acabe de encajar en el proceso de testing.

Concretando más en el estado del arte en España, el "World Quality Report 2017-2018" [2], concretamente en su versión sobre Europa Sur, menciona que España ha demostrado interés por DevOps, metodologías ágiles y la automatización, pero parece estar teniendo problemas con encontrar profesionales y adaptarse las dos últimas.
Hay también un creciente interés en el testeo de la seguridad, siendo ésta una de las habilidades más valoradas.

Una pregunta que surge a menudo sobre el futuro del testing es si la automatización se impondrá por delante del testeo manual, y de hecho parece que en ocasiones hay una tendencia a "sobreautomatizar", pero pese a todos hay tests que requieren opiniones o criterios más subjetivos y humanos, por lo que, en vez de imponerse una técnica por encima de la otra, es muy probable que en un futuro simplemente se usen conjuntamente, cada una en la situación más apropiada.

4 LIMITACIONES

Es menester admitir que el proyecto presenta una serie de limitaciones a considerar, motivo por el cual se desearía poder revisitar este proyecto (o realizar uno nuevo de temática similar) en un futuro en el cual no estén presentes.

La mayor causa de estas limitaciones es que no se está realizando este proyecto desde el entorno de desarrollo de la compañía poseedora de la web. Esto implica que no se tiene acceso al código, lo cual imposibilita cualquier tipo de test de caja blanca, sea test unitario, path coverage, decision coverage u otros, no se conocen con certeza los requisitos funcionales y no funcionales de la web, por lo que en ocasiones se desconoce la intención o motivación de ciertas implementaciones, y al no contar con el permiso explícito de los propietarios sería una práctica bastante cuestionable realizar testing de seguridad o de carga en la página, puesto que no se quiere impactar su funcionamiento actual sin el beneplácito de los propietarios.

Es por estos motivos que los tests realizados en el proyecto son todos de carácter exploratorio y funcional, en caja negra, y en algunos tests sólo se podían realizar conjeturas sobre las intenciones tras algunas implementaciones, aunque en determinados tests, la respuesta de la web permitía intuir la estructura de ciertas partes de su código.
Sin embargo, el hecho de que ciertos aspectos de la página, principalmente técnicos, no se hayan podido testear no implica que el contenido del proyecto sea escueto o que no se haya realizado una labor concienzuda para el testeo de la web, pero lo que sí que implica es que de ninguna manera se puede afirmar que se haya realizado un testing completo.

5 METODOLOGÍA

Para la realización de este proyecto se optó por utilizar una metodología ágil, concretamente SCRUM, debido a la adaptabilidad que daba a la hora de modificar objetivos, de cara a poder recalibrar el TFG en caso de situaciones inesperadas, además de ser una metodología muy común en el sector y se desea simular un entorno profesional de testing el máximo posible. Aunque es relativamente incorrecto mencionar que se ha usado SCRUM al ser un equipo compuesto por una sola persona (motivo por el cual los roles y reuniones de seguimiento como el daily scrum pierden el sentido), sí que es cierto que se ha realizado el desarrollo mediante sprints, planteando milestones que asumir a lo largo de cada uno.

Respecto al diseño del software en sí, dado a que se basaba en interactuar estrechamente con una página web, se ha diseñado según el principio de PageObject, tal y como recomienda la documentación de Selenium[3], separando la estructura HTML de la página web y su funcionamiento de los tests en sí, siendo el caso más explícito la relación entre el PageObject que representa a la página principal donde se realiza la búsqueda y la clase que contiene todos los tests relacionados con realizar búsquedas con diferentes parámetros.

6 Planificación

6.1 Milestones del proyecto

Tal y como se mencionó en el apartado previo, se planificaron varias milestones a cumplir en cada sprint, de cara al desarrollo de este proyecto. Se definieron 3 milestones principales que alcanzar, cada una vinculada con la fecha de entrega de los informes de progreso 1, 2, e informe final.

A continuación se detallan los 3 milestones principales:

- El primer milestone consistía en configurar todo el entorno de desarrollo, programar unos tests automatizados simples del resto de páginas en *adioso.com* exceptuando la página principal (la cual contiene la búsqueda de vuelos) e integrar dicho programa con Jenkins, un servidor utilizado a menudo para integración continua. El principal propósito de este milestone es tener una base sólida sobre la cual se trabajará en los dos siguientes milestones, además de servir para empezar a familiarizarse con Selenium y TestNG.
- El segundo milestone consistía en la realización y documentación exhaustiva de tests exploratorios funcionales relacionados con la búsqueda de vuelos, registrando el valor que se introdujo en cada campo, el resultado esperado, el resultado recibido, un link que lleve al resultado de dicha búsqueda y observaciones adicionales. El objetivo de este milestone es establecer de forma precisa los tests que se automatizarán en el tercer milestone, teniendo además para ellos una documentación sólida que facilite su seguimiento.
- El tercer y último milestone consistía en automatizar los tests confeccionados en el segundo milestone, optimizar el código y asegurarse de que toda la infraestructura funciona correctamente. En este último milestone es cuando se da uso a toda la base funcional desarrollada durante el primer sprint, construyendo sobre ella para plasmar todos los tests.

Durante la planificación se preveía la posibilidad de que hubiera complicaciones en la configuración del entorno durante el primer milestone, y que no se pudieran satisfacer todos los requisitos de ese sprint a tiempo, puesto que no se podía prever con certeza qué complicaciones podrían surgir durante la instalación y configuración más el aprendizaje de Selenium. Era por ese motivo que el segundo milestone no presentaba explícitamente nuevos objetivos relacionados con el código, ya que existía la posibilidad de que no se cumplieran todos los objetivos del primer sprint y éstos se tuvieran que finalizar durante el segundo.

En el primer sprint los tests que se determinó realizar era sólo automatizar un par de tests relativamente simples, de cara a poder familiarizarse lo suficiente con Selenium como para que la automatización que se realizaría en el tercer sprint no presentara demasiadas complicaciones.

6.2 Herramientas

El proyecto ha sido realizado en Eclipse, utilzando el lenguaje Java, junto a los siguientes componentes de software:

- Selenium Webdriver [4], la herramienta con la cual se testea la web.
- TestNG [5], el framework de test en el que se ha creado la test suite y que además facilita la interacción con Jenkins.
- Chromedriver.exe [6], geckodriver.exe [7] (el cual representa a Firefox) e internetexplorerdriver.exe, tres ejecutables que necesita Selenium Webdriver para poder utilizar el navegador correspondiente.
- Jenkins, el software de integración continua en el que se desplegará el programa, el cual permite obtener feedback rápido de si una nueva versión de un software supera los tests o no antes de desplegarlo al repositorio/servidor de producción.

Pese a que Jenkins no es un elemento indispensable para el funcionamiento de este proyecto, se ha incluido debido a que la integración continua es cada vez más común en los equipos de testing y ésta es una herramienta comúnmente utilizada para ello, y en este TFG se desea simular de manera fidedigna el tipo de infraestructura y procedimientos que podría encontrarse dentro del departamento de testing de una empresa.

7 Desarrollo e implementación

El desarrollo hasta alcanzar el primer milestone consistió en instalar e integrar los plug-ins mencionados (siendo lograr que el driver de internet explorer funcione lo más complicado, puesto que hoy en día está en desuso).

Una vez logrado esto, se creó una serie de clases que representaban a cada una de las páginas más significativas de *adioso.com*, exceptuando la página principal. Esto se debe a que el resto de páginas eran sencillas (básicamente eran similares a un blog, sin ningún tipo de funcionalidad aparte de mostrar texto) y permitirían familiarizarse con Selenium y automatizar tests. La página principal, y por tanto la funcionalidad de buscar vuelos, se testearía en el segundo milestone y dichos tests se automatizarían en el tercer milestone con la experiencia adquirida.

Para las otras páginas se realizó un test consistente en buscar todos los links contenidos en ellas y comprobar si

había algún error 404 en algún link. Para ello se creó la siguiente función, en la cual se busca todos los elementos dentro del tag "*<a>*", se obtiene la propiedad *href*, se comprueba mediante una expresión regular si tienen el formato adecuado y se establece una conexión HTTP, obteniendo el código recibido. Al final de esta función se muestran todos los links que devolvían un error 404.

```
//This method receives a driver and an URL, and opens that given URL on the
//designated browser, grabs all the links that can be accessed
//from that URL and checks if they return a 404 or not.
public static void test404(WebDriver driver){

    //This boolean and list will be used to display all the 404's found during the test
    List<WebElement> list404 = new ArrayList<WebElement>();
    boolean found404 = false;

    //We find all elements within that url that have a link
    List<WebElement> elementsList = driver.findElements(By.tagName("a"));

    //Using getResponseCode, we check the link on every element
    //to see whether it works or not,and print them
    for (int i=0; i < elementsList.size(); i++) {
        boolean isValid = true;
        //Before proceeding we check if that Element is, indeed, an URL
        if(isUrl(elementsList.get(i).getAttribute("href"))) {
            //Once we confirmed it is an URL, we proceed to try to connect to it
            isValid = getResponseCode(elementsList.get(i).getAttribute("href"));
            if (isValid) {
                System.out.println("Works:" + elementsList.get(i).getAttribute("href"));
            }
            if(!isValid){
                //We add the 404 link to the list, and put the boolean on true
                //so we know that we'll have to display that list
                list404.add(elementsList.get(i));
                found404= true;
            }
        }
    }
    //If we found at least one 404, we go through this loop and
    //display all links that were found with a 404 from within that page
    if (found404){
        System.out.println("The following 404's have been found: \n");
        for(int i=0; i < list404.size();i++) {
            System.out.println(list404.get(i).getAttribute("href"));
        }
        Assert.fail("One or more 404's have been found");

    }


}
```

Figura 3: Función test404

Para tres páginas concretas se realizó otro test, ya que tenían una sección iluminada en el header (que era igual para las tres) según en qué página se encuentra el usuario. Es decir, si se está en la página de "About", la palabra "About" en el header está iluminada mientras "Careers" y "Press" están oscurecidas. Sin embargo, en estas páginas dicha implementación era errónea. En "About", la palabra "About" se iluminaba correctamente. En "Careers" se iluminaba la palabra "Press" y en "Press" ninguna palabra se iluminaba. La comprobación de esto se podía automatizar en Selenium, gracias a su función "*element.getCssValue("color")*", la cual lee las propiedades CSS del elemento HTML obtenido y, en este caso, nos devuelve el color, estandarizado en formato RGBA. Hay que remarcar que hay un bug en esta comprobación. Por algún motivo que se desconoce y no se ha logrado solventar, Firefox devuelve el color de la palabra "About" en formato RGB. Esto no ocurre ni para el resto de palabras ni en el resto de navegadores.

Figura 4: Header en la página "About"

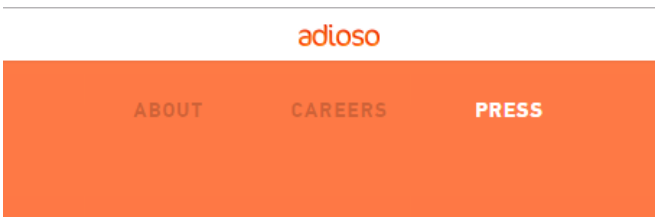

Figura 5: Header en la página "Careers"

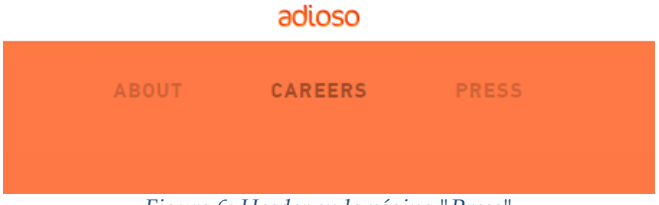

Figura 6: Header en la página "Press"

Una vez este código era funcional, se procedió a desplegarlo en Jenkins [8]. Primero es necesario arrancar el servidor de Jenkins, introduciendo en consola el comando

"java –jar jenkins.war"

Jenkins se ejecutará por defecto en el puerto 8080, así que podemos acceder a su dashboard introduciendo "*localhost: 8080*" en nuestro navegador. Se creó un nuevo proyecto en Jenkins, al cual se le proporcionó el path de nuestro workspace. Además, se especificó que ejecutara el siguiente comando:

" java -cp C:\[Path to libs in the project]*; C:\[Path to bin in the project]\ org.testng.TestNG testng.xml"

Como se puede observar, en el comando se proporciona el path a dos carpetas (la que contiene todos los .jar y la que contiene los binarios) y, además, se menciona a "*testng.xml*". Como ya se ha detallado previamente, TestNG es el framework de test que se utiliza en este proyecto y es fácilmente integrable con Jenkins. Esto se debe a este archivo .xml que genera, el cual especifica qué clases del código se desean testear. En nuestro caso todos los builds que se han realizado dan error, no por no ejecutarse correctamente, si no debido a que algunos tests fallan porque, efectivamente, detectan errores en la página web, así que no hay ningún build "con éxito" en el cual se hayan detectado 0 errores. Posteriormente se modificaron los ajustes para que Jenkins notificara por correo de los resultados de cada build realizado. Aunque puede personalizarse el contenido de estos mails, se decidió dejar el contenido por defecto, el cual muestra la salida por consola.

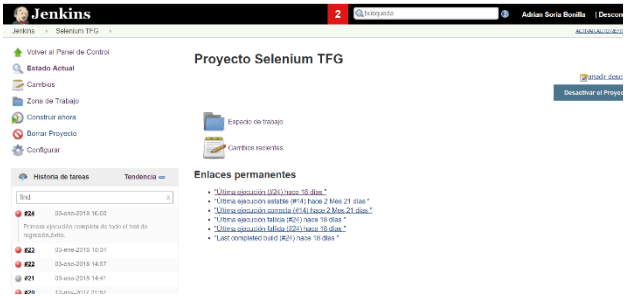

Figura 7: Muestra del aspecto del proyecto en Jenkins. Se puede observar cada ejecución completa y las salidas por consola de cada una

Una vez realizadas todas estas tareas, se dio el primer milestone por alcanzado y finalizado, por lo que se procedió al segundo milestone: la documentación de tests sobre la funcionalidad principal de la página, la búsqueda de vuelos.

Los tests están documentados en un archivo Excel, en una tabla con los siguientes campos:

- Identificador numérico del test.
- Descripción del test o la motivación por la que se utilizan esos valores concretos.
- Los valores en cada campo del sistema de búsqueda para ese test en concreto(Origen, Destino, Fecha de Salida, Fecha De Retorno, Número de pasajeros adultos, número de pasajeros menores de edad y la clase de los billetes (Business, Economy o First).
- Resultado esperado de la página.
- Resultado obtenido de la página.
- Link a la URL resultante de la búsqueda.
- Status del test (PASSED, FAILED o QUESTIONABLE)
- Un apartado de observaciones para conjeturas sobre los resultados, clarificaaciones o detalles sobre tests posteriors que se hayan realizado para investigar en mayor profundidad los resultados de ese test.

Este archivo Excel está dividido en 3 pestañas, según qué categoría general de los campos de búsqueda se está probando: Origen y destino, puesto que ambos encapsulan lugares, fecha de salida y fecha de retorno, puesto que ambas tratan con fechas, y finalmente cantidad de billetes y su categoría, que tiene una sección propia.

En el apartado de resultados se hablará en mayor profundidad sobre los resultados de dichos tests.

La redacción de estos tests se finalizó correctamente y por lo tanto se consideró el segundo milestone por alcanzado, por lo que se procedió al tercer y último milestone: la automatización de los tests redactados en el segundo milestone.

Para ello se crearon dos clases más en el código creado para el milestone uno: *MainPage*, el PageObject que representa a la página principal, y *MainPageTests*, la clase que contiene todos los tests que se realizan sobre ella. Como ya se mencionó, MainPage posee una serie de funciones que MainPageTests llama para obtener el elemento html que le interesa (en este caso, los input fields para cada campo de la búsqueda y el botón para iniciar la búsqueda), para que posteriormente MainPageTests pueda realizar los tests que le interese manipulando estos elementos, sin importar los cambios futuros que pueda haber en la estructura de la página, lo que solo repercutiría en modificar el contenido de los métodos que MainPage usa para poder proporcionar sus elementos.

La comprobación de los resultados de cada búsqueda no es trivial y es preferible que lo valide un humano en vez de una máquina, por lo que se ha usado un híbrido entre testing automatizado y testing manual: los tests se ejecutan automáticamente, toman una captura de pantalla del resultado y la guardan en una carpeta creada dinámicamente en cada ejecución del código. Estas carpetas se encuentran dentro de la carpeta *mainPageScreenshots* dentro del workspace. Cada una de estas carpetas tiene un nombre en función de la fecha y hora en la que se ha ejecutado el código, y cada una tiene 3 carpetas más en su interior: una para cada uno de los 3 tipos de campo que se han testeado, igual que las 3 pestañas presentes en el archivo Excel. Por lo tanto, se puede programar una ejecución de todo el test de regresión, y posteriormente se puede ir a la carpeta correspondiente creada a raíz de esa ejecución y comprobar los resultados que muestra la captura de pantalla del resultado de la búsqueda de cada test de cara a validar si ha fallado o no.

Puesto que los tests en la página principal siguen la misma estructura (realizar una búsqueda y comprobar los resultados), se ha creado una función genérica llamada *genericTest,* a la cual se llamará para cada test individual. Esta función recibe muchos parámetros, los cuales son:

- Los strings a introducir en el origen, destino, fecha de salida y fecha de retorno.
- El número de adultos y niños.
- El tipo de billete.
- El string a introducir en el campo de billete (aunque posteriormente se comentará como tiene un impacto nulo en los resultados).
- El path en el que se encuentra la carpeta en la que debe guardar la captura de pantalla (path hasta la carpeta de origen y destino, fechas de salida y retorno o de tipo billete).
- El string con el nombre que dará a la captura de pantalla, que será "testX", siendo X el número del test tal y como aparece en el archivo Excel.

Esta función pide a *MainPage* que le proporcione los WebElements que necesita (como ya se mencionó, cada uno de los input fields y el botón para iniciar la búsqueda), escribe en ellos los valores de cada string, realiza una búsqueda, espera implícitamente 15 segundos, toma una captura de pantalla y la guarda en

la carpeta que corresponda.

```
//Since all tests are related with searches, this function performs the basic actions for every sear
//It retrieves the input fields in the main page, writes into them what has been specified into the
//After waiting a few seconds for the search to load, it takes a screenshot and saves it in the righ
public static void genericTest(String originString, String destinationString, String departureString,
 int childrenNumber, int ticketClassNumber,String ticketFieldString,String folderPath, String screen
    //We make sure we're in the main page
    driver.get("http://adioso.com");

    //We ask the MainPage class to provide us each one of its input fields, and write in them
    WebElement origin = MainPage.getOriginField(driver);
    origin.sendKeys(originString);

    WebElement destination = MainPage.getDestinationField(driver);
    destination.sendKeys(destinationString);

    WebElement departure = MainPage.getDepartureField(driver);
    departure.sendKeys(departureString);

    WebElement returning = MainPage.getReturnField(driver);
    returning.sendKeys(returningString);

    //The 3 following fields(adults,children and class) aren't written into,
    //but rather, they are a menu you select from,
    //so they are handled differently
    //Index 0 in Adults is 1 Adult. So if we want 3 adults, we'd go into index 2.
    Select adults = MainPage.getAdultsTicketField(driver);
    adults.selectByIndex(adultsNumber - 1);

    //However, index 0 in children is 0, so in this case the number
    //of children matches the index
    Select children = MainPage.getChildrenTicketField(driver);
    children.selectByIndex(childrenNumber);

    Select ticketClass = MainPage.getClassTicketField(driver);
    ticketClass.selectByIndex(ticketClassNumber);

    //Once all fields have been filled, we click the search button
    WebElement submit = MainPage.getSubmitField(driver);
    submit.click();

    driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);

    //After waiting a few seconds, we take a screenshot.
    //The destination folder is the path(workspace folder for screenshots +  current date)
    // +  whether it's in the Origin&Destination, FlightDates or TicketInfo folder)
    //We name the screenshot according to the test it was taken in, and save it as a png
    File screenshot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
    File destinationFolder = new File(folderPath + "/" + screenShotName +".png");
    FileUtils.copyFile(screenshot, destinationFolder);

}
```

Figura 8: Función genericTest

Cada test concreto realizaba una llamada a esta función y le pasaba los parámetros correspondientes. Este tipo de estructura es fácilmente mantenible y escalable, por lo que resulta sencillo añadir nuevos tests, aunque se podría considerar separarlos en tres clases diferentes (una para cada pestaña del archivo Excel) si eventualmente se sigue escalando y alcanza un tamaño demasiado grande.

Precisamente esta clase, *MainPageTests*, es la que más uso hace de los tags que proporciona el framework de TestNG. Además de usar el tag "@Test" en cada función que deseemos que sea tratada como un test, también hace uso del tag "@BeforeClass" (para especificar que un método ha de ocurrir antes de todo lo demás en esa clase, y en este caso el método era el que se ocupaba de la creación dinámica de las carpetas para guardar las capturas de pantalla), y el tag "@AfterClass", en el cual cerramos el driver una vez hemos finalizado todos los tests. TestNG ofrece más tags, los cuales demuestran tener potencial para poder estructurar un proyecto versátil, mantenible y escalable.

Con el código funcional y Jenkins funcionando correctamente, creando además las carpetas ya mencionadas para cada ejecución y guardando las capturas de pantalla, se dio el tercer milestone por alcanzado con éxito y, por tanto, todos los objetivos y con ellos el proyecto.

8 Resultados

En total se realizaron 53 tests durante el milestone 2, más 33 realizados en el milestone 1 (aunque estos últimos són permutaciones del test de 404 y el test de color realizados para cada página y en distintos navegadores, así que no hay demasiadas diferencias entre estos 33), además de tests adicionales mencionados en el archivo excel para indagar más o confirmar hipótesis sobre los 53 realizados en el segundo milestone. En ocasiones se hace mención explícita de estos tests adicionales en el apartado de observaciones.

Los resultados de los tests realizados sobre la página fueron inesperados.

Comenzando con el milestone uno, se pudieron observar dos defectos. El primero, un error de iluminación en una serie de palabras en la cabecera de tres páginas. Sin embargo, el segundo y de mayor impacto era que no existía página en *www.adioso.com* que no tuviera un link hacia una página 404. El motivo es que la dirección www.adioso.com/experiences/ parece no existir, pero pese a todo se proporciona un link a ella y dicho link se encuentra en el footer presente en todas las páginas.

Este defecto de bastante calibre implicaba, de manera implícita, que era probable que hubiera otros errores relativamente graves en la implementación.

Esta suposición se confirmó durante el testing de la funcionalidad de búsqueda de vuelos durante el segundo milestone. Los resultados han sido considerablemente caóticos, y se procederá a hablar de ellos por categorías: los relacionados con origen y destino, los relacionados con las fechas de salida y retorno y los relacionados con los billetes en sí.

Respecto origen y destino, el resultado más significativo es que, en caso de recibir una palabra desconocida, el sistema puede llegar a aproximarla a la palabra más cercana conocida, pero no siempre se da el caso, y en ocasiones hay tantos grados de separación que sería mejor no realizar la aproximación. Esto se ve con "Jungst", palabra ficticia, la cual aproxima a "Rongshui", una región en China, pese a que el parecido es bastante escaso. No siempre aproxima las palabras, sin embargo, porque en casos como "Trunglorpul", otra palabra ficticia, la cual elige no aproximar. Sin embargo, el criterio que el sistema usa para aproximar no es demasiado claro, y sería interesante poder ver a nivel código cómo decide si debería aproximar o no. Definitivamente no está relacionado con el tamaño del string, puesto que introduciendo 1500 veces únicamente el carácter "a", el sistema elige aproximarlo a "Shijak AL". Cuando recibía un campo en blanco en el origen, usaba un valor por defecto, posiblemente basado en la IP del cliente (en este caso, el valor por defecto que usaba era "Madrid ES", pese a tener opciones para Barcelona). Sin embargo, el sistema siempre impondrá que se introduzca un destino, y muestra mensaje de error en caso negativo, o podía llegar a usar el valor "Anywhere", el cual mostraba cualquier destino disponible. En ocasiones el mensaje de error mencionaba que origen y destino no podían ser el mismo, pese a que no había ninguna similitud entre ellos, como se

puede ver en la figura 10, donde el origen es Madrid ES pero el destino es el string "% + - /".

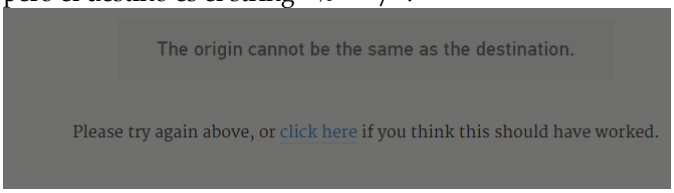

Figura 9: Muestra del resultado de introducir el string "% + - /" en destino. Nótese como el mensaje de error no corresponde con el tipo de error

Respecto las fechas de salida y retorno, son la categoría con mayor cantidad de inconsistencias. Para empezar algunos de los propios valores por defecto que ofrece no funcionan como cabría esperar. Por ejemplo, el valor "Anytime", el cual sólo está presente en el apartado para la fecha de salida y no la de retorno, a veces muestra 0 resultados, en especial cuando se elige como opción de destino alguna opción con múltiples posibilidades, como Europa, pese a que una búsqueda individual sí que muestra.

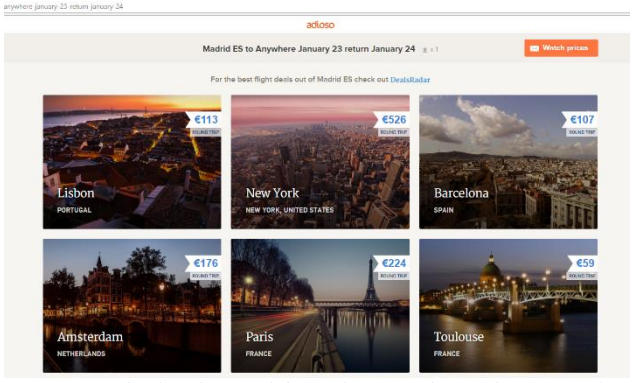

Figura 10: Resultados de una búsqueda normal en adioso usando "Anywhere" como destino y fechas de salida y retorno razonables

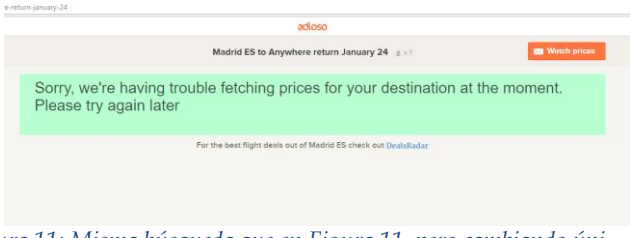

Figura 11: Misma búsqueda que en Figura 11, pero cambiando únicamente la fecha de salida por "Anytime"

Otra decisión es la de ofrecer la opción "One Way" en fecha de retorno. Pese a que esto tiene sentido, es inconsistente respecto la implementación hecha con "Anytime". El sistema reconoce "Anytime" como una palabra reservada y con un comportamiento reservado para ella, pero "One Way" no es utilizada como palabra reservada. Se puede afirmar esto pese a no poseer el código por el comportamiento del sistema: cualquier string no reconocido en el campo de fecha de retorno se ignorará y se tratará como un campo vacío, y el sistema ofrece billetes sólo de ida si la fecha de retorno está vacía. Introducir "One Way" tiene el mismo comportamiento y no hay ningún cambio ni en los resultados ni en la URL de éstos. Es decir, al seleccionar la opción "One Way", es irrelevante que el sistema escriba "One Way" en el campo de fecha de retorno o cualquier otro tipo de palabra. Por lo tanto, o se debería obligar a que si el string no se reconoce o es un campo blanco muestre un error y muestre billetes sólo de ida únicamente si se usa "One Way", haciendo de ella una palabra reservada, o se elimina la opción "One Way" completamente y que introduciendo cualquier string no reconocido o vacío muestre sólo vuelos de ida.

El caso de las palabras reservadas no es el único problema relacionado con las fechas. El sistema suele cometer errores si hay un cambio de año (y de hecho no realiza ningún tipo de comprobación sobre los cambios de año, así que sería más óptimo para el sistema que directamente ofreciera un calendario más limitado), da falsos positivos, los períodos máximos de duración del viaje que ofrece son demasiado grandes y ni los puede procesar, y hay elementos cuestionables como que se pueda tener una fecha de retorno aproximadamente 3 horas después de la llegada del avión de ida.

Respecto los tipos de billete, el campo de texto tiene impacto nulo en la búsqueda, y sólo importa lo que se selecciona en el menú, motivo por el cual no tiene sentido la presencia de ese campo. Se ha probado con strings de diferentes tipos, seleccionar ciertas opciones en los menús pero escribir en el campo de texto algo diferente (por ejemplo, seleccionar 5 adultos en el menú pero escribir "3 Adults" en el campo de texto), y ninguna de estas opciones ha impactado. Esto nos lleva a asumir que únicamente utiliza el campo de texto para mostrar lo que se ha elegido en los menús, pero en tal caso no debería permitirse que fuera modificable por el usuario. Además, la cantidad máxima de pasajeros que se puede elegir, pese a ser una opción que ofrece el sistema, no puede ser procesada. Es decir, si se eligen 6 adultos y 6 niños, no encuentra tal cantidad de billetes.

Como nota adicional, no hay validación de tamaño en ninguno de los campos de texto, por lo que con 4000 caracteres se puede causar un overflow y recibir un "Bad Gateway Error". Además, puede reconocer algunos caracteres especiales y enviar mensaje de error al respecto, pero hay otros que ignora completamente, motivo por el cual en vez de enviar un mensaje de error sobre ellos actúa como si hubiera recibido un campo en blanco.

Resultante de este proyecto se ha generado el archivo Excel en el que están documentados todos los tests realizados y el código en el cual dichos tests han sido automatizados.

En el anexo a este documento se proporciona una representación más visual de los tipos de tests realizados y sus tipos. El archivo Excel no se incluye en el anexo debido a la cantidad de texto y columnas, y no cabría de forma leíble en el documento, por lo que éste se encontrará dentro del dossier de este TFG.

9 Conclusiones

Definitivamente la página presenta una cantidad preocupante de problemas e inconsistencias, en especial referente a fechas, por lo cual no es una web de confianza para la búsqueda de vuelos hasta que no se realicen modificaciones extensivas a su funcionamiento, posiblemente incluso empezando de nuevo. Parece ser que la página ha sido abandonada, o que se han realizado muy pocas labores de mantenimiento o comprobaciones, tal y como se puede ver en defectos bastante obvios pero sencillos de arreglar como la iluminación errónea de los headers como el hecho de que la página de "Experiences" mostrada en el footer parezca no existir realmente.

Como no se logró contactar con los propietarios de la página, no ha sido posible realizar testing de caja blanca, seguridad o de carga, y no se les ha podido ofrecer el test de regresión realizado durante el proyecto de cara a que fuera usado como soporte durante su proceso de mejora de la página. Dada la cantidad de resultados extraños obtenidos durante el testing de la funcionalidad de búsqueda de vuelos, habría sido muy interesante poder ver el código y las comprobaciones que realizaba, porque no se entiende muy bien la lógica tras ciertas comprobaciones o la secuencia de eventos que ha sucedido para obtener algunos resultados (como el mencionado del mensaje de error sobre "Origen y Destino no pueden coincidir" cuando no coinciden en absoluto), además de que con ello se podría haber prescindido del híbrido entre testing manual y automatizado con las capturas de pantalla guardadas en carpeta y se podría haber realizado una labor de automatización pura, ya que se podrían realizar comprobaciones en las diferentes estructuras de datos, objetos, etc, que hubiera dentro del código.

Selenium ha demostrado ser una herramienta intuitiva de usar y con mucho más potencial de lo esperado. No se preveía la capacidad de comprobar colores o la facilidad para la toma de capturas de pantalla, y su uso y manejo sencillos combinados con su potencial justifican por qué es uno de las herramientas de automatización más utilizadas.

10 Agradecimientos

Quería agradecer a Xavier Otazu Porter, director de mi TFG pero también profesor que me impartió la asignatura de Test y Calidad del Software, la cual me permitió descubrir aspectos del testing y que despertó mi interés por QA.

Quiero agradecer también a la comunidad de la web "uTest", web para crowdsourcing de test que cuenta con foros y una comunidad activa con la cual descubrí muchas facetas sobre los tests y experiencias de profesionales en el sector.

Por último, pero no por ello menos importante, quiero agradecer a mis padres y amigos por proporcionarme apoyo moral y ánimos, elementos sin los cuales el desarrollo de este proyecto habría sido imposible.

Bibliografía

[1] QA Intelligence Blog y TeaTime with Testers "State of Testing Report 2017" [En línea] Disponible en: http://qablog.practitest.com/wp-content/uploads/2017/03/State_of_testing_2017_final_report.pdf

[2] Sogeti, Capgemini y Micro Focus "World Quality Report 2017 – 2018" [En línea] Disponible en: https://www.sogeti.es/globalassets/global/downloads/testing/wqr-2017-2018/wqr-2017_country-pullouts_southern-europe_v2_secure.pdf

[3] Selenium HQ "Page Object Design Pattern" [En línea] Disponible en: https://github.com/SeleniumHQ/selenium/wiki/PageObjects

[4] Selenium HQ, Descarga para Selenium Webdriver e InternetExplorerDriver en Java [En línea] Disponible en: http://www.seleniumhq.org/download/

[5] TestNG, Descarga para el framework de TestNG [En línea] Disponible en: http://testng.org/doc/download.html

[6] Google Chrome, Descarga para el driver ChromeDriver [En línea] Disponible en: https://sites.google.com/a/chromium.org/chromedriver/downloads

[7] Mozilla Firefox, Descarga para el driver GeckoDriver [En línea] Disponible en: https://github.com/mozilla/geckodriver/releases

[8] Jenkins, Descarga para el servidor para integración contínua Jenkins [En línea] Disponible en: https://jenkins.io/download/

ANEXO – VISUALIZACIÓN DE LOS TESTS:

En la siguiente tabla se muestran las diferentes categorías de test realizadas y los resultados de cada uno.

Como podemos ver, la cantidad de tests marcados como "FAILED" y "QUESTIONABLE" sobrepasa en una cantidad bastante preocupante a los tests marcados como "PASSED", lo que delata la implementación caótica del sistema de búsqueda.

Tipo de test	Cantidad realizada	# de PASSED	#de FAILED	# de QUESTIONABLE
Overflows	5	0	5	0
Valores por defecto	14	5	4	5
Secuencias de caracteres extraños y palabras desconocidas para el sistema	27	10	7	10
Otros (casos estándar, coincidencias entre valores, inconsistencias entre fechas)	7	3	3	1

Respecto a qué parámetro de búsqueda causa más problemas, el siguiente gráfico muestra cómo, sin lugar a dudas, el mayor problema son las fechas. Pese a que el sistema funciona correctamente para una búsqueda relativamente estándar, sin utilizar valores por defecto del sistema ni con cambios de año de por medio y con fechas de salida y retorno fijas, cualquier desviación de este escenario tiende a generar problemas