Desarrollo de un Sistema de Automatización de Pruebas de Regresión en una Plataforma Web

Karol Daniela Alzate Mejía

Informe de práctica empresarial, como requisito para optar por el título de Ingeniero de Sistemas.

Asesor
Danny Alexandro Múnera Ramírez, PhD en Informática

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín, Antioquia, Colombia
2023

Cita	Alzate Mejía [1]
Referencia Estilo IEEE (2020)	[1] Alzate Mejía, “Desarrollo de un Sistema de Automatización de Pruebas de Regresión en una Plataforma Web”, Semestre de industria, Ingeniería de Sistemas, Universidad de Antioquia, Medellín, Antioquia, Colombia, 2023.

TABLA DE CONTENIDO

LISTA DE FIGURAS

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

IEEE	Institute of Electrical and Electronics Engineers
E2E	End-to-End
UI	User Interface
CI	Continuous Integration
UdeA	Universidad de Antioquia
POM	Page Object Model

RESUMEN

La automatización de pruebas es una alternativa que muchas empresas están comenzando a utilizar en sus equipos para disminuir la cantidad del tiempo que la ejecución de sus pruebas pueda tomar y tener una mayor cobertura de estas. Este informe describe la aplicación de un sistema de automatización de pruebas para una empresa de publicidad que, como muchas del mercado, está empezando a ver la automatización como una solución a estos problemas. Las pruebas de regresión de una de sus plataformas fue el foco principal, estas eran pruebas que mayormente se realizaban en la Interfaz de Usuario, por lo que se usó Selenium WebDriver, TestNG, Java y Maven para la realización de éste sistema, logrando automatizar 9 de los 14 casos de prueba definidos como regresión en el equipo. Gracias a que la empresa hace uso de Integración Continua, se validaron las pruebas a través de la creación de un pipeline en Jenkins que corrió las pruebas de manera exitosa.

***Palabras clave* — Automatización de pruebas, Selenium WebDriver, pruebas E2E, pruebas de regresión.**

ABSTRACT

Automation testing is an alternative that many enterprises are considering using in their teams to decrease the amount of time that test execution can take and to have higher test coverage. This report describes the application of a test automation system for an advertising company that, as many of the market, is beginning to see automatization as a solution to problems. The regression testing of one of its platforms was the main focus, these tests were mainly performed in the User Interface, so Selenium WebDriver, TestNG, Java, and Maven were used for the implementation of this system, managing to automate 9 of the 14 test cases defined as regression by the team, and thanks to the fact that the company makes use of Continuous Integration, the tests were validated through the creation of a pipeline in Jenkins that ran the tests successfully.

***Keywords* — Test Automation, Selenium WebDriver, E2E testing, Regression testing.**

I. INTRODUCCIÓN

La tecnología se encuentra presente en muchas áreas laborales en la actualidad, tales como el marketing y la publicidad. Haciendo que en éstas haya elementos como sitios web que difunden su información o tengan herramientas internas que ayuden a implementar varios de sus procesos. Este es el caso de la compañía, que por razones de confidencialidad llamaremos de ahora en adelante *EmpresaA*, la cual es una empresa especializada en marketing con más de un siglo de experiencia. *EmpresaA*, como muchas de las empresas actuales, están incorporando cada vez más el uso de las plataformas web para mejorar sus procesos.

EmpresaA tiene actualmente siete plataformas web internas, y en ellas se hace uso de las buenas prácticas, por lo que cada funcionalidad nueva que se desarrolla en el proyecto es testeada por testers manuales, los cuales proceden a generar y testear los casos de prueba. Algunas de las plataformas son *PlatafomaB*, un landing page que contiene información que debe ser resaltada, como algunas noticias o personas, e incluye el manejo de roles. *PlatafomaC*, contiene las noticias relacionadas a la empresa, con ella se relaciona una página que se encarga de administrar, actualizar, entre otras acciones, dichas noticias. *PlataformaD*, muestra todos los empleados de *empresaA*, con su respectivo perfil, y se pueden generar diferentes interacciones como chats o llamadas. Y la última, *plataformaE*, es un buscador de cualquier información de la empresa, es decir que puede traer la información brindada por las plataformas anteriormente mencionadas.

Los casos de prueba generados por los testers manuales se pueden clasificar como parte de la regresión o no. Las pruebas de regresión son las que se hacen, para algunos equipos, al final del sprint (ciclo de desarrollo de SCRUM) o ya dependiendo de cómo lo exija el negocio, como es el caso de *EmpresaA*. Según la IEEE las pruebas de regresión son pruebas que deben repetirse cuando se realiza un cambio en un producto de software ya testeado, evaluando la naturaleza del cambio para determinar el posible impacto que éste pueda tener y correr los casos de prueba que se adapten a él [1]. Como el proyecto contiene varias plataformas y éstas ya llevan funcionando un buen tiempo, se han generado aproximadamente 1000 casos de prueba; la cantidad de casos de prueba que deben correrse cuando se hace la regresión, pueden variar dependiendo de los cambios agregados o el tiempo en que se haga la regresión.

Pero, debido a que la idea es encontrar que no se hayan incluído errores en las funcionalidades principales del sistema, hay un conjunto de pruebas que están en un apartado

llamado *Regression*. Esto lo que significa es que todos los casos allí definidos deben ser ejecutados de manera obligatoria en todas las regresiones que se realicen. Para *PlataformaC*, éste apartado cuenta actualmente con 14 casos de prueba, y éstos son pruebas E2E. Las pruebas E2E son un método que evalúa todo el flujo de la aplicación (para este caso el flujo de *PlataformaC*) y verifica que todas las funciones respondan como se espera en un escenario de la vida real [2]. Éstas se caracterizan por ser pruebas que toman mucho más tiempo y esfuerzo que las demás, por lo que éstas fueron el enfoque principal de esta práctica, con el fin de disminuir los tiempos de ejecución en las pruebas de regresión y ayudar a los testers manuales automatizando estos casos.

En este trabajo, se creó un sistema de automatización para las pruebas de regresión de la *plataformaC* de *EmpresaA*. Al ser este un proyecto en el que se testean plataformas web, se usaron herramientas de automatización de pruebas enfocadas a la parte de interfaz de usuario. Más específicamente, Selenium WebDriver con Java como el lenguaje de programación, Maven para la construcción y gestión del proyecto y por último TestNG, para la creación y definición de la ejecución de las pruebas. Con este sistema se lograron automatizar 9 casos de prueba, de los 14 catalogados como regresión. El funcionamiento del proyecto fue validado a través de la creación de un pipeline en Jenkins que corrió todas las pruebas de manera exitosa.

El documento está dividido en seis capítulos (incluyendo este capítulo de Introducción) que muestran el paso a paso de la realización de este proyecto. Exponiendo cuáles fueron los objetivos, tanto el general como el específico, en el capítulo 2, el marco teórico en el capítulo 3, qué metodología se utilizó en el capítulo 4, los resultados que se obtuvieron en la terminación del proyecto en el capítulo 5 y las respectivas conclusiones en el capítulo 6.

II. OBJETIVOS

A. Objetivo general

Implementar un sistema de automatización de pruebas que permita cubrir los casos de pruebas de regresión en las funcionalidades más fundamentales de la plataforma web *PlataformaC* de la *EmpresaA*.

B. Objetivos específicos

- Analizar el código y las pruebas manuales existentes en el proyecto.
- Definir qué patrones de diseño y herramientas son las que mejor se adaptan a las necesidades del proyecto.
- Automatizar los casos de prueba de regresión relacionados a la *plataformaC* de *EmpresaA*.
- Desarrollar una plataforma que soporte el flujo de automatización de pruebas.

III. MARCO TEÓRICO

Este marco teórico aborda los conceptos y herramientas que son relevantes para el entendimiento de los procesos y decisiones tomadas en la realización de este sistema de automatización, a continuación se describen los diferentes tipos de pruebas que existen en el desarrollo de software, en qué consiste la automatización de pruebas y diferentes herramientas que pueden ser utilizadas para realizar este proceso.

A. Tipos de pruebas a nivel de Software

Para la entrega de un producto de software confiable y de calidad, es importante que este sea probado previamente a una entrega a producción, cuando hay funcionalidades nuevas, o una antigua ha sido actualizada. Gracias a esta práctica, existe una clasificación de las pruebas que se pueden realizar en un sistema de software. Algunas de estas son, las pruebas de carga que se encargan de evaluar el comportamiento de un sistema bajo ciertas condiciones [3], las pruebas de regresión son las que se ejecutan para asegurar que no se han introducido nuevos errores en un código previamente probado[4], las pruebas de desempeño prueban que el sistema cumpla con un rendimiento específico[5], entre otras. Al haber tantas, depende más de las necesidades del equipo cuáles de estas se van a usar, pero es más común que se encuentren las pruebas de regresión, mínimamente, en un proceso de testing de un sistema.

B. Automatización de pruebas

La automatización de pruebas consiste en, a través de la escritura de scripts o uso de herramientas de software, automatizar un proceso que se realiza de manera manual[6]. Las pruebas manuales conllevan mucho tiempo en ejecutarse, y al ser un proceso que se debe repetir cada cierto tiempo, es bastante susceptible a errores humanos, implicando así que en los proyectos no se puedan encontrar bugs que pueden estarse integrando en el ambiente de producción. Con la automatización, estos problemas disminuyen ya que las pruebas se pueden correr cuantas veces sean necesarias disminuyendo la carga sobre las personas que ejecutan estas pruebas de manera manual, permitiendo así que éstas se pueden enfocar en pruebas que contengan funciones más delicadas o la ejecución de otros tipos de pruebas.

En la figura 1, se puede apreciar la pirámide de Cohn, en ella se encuentran las pruebas unitarias en la base, estas pruebas como indica su nombre, verifican que las unidades individuales de código sean válidas para ser usadas [3]. Son las pruebas más rápidas de realizar, lo que a su vez las hace menos costosas, y son las que miran los componentes de una manera más aislada. Le siguen las pruebas de integración que se usan para evaluar la capacidad de comunicarse dos o más sistemas integrados entre sí[1]. Por último, en la parte superior se encuentran las pruebas E2E, que dicho de otra manera significa End-to-End (de extremo a extremo), en estas se encuentran las pruebas de UI que simulan las interacciones del usuario[7], esto hace que a su vez sean las más lentas en realizar, lo cual genera un costo más grande, pero también es la que contiene mayor integración, y cobertura de las pruebas.

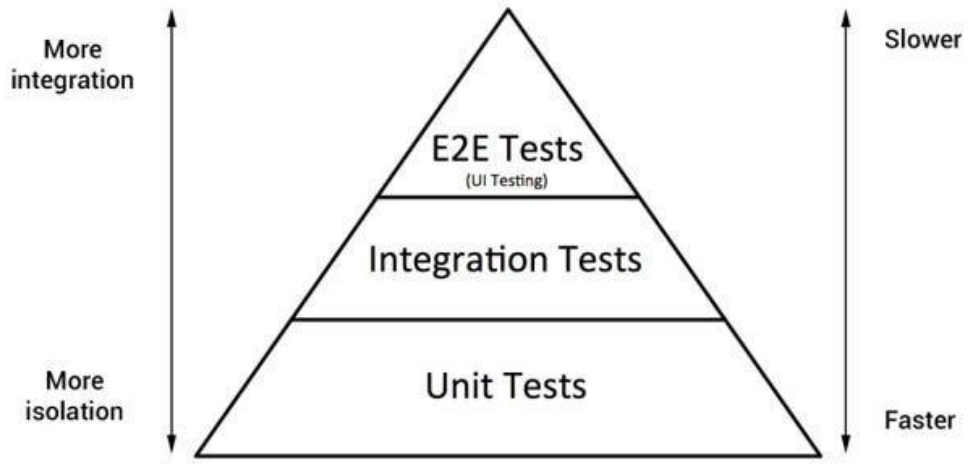


Figura 1. Pirámide de Cohn

C. Herramientas de automatización

Para cada una de las pruebas mencionadas anteriormente, hay diferentes herramientas que se pueden implementar. Por ejemplo, para las pruebas de UI, existen herramientas como Cypress, Playwright o Selenium. Para las pruebas de API, hay herramientas como Rest-Assured, Katalon o Karate. Para las pruebas mobile, está Appium con Selenium o Kobiton. Estas se eligen dependiendo de qué características y necesidades tenga el proyecto, porque puede ser prioridad el usar ciertas pruebas o no, o dependiendo de su sistema interior, es mejor usar frameworks que usen un lenguaje de programación específico.

D. CI y la automatización de pruebas

La integración continua, según Martin Fowler, es una práctica en la que los equipos de desarrollo software integran su código de manera constante, y cada integración se verifica mediante una compilación automatizada, en la que se incluyen pruebas, con el fin de detectar errores de manera temprana[8]. Aquí es donde las pruebas automatizadas hacen un gran aporte, ya que al estarse ejecutando de manera automática en un pipeline cada que se añade un nuevo cambio, hay un persona que no está corriendo estas mismas pruebas manualmente y el proceso se puede realizar de una manera más rápida, permitiendo la entrega de un producto con mayor calidad y listo para ser deployado, en caso de que las pruebas pasen de manera exitosa. Para este tipo de implementaciones se usan herramientas como Jenkins, Azure DevOps, entre otros.

IV. METODOLOGÍA

En este capítulo se detalla la metodología planteada para el desarrollo del proyecto de práctica. El desarrollo se dividió en ocho módulos, agrupados en cuatro fases (ver Figura 2). Cada módulo será explicado con detalle más adelante. La primera fase fue de introducción tanto del proyecto como de las diferentes herramientas que se necesitaban conocer para la elaboración del proyecto de automatización. La segunda fase, fue el primer acercamiento a la automatización de nuevas pruebas que no estaban en el código base, como preparación para realizar las pruebas de regresión. En la fase tres se hizo el análisis e implementación de los casos de prueba de regresión. Y por último, en la fase cuatro se realizó la investigación de cómo realizar un pipeline en Jenkins y su respectiva implementación.

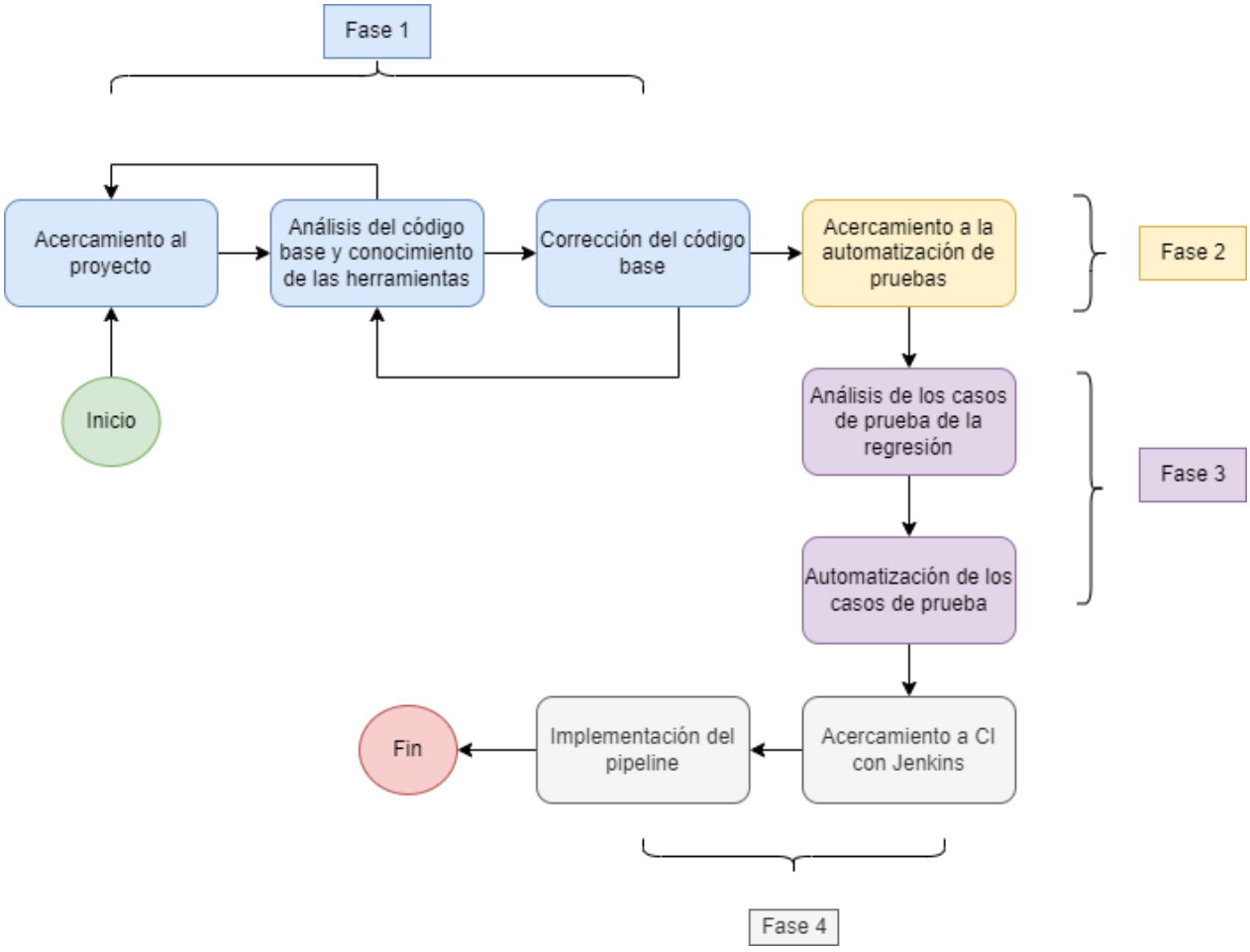


Figura 2. Metodología propuesta para el desarrollo del proyecto.

1. *Acercamiento al proyecto*

   Inicialmente se realizó un análisis sobre cuáles eran los objetivos del proyecto y el porqué era necesaria la inclusión de las pruebas automatizadas, todo esto acompañado por el asesor interno. En esta parte se hizo un acercamiento a las diferentes plataformas con las que contaba la empresa y las diferentes funcionalidades de cada una.

2. *Análisis del código base y conocimiento de las herramientas*

   El proyecto contaba con un código base realizado con Java, como lenguaje de programación, y Selenium WebDriver y TestNG para la implementación de pruebas de UI. Por lo tanto, fue necesario realizar una investigación sobre cómo se usaban estas herramientas para la automatización y, con esta información, se revisó el código base desde el punto de vista de qué arquitectura o modelos estaban siendo utilizados, qué pruebas estaban o no funcionando actualmente, con el objetivo de realizar las respectivas correcciones.

3. *Corrección del código base*

   En el código base, después del análisis, se encontró que varios de los casos de prueba ya no eran válidos, ya que al estos haberse realizado hacía mucho tiempo, ya no cumplían con las reglas actuales del proyecto. Así que lo primero que se realizó fue una investigación sobre los diferentes patrones de diseño que existen para ver cuál era el mejor para realizar esta práctica, junto con otras herramientas. Luego de esto, las mejores prácticas se aplicaron y se hizo una limpieza del código, eliminando los casos que ya no aplicaban para el proyecto actual, actualizando los métodos que allí se encontraban.

4. *Acercamiento a la automatización de pruebas*

   En este proceso se planteó por parte del asesor interno la realización de nuevos casos de prueba con el fin de adquirir más experiencia y conocimiento sobre la herramienta de automatización. Por lo que se empezó con una funcionalidad con la interacción de filtrar los elementos de una página. Luego de realizadas estas pruebas, se procedió a realizar unas con un nivel de complejidad más alto, ya que éstas contenían más interacciones, no

sólo en una misma página, sino entre diferentes páginas, por lo que había que realizar más pasos para la validación.

5. *Análisis de los casos de prueba de la regresión*

   Ya habiendo automatizado algunas pruebas del proyecto, se realizó un análisis de los diferentes casos de prueba que estaban catalogados como regresión en el test plan del equipo para la plataformas que se manejó en este proyecto (*plataformaC*). Este análisis constaba de observar cuáles casos de prueba eran candidatos a ser automatizados y el cómo se iban a implementar las pruebas en la herramienta de automatización.

6. *Automatización de los casos de prueba*

   Teniendo los casos de pruebas candidatos a automatizar identificados, se siguió con la implementación de estos a través de nuestro proyecto con Selenium y TestNG, siempre haciendo uso de los respectivos patrones y arquitecturas ya definidas en la aplicación base.

7. *Acercamiento a CI con Jenkins*

   En este acercamiento se tuvo que realizar una investigación sobre el uso de Jenkins y cómo lograr un pipeline que funcionara con un proyecto de automatización de pruebas con Selenium, TestNG y Maven.

8. *Implementación del pipeline*

   Ya con el conocimiento adquirido en el acercamiento, se procede a realizar las respectivas configuraciones del proyecto y la creación del pipeline de Jenkins. La validación se realizó mediante una corrida del pipeline en la que se mostraron los casos de prueba automatizados en el punto 6 y estos resultados se visualizaron de manera fácil y legible, con los respectivos datos que retornan las pruebas al momento de correrse.

V. RESULTADOS

La primera fase del proyecto constó de un análisis y corrección del código base. Esto se hizo aplicando el patrón de diseño Page Object Model (POM), que consta de dividir las clases que contienen la búsqueda e interacción que realiza Selenium sobre una página web en una carpeta (Pages), y las clases donde se ejecutan las pruebas (que llama a los métodos de las clases anteriormente mencionadas) se agrupan en otra carpeta (Tests). La base del proyecto ya tenía definido este patrón de diseño, pero todo estaba almacenado en una carpeta llamada *main*. De allí se desplegaban todas las demás carpetas, esto se cambió debido a que las pruebas deben ser almacenadas en *test* y no en *main*, por lo que se hizo la respectiva reestructuración.

Además, algunas funcionalidades o métodos se generalizaron en otras clases ya que eso ayudaba a que estos fueran consumidos con mayor facilidad, como es el caso de los waits. Para buscar un elemento en una página web e interactuar con él es necesario esperar a que este esté presente en la página o se haga algún tipo de espera en específico. Por lo que para funcionalidades de este tipo, que son tan generales, se creó una nueva clase que los contuviera.

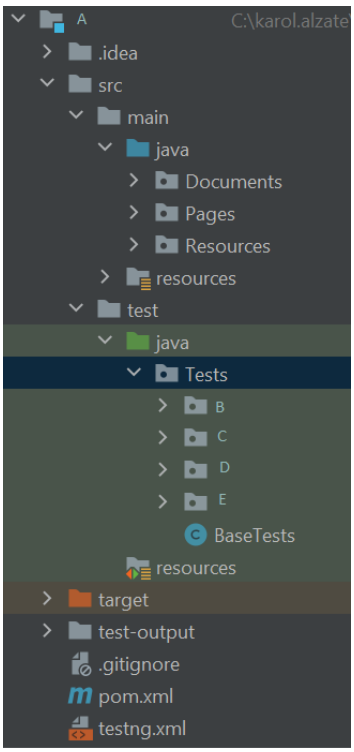


Figura 3. Estructura del proyecto

En la Figura 3 se puede apreciar la estructura final del proyecto con el modelo POM aplicado, con las clases divididas y agrupadas para cada caso. Este paso es importante porque al momento de correr las pruebas desde Maven, este buscará la carpeta *test* y ejecutará las pruebas allí almacenadas. Con la configuración anterior, no hubiera sido posible correr las pruebas con Maven.

Ahora, adentrándonos a lo que había al interior de las clases ya definidas al momento de ejecutar los scripts, las pruebas fallaron debido a que muchos de los métodos que se estaban implementando no funcionaban correctamente. Esto se debía a dos motivos, el primero es que la base se había realizado hacía mucho tiempo y se presentaba el caso de que los elementos de la página no se podían encontrar de la misma manera o habían sido reemplazados. El segundo motivo viene de que algunas de las funcionalidades principales del equipo cambiaron o se migraron a otras tecnologías, como fue el caso del login. El login era realizado anteriormente como una funcionalidad que era parte del proyecto, pero esto después cambió y se decidió usar una herramienta de autenticación externa. Esto hizo que las pruebas del login que había ya no funcionaran y se debieran reemplazar los métodos que allí se encontraban.

También se notó que no se estaba implementando una herramienta de reportes, por lo que esta también fue agregada. Es indispensable dicha implementación ya que, como se propuso que esto debía ser implementado por un pipeline, es una manera más fácil de mostrar la información de la corrida de las pruebas, en esta se incluye información sobre los tiempos que se demoran en correr los casos, cuáles están fallando, en qué punto, y su respectivo mensaje de error.

Después de realizada esta fase, se procedió a realizar un acercamiento a algunas funcionalidades de la página web de *plataformaC*. En este caso fue la validación de algunos filtros de la página principal. Luego de esta validación, se realizaron pruebas más avanzadas ya que incluían no sólo interacción con la página principal, sino también con otras otras subpáginas que alimentaban la información que se mostraba en la página principal. En estas pruebas se aplicaron acciones más complejas como cambios de páginas, actualización de datos, entre otros.

En este punto, ya se tenía un conocimiento más profundo de cómo funcionaba la página y qué funciones realizaban en general sus elementos. Por lo tanto, se realizó el análisis de los casos de prueba relacionados a la regresión. La página web de *plataformaC* contaba con 14 casos de prueba en el test plan de la regresión, de los cuales eran candidatos a automatizar 10 casos. La

forma de identificar de manera general cuáles de estos casos eran candidatos a automatizar era identificando aquellos que contenían interacciones que no eran posibles de realizar mediante el uso de Selenium, como por ejemplo, la asignación de roles que se hacía en una herramienta externa, visualizar si una imagen si había sido cargada de manera correcta, entre otras.

Durante la fase de automatización de las pruebas surgieron algunos retos respecto a algunos casos dado que, a pesar de que en su análisis inicial ya se habían descartado ciertos casos, uno más se tuvo que descartar ya que la acción que se realizaba en la página no era posible replicarla a través de la automatización. El desarrollo con el que contaba la página web realizaba una interacción que no permitía que Selenium interactuara con el elemento de la manera que el caso de prueba necesitaba. Por lo que este caso también se tuvo que descartar. Teniendo finalmente 9 casos de prueba automatizados.

Finalmente, con los casos automatizados se implementó el pipeline en Jenkins. Para este proceso se presentaron algunos retos ya que Jenkins se puede configurar de diferentes maneras, dependiendo de cómo esté realizado el proyecto. El primer reto de este proyecto era que el código no estaba alojado en ningún repositorio, ni debía ser alojado en alguno aún, debido a unas restricciones existentes. Por lo tanto, el pipeline se comenzó a realizar desde un enfoque local, generando todas las configuraciones dentro del Jenkins apuntando a una carpeta perteneciente al local de nuestro computador y no a un repositorio. Esto funcionó, pero no era la manera ideal ya que se esperaba que en el futuro el proyecto de automatización fuera agregado dentro de un repositorio. Por ende, para configurar el proyecto como si éste hiciera parte de un repositorio, se creó un proyecto nuevo en un repositorio privado con las configuraciones que contenía el proyecto original. Se cambiaron los nombres de todos los tests y se modificaron, ya que en sí, sólo se necesitaba que la configuración del proyecto fuera similar al proyecto de automatización principal, por lo que ningún caso de prueba fue replicado; se implementaron unos test de otra página para no revelar información del proyecto.

Ya con este proyecto de ejemplo almacenado en un repositorio, se le creó un Jenkinsfile que contuviera la configuración de los comandos y variables de entorno que se usaron en la configuración local para que este pudiera funcionar. Nuevamente se configura el pipeline, pero ya apuntando al repositorio en vez de al archivo local, y se procede a correr las pruebas. En la Figura 4 podemos observar como estas se corrieron con éxito.

```
[INFO] Tests run: 15, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 59.789 s - in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 15, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:11 min
[INFO] Finished at: 2023-07-14T01:37:00-05:00
[INFO] ------------------------------------------------------------------------
```

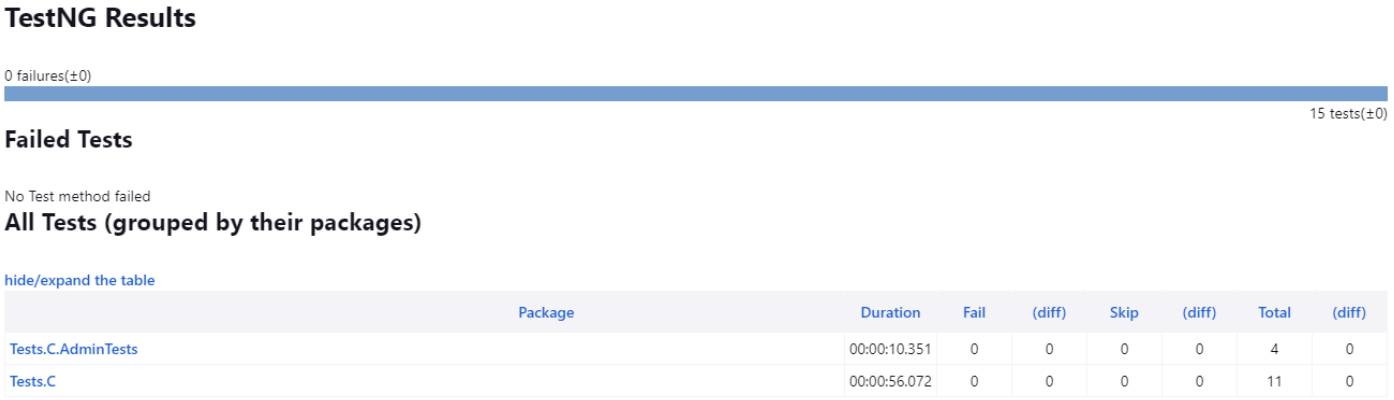
Figura 4. Resultados del pipeline

Anteriormente, se hizo bastante énfasis en los reportes de los resultados de los casos de prueba, por lo tanto, al pipeline también se agregaron herramientas que permitieran la visualización de dichos reportes: HTML Publisher y TestNG Results. En la Figura 5, se muestra el reporte HTML generado por Jenkins y la figura 6 el reporte generado por TestNG.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
Regression						
CTests	15	0	0	30,890		

Class	Method	Start	Time (ms)
Regression			
CTests — passed			
Tests.C.AdminTests.AdminPageTests	checkFilters	1689316617178	451
	checkFunctionality1Works	1689316617630	110
	checkFunctionality2Works	1689316617741	352
	checkPagination	1689316618093	351
Tests.C.DetailPageTests	checkDetailPage	1689316608112	722
Tests.C.HomePageTests	checkAddAndRemoveFromOpt1	1689316596211	1362
	checkAddAndRemoveFromOpt2	1689316597575	231
	checkCardsAreSortByOption1	1689316597807	184
	checkCardsAreSortByOption2	1689316597992	126
	checkCardsAreSortByOption3	1689316598118	140
	checkCardsAreSortBytOption4	1689316598258	141
	checkFiltersResultsAreCorrectlyFiltered	1689316598399	141
	checkHomePageElements	1689316598540	176
	checkNotFoundMessage	1689316598716	140
	checkSortByDefault	1689316596102	103

Figura 5. Resultado HTML del pipeline



TestNG Results

0 failures(±0)

15 tests(±0)

Failed Tests

No Test method failed

All Tests (grouped by their packages)

hide/expand the table

Package	Duration	Fail	(diff)	Skip	(diff)	Total	(diff)
Tests.C.AdminTests	00:00:10.351	0	0	0	0	4	0
Tests.C	00:00:56.072	0	0	0	0	11	0

Figura 6. Resultado TestNG del pipeline

Ambos reportes (Figura 5 y Figura 6) especifican el nombre de los casos de prueba, cuáles fallaron, cuáles pasaron y el tiempo que demoraron en correr, entre otro tipo de información. Este reporte muestra información más descriptiva que los resultados mostrados en la Figura 4 que sólo muestran la salida obtenida de la consola, igualmente de todos se obtiene que todos los casos de prueba pasaron. En los reportes se puede ver que el total de casos de prueba automatizados fueron 15, de estos 9 pertenecen a los casos de prueba de la regresión y 6 son los casos que se automatizaron en la fase 2, la de acercamiento a la automatización de pruebas.

VI. CONCLUSIONES

En este trabajo se implementó un sistema de automatización de pruebas que contiene las pruebas de regresión de la plataforma web *plataformaC* para la *empresaA*. Obteniendo un pipeline configurado en Jenkins capaz de correr las pruebas automatizadas implementadas. Todo esto se realizó con el fin de disminuir el tiempo de ejecución de las pruebas de regresión del equipo y también para brindar una base para la implementación de las demás pruebas presentes en el equipo, con una estructura correcta y con buenas prácticas, que permitan una mayor escalabilidad en el proyecto.

Para la realización de este sistema se hizo uso de Selenium WebDriver, TestNG, Maven, Java y Jenkins. Estas eran las herramientas que más se acomodaban a las necesidades del equipo dado que su código base se encontraba realizado en este lenguaje, y Jenkins ya estaba siendo implementado por el equipo para los desarrollos del proyecto en general. Con este último se pudo lograr no sólo la ejecución de las pruebas de manera exitosa, sino también la generación de los reportes. Éstos son importantes si se quiere hacer un seguimiento de las pruebas a futuro, su duración y si estas fallan constantemente, identificar las posibles razones por las que pueda estar pasando. Aparte de que estos se pueden descargar y ser enviados en caso tal que se quiera realizar una muestra de las pruebas.

La automatización de pruebas se puede aplicar a muchos tipos de pruebas que se adaptan a todo tipo de necesidades, es un campo que contiene muchas aplicaciones y cada vez se va extendiendo más entre más tecnologías surgen. Es una opción considerable para la disminución de tiempos y la detección temprana de errores, que a su vez ayudan a entregar un producto con más calidad. Por este motivo se recomienda la implementación de este proyecto de automatización no sólo para las pruebas de *plataformaC*, sino también para las demás plataformas existentes, extendiéndose no sólo a las pruebas de regresión sino también a aquellas que contengan una prioridad alta en el equipo. Como también el uso de otro tipo de pruebas, como las pruebas de performance o de carga.

REFERENCIAS

[1] "IEEE Standard for Software and System Test Documentation," *IEEE Std 829-2008*, pp. 1–150, Jul. 2008.

[2] K. Raiküla, "Implementation of automated end-to-end testing in web applications," 2023. https://www.theseus.fi/bitstream/handle/10024/794423/Raikula_Karina.pdf?sequence=2 (accessed Jul. 14, 2023).

[3] K. M. Mustafa, R. E. Al-Qutaish, and M. I. Muhairat, "Classification of Software Testing Tools Based on the Software Testing Methods," in *2009 Second International Conference on Computer and Electrical Engineering*, Dec. 2009, vol. 1, pp. 229–233.

[4] G. Duggal and B. Suri, "Understanding regression testing techniques," in *Proceedings of 2nd National Conference on Challenges and Opportunities in Information Technology*, 2008. [Online]. Available: https://www.researchgate.net/profile/Bharti-Suri/publication/228943618_Understanding_Regression_Testing_Techniques/links/5580f92f08aea3d7096e5842/Understanding-Regression-Testing-Techniques.pdf

[5] J. Pan, "Software testing," *Dependable Embedded Systems*, vol. 5, no. 2006, p. 1, 1999.

[6] M. Sharma and R. Angmo, "Web based automation testing and tools," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 908–912, 2014.

[7] D. Cherepovskiy and P. Drobintsev, "An approach to automating software product quality assurance using templates for all levels of the testing pyramid." http://83.149.199.66/2021/submissions/SYRCoSE_2021_paper_03_6.pdf (accessed Jul. 15, 2023).

[8] M. Fowler, "Continuous Integration," *martinfowler.com*. https://martinfowler.com/articles/continuousIntegration.html (accessed Jul. 15, 2023).