



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE APIZACO

IMPLEMENTACIÓN DE PRÁCTICAS DevOPS EN LA ORGANIZACIÓN SEGURIDATA.

TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

PARA OBTENER EL TÍTULO PROFESIONAL DE:

ING. EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES.

PRESENTA:

JAQUELINE CUAHTEPITZI BASTIDA

NOMBRE DEL ASESOR:

PEDRO CUAUTLE TONIX



APIZACO, TLAX., OCTUBRE 2023.



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Apizaco
Sistemas y Computación

Apizaco, Tlaxcala, 31 de mayo de 2023.

OFICIO No: SC/OPV/140/23

ASUNTO: Liberación de proyecto
para titulación integral

MARTÍN ROJAS RAMÍREZ
JEFE DEL DEPTO. DE DIVISIÓN DE ESTUDIOS PROFESIONALES
P R E S E N T E.

POR ESTE MEDIO LE INFORMO QUE HA SIDO LIBERADO EL SIGUIENTE PROYECTO PARA LA TITULACIÓN INTEGRAL.

NOMBRE DEL EGRESADO:	JAQUELINE CUAHTEPITZI BASTIDA
CARRERA:	INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES
NO. DE CONTROL:	15371145
NOMBRE DEL PROYECTO:	IMPLEMENTACIÓN DE PRACTICAS DevOPS EN LA ORGANIZACIÓN SEGURIDATA
PRODUCTO U OPCIÓN:	TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL

AGRADEZCO DE ANTEMANO SU VALIOSO APOYO EN ESTA IMPORTANTE ACTIVIDAD PARA LA FORMACIÓN PROFESIONAL DE NUESTROS EGRESADOS.

ATENTAMENTE

Excelencia en Educación Tecnológica
Pensar para Servir, Servir para Triunfar

ALAN AUGUSTO GALLEGOS CUELLAR
JEFE DEL DEPTO. DE SISTEMAS Y COMPUTACIÓN



SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL
INSTITUTO TECNOLÓGICO DE APIZACO
DEPARTAMENTO DE
SISTEMAS Y COMPUTACIÓN

PEDRO CUAHUTLE TONIX

ASESOR INTERNO

c.c.p. División de Estudios Profesionales
c.c.p. Depto. de Servicios Escolares
c.c.p. Archivo
AAGC/mrol*



Carretera Apizaco-Tzompantepec, esquina con Av. Instituto Tecnológico S/N,
Conurbado Apizaco-Tzompantepec, Tlaxcala, Mex. C.P. 90491 Tel. (241) 4172010 Ext. 115,
e-mail: sistemas@apizaco.tecnm.mx | apizaco.tecnm.mx



2022 Flores
Ricardo
Magón
Año de la Revolución Mexicana



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

Instituto Tecnológico de Apizaco
División de Estudios Profesionales

Apizaco, Tlax., **28/junio/2023**
No. OFICIO: DEP-CT/502/2023

ASUNTO: AUTORIZACIÓN DE PUBLICACIÓN

JAQUELINE CUAHTEPITZI BASTIDA
EGRESADA DE LA CARRERA DE
INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES
CON NÚMERO DE CONTROL: 15371145
P R E S E N T E

Por este medio, me permito informar a usted que por aprobación de la comisión revisora asignada para valorar el trabajo que mediante la opción **TITULACIÓN INTEGRAL INFORME DE RESIDENCIA PROFESIONAL** cuyo tema es: **"IMPLEMENTACIÓN DE PRÁCTICAS DevOPS EN LA ORGANIZACIÓN SEGURIDATA"**, conforme a lo establecido en el procedimiento para la obtención del Título Profesional en el Instituto Tecnológico, la División de Estudios Profesionales a mi cargo le otorga:

AUTORIZACIÓN DE IMPRESIÓN

Sin otro particular por el momento, le envió un cordial saludo.

ATENTAMENTE

Excelencia en Educación Tecnológica®
Pensar para Servir, Servir para Triunfar®

MARTÍN ROJAS RAMÍREZ
JEFE DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES



c.p. archivo



Av. Instituto Tecnológico No. 418, San Andrés Ahuashuatepec, Municipio de Tzompantepec,
Tlaxcala, Mex. C.P. 90491 Tel. (241) 4172010 Ext. 304,
e-mail: dep_titulación@apizaco.tecnm.mx



2023
Francisco
VILLA

Agradecimientos.

A la empresa SeguriData, por la confianza que han depositado en mí. Agradezco la oportunidad que me ha brindado por hacerme parte de su equipo de trabajo. Este apoyo y confianza ha sido de gran valor para mí. Espero aportar valor a sus proyectos y metas, para seguir siendo una gran organización. Hoy, gracias a esta organización cuento con las herramientas y estrategias necesarias para capacitarme y ser una mejor profesionalista.

Jaqueline

Resumen.

El presente trabajo de Residencias Profesionales expone la implementación de prácticas DevOps en la organización SeguriData Privada S.A DE C.V.

El objetivo de este proyecto es el de realizar un análisis de la situación actual de la organización, investigar acerca de las prácticas DevOps en una empresa de software, pasando por las diferentes áreas en las que incide, así como la investigación de herramientas de software para la automatización de pruebas para el área de QA de la organización.

Por otra parte, se implementará en la organización la adopción de este tipo de prácticas, concretamente se mostrará como DevOps mejora la gestión de incidentes que se lleva a cabo en el ámbito de QA mediante la herramienta Selenium.

ÍNDICE

GENERALIDADES DEL PROYECTO	1
1.1 Introducción.	1
1.2 Descripción de la empresa u organización y del puesto o área del trabajo del estudiante.	2
1.3 Problemas a resolver	4
1.4 Objetivos	5
1.5 Justificación.	6
MARCO TEÓRICO	7
DESARROLLO DEL PROYECTO	9
2.1 Procedimiento y descripción de las actividades realizadas.	9
2.1.1 Situación Actual de la organización.....	9
2.1.2 Practicas DevOps en una empresa de desarrollo de software.....	10
2.1.3 Integración de equipos	11
2.1.4 Proyecto piloto.....	13
2.1.5 Implementación DevOps.	13
2.1.6 Investigación de Herramientas de uso comercial y de uso libre para probar el software.	17
2.1.7 Implementación de herramientas DevOps para la automatización de pruebas.	24
2.1.8 Implementación de Selenium para pruebas de software a Producto Vertical SeguriSign.....	26
2.1.9 De la evaluación de los productos, se toma la Decisión.....	27
RESULTADOS	35
3.1. Análisis de la aplicación.	35
3.1.1 Selenium Web Driver.....	35
3.1.2 Selenium IDE	35
3.1.3 TestNG	36
3.2 Preparación de la infraestructura.	37
3.2.1 Instalación de Java.....	37
3.2.2 Instalación del IDE Eclipse.	38
3.2.3 Creación de proyecto Maven.....	42
3.2.4 Instalación de Dependencias en Maven.....	45

3.2.5 Como ejecutar pruebas en Selenium en el Navegador Chrome	50
3.2.6 Metodología de Programación con la que se trabajará	52
3.2.7 Patrón de diseño: Page object models (POM)	54
3.3 Creación de casos de prueba	55
3.3.1 DriverSetup.java	55
3.3.2 ClasePage.java	57
3.3.3 EscenariosPruebas.java	68
3.3.4 CasosPrueba.class	69
3.4 Complementación de las pruebas.....	73
3.5 Ejecución y Depuración.	73
3.5.1 Resultados de la prueba.....	74
CONCLUSIONES	76
4.1 Conclusión	76
4.2 Recomendación	76
COMPETENCIAS DESARROLLADAS Y/O APLICADAS	77
FUENTES DE INFORMACIÓN.	78

GENERALIDADES DEL PROYECTO

1.1 Introducción.

En los últimos años, se han realizado cambios en el desarrollo de software de sistemas de información hacia una forma más ágil de trabajar. Los equipos ágiles se responsabilizan de la consecución de requerimientos en un conjunto multidisciplinar, donde la parte de negocio y la parte de desarrollo aparecen representadas en un solo proyecto. DevOps es un nuevo movimiento que trata de mejorar la agilidad en la prestación de servicios. Fomenta una mayor colaboración y comunicación entre los equipos de Desarrollo, Operaciones y Calidad.

En el ámbito de las tecnologías de la información, se requiere una respuesta cada vez más rápida a las necesidades de los clientes y a la vez un servicio que sea estable, seguro y predecible. Esto establece un compromiso entre estabilidad y cambio. Trabajar en una solución que entregue cambios de forma frecuente (como pueden ser las metodologías ágiles) sin tener en cuenta la fiabilidad en la gestión operacional o la comunicación entre Desarrollo, Operaciones y Calidad hace que en muchas ocasiones esa solución no sea suficiente.

En este contexto, el movimiento DevOps plantea una visión integrada en el desarrollo del software que permite implementar cambios de manera rápida y segura, garantizando la fiabilidad en las operaciones (puesta en marcha del software, monitorización y análisis de incidencias).

1.2 Descripción de la empresa u organización y del puesto o área del trabajo del estudiante.

- Empresa

SeguriData Privada S.A de C.V es una empresa mexicana constituida en 1996 dedicada a Integrar soluciones de valor que permite implementar soluciones integrales de Seguridad de la Información, obteniendo mayor seguridad y beneficios para las organizaciones.

La empresa ha fortalecido sus soluciones con socios estratégicos para ofrecer las herramientas que permitirán modernizar y hacer más eficientes los procesos en cumplimiento con los objetivos de la institución sea del gobierno o de la iniciativa privada.

Con sus procesos de innovación y continua mejora, dentro de su Plan estratégico para el año 2020, está la Implementación y adopción de DevOps, involucrando a las áreas de Desarrollo, QA y Operaciones, con el objeto de dar respuestas ágiles a los clientes tanto internos como externos.

- Puesto o área del trabajo el estudiante.

Quality Assurance (QA). Aseguramiento de la calidad.

En el área de control de calidad (QA), se maneja una metodología de Pruebas para la organización llamada Proceso T, la cual define los pasos a seguir, los cuales coinciden con el proceso definido para las Pruebas de calidad de los productos y servicios, los cuales involucran:

- Planes de prueba
- Matrices de prueba:
 - a) Escenarios de prueba
 - b) Casos de uso
 - c) Casos de prueba
- Pruebas de funcionalidad
- Pruebas de usabilidad

- Pruebas de Seguridad

Las pruebas de calidad son ejecutadas en conjunto con el área de desarrollo con el objetivo de producir software de alta calidad, y entregar a los clientes en tiempo y forma con eficacia y eficiencia.

El puesto dentro del área de Control de calidad que ejecuta y diseña las pruebas, cumpliendo con los puntos anteriores es el de Tester, cuyas funciones actuales son:

1. Revisión y entendimiento de requerimientos de usuario
2. Creación de matrices de prueba
3. Analizar, crear y ejecutar casos de prueba funcionales
4. Identificar insumos requeridos para la ejecución de las pruebas
5. Elaboración de documentos de evidencia de pruebas
6. Reportar y hacer seguimiento a los defectos encontrados
7. Reportar estatus de actividades

1.3 Problemas a resolver

Actualmente el desarrollo de aplicaciones se da con metodologías tradicionales de cascada, donde las funciones de las áreas de desarrollo, QA y operaciones se dan de manera separada, lo cual dentro de la organización ha generado una serie de problemáticas, las cuales se enuncian a continuación:

- Incumplimiento de fecha compromiso a clientes

Como consecuencia de que las pruebas y entregas se realizan al finalizar el desarrollo, no se tienen periodos de pruebas cíclicos y los errores son detectados al final previo a la entrega comprometida al cliente, esto hace que los tiempos no se cumplan y exista molestia por parte de los clientes.

- Productos y servicios con baja calidad

Como consecuencia de que las pruebas y entregas se realizan al finalizar el desarrollo, no se tienen periodos de pruebas cíclicos, por lo que en ocasiones los productos y servicios se entregan a los clientes con problemas de usabilidad, que se dejan pendientes para futuras versiones.

- Desarrollos incompletos

En ocasiones los desarrollos no contemplan la totalidad de requerimientos solicitados.

- Entregas con problemas para clientes

Como consecuencia de que las pruebas y entregas se realizan al finalizar el desarrollo, no existe tiempo para que quien instala al cliente esté capacitado y conozca la solución, esto ocasiona problemas en las entregas.

- Comunicación interna deficiente

La presión en las entregas y los tiempos cortos, ocasionan roces entre los integrantes de las áreas, aunado a que la comunicación es ineficiente y poco clara.

- Problemas entre recursos de las diferentes áreas

Cuando se dan problemas en las entregas a los clientes, se buscan culpables, esto ocasiona un ambiente tenso y genera problemas entre recursos.

1.4 Objetivos

Objetivo General

Implementar herramientas y prácticas DevOps para agilizar y automatizar los procesos entre las áreas de Desarrollo de software, QA y Operaciones dentro de la organización SeguriData, garantizando procesos rápidos, fiables y de mayor calidad.

Objetivos específicos

- Realizar investigación y análisis de las prácticas de DevOps.
- Estudiar la situación actual de la organización para elegir un camino de transición en donde las áreas involucradas vayan adaptando estas prácticas de la forma más adecuada en relación a sus necesidades.
- Análisis, diseño, desarrollo e implementación de un Sistema para realizar pruebas automáticas para el área de Aseguramiento de la calidad (QA).
- Incluir dentro de las prácticas de DevOps la metodología SCRUM.
- Implementar las prácticas de DevOps dentro de la organización mediante un proyecto piloto de alguno de los productos con los que actualmente cuenta la organización.

1.5 Justificación.

Actualmente la empresa no cuenta con prácticas ágiles a lo largo del ciclo de vida de desarrollo de software, haciendo que los tiempos no sean los mejores y se presenten problemas a lo largo del camino.

Hoy día la empresa ha tomado la decisión de realizar un cambio sobre sus metodologías, buscando implementar prácticas ágiles de la mano de DevOps. Se propone entonces realizar una investigación de las prácticas DevOps para realizar la implementación de la misma dentro de la organización, aportando de esta manera las herramientas y pasos necesarios para el cambio y ayudar a la organización en esta transición a un resultado favorable.

Dentro de la implementación de DevOps interviene toda la organización, de manera principal e inmediata se involucran las áreas de: Desarrollo, QA y Operaciones, con apoyo de la Dirección General.

En el área de QA las primeras actividades a realizar es la automatización del proceso de pruebas, realizando investigación e implementación de herramientas, así como la capacitación y participación en la metodología Scrum.

MARCO TEÓRICO

Las empresas de hoy necesitan responder a las necesidades del cliente a velocidades sin precedentes [1]. Impulsadas por esta necesidad de velocidad, muchas compañías se apresuran al movimiento DevOps e implementan la Entrega Continua (CD).

Uno de los objetivos principales de cualquier prueba de software, además de garantizar la calidad del producto y/o servicio, es reducir costos, tiempo y esfuerzos que toma realizar cada prueba, y el tiempo que se consume al reportar los errores hallados, al área de Desarrollo para su corrección, y esto sucede justo cuando esta por realizarse la entrega a los clientes.

Uno de los mayores retos es cuando se requiere realizar pruebas cuando el sistema ya está en producción, y se reporta algún incidente que afecta la producción de un cliente.

El marco teórico de referencia para la implementación de DevOps, y en específico la automatización de pruebas y agilidad, son: Scrum y las metodologías Ágiles.

El principio de equipos auto-organizados y auto-gestionados que propone la metodología Scrum para el desarrollo de software podría ser extrapolado a toda una organización, buscando crear una dinámica en la que sus equipos de operaciones, tecnología, ventas, mercadeo, finanzas y legal se articulen de forma autónoma para colaborar en la definición de los requerimientos del cliente/usuario en perfecta sincronía con las personas distribuidas globalmente tanto a nivel interno –la empresa– como externo –los clientes– (Davidson y Klemme, 2016).

Bajo una ideología Agile, el trabajo se debe enfocar en cumplir con las expectativas de los clientes (Denning, 2018), brindando una mayor agilidad a toda la organización, permitiendo una mutación en las formas, las composiciones de los equipos, el auto reclutamiento y la autonomía para definir los negocios, los mercados y los nichos donde se quiere competir, buscándoles un liderazgo con productos ganadores que satisfagan las necesidades de los clientes. En Scrum, los equipos son conformados por alrededor de nueve integrantes que no tienen una jerarquía marcada; a cada integrante del equipo se le asigna el mismo grado de responsabilidad y cada uno de ellos confía en el otro.

Para permitir estas libertades y asignaciones de confianza que se le asignan a cada colaborador es importante mantener los siguientes tres niveles de planeación usados en Scrum:

- 1) Reuniones diarias de 15 minutos. Cada integrante comparte con sus pares las actividades del día anterior, las que va a realizar en el día y los impedimentos que cree tener para lograrlas. Todo el equipo está de pie.
- 2) Planeaciones de la iteración. Se hacen cada 15 días para establecer los objetivos grupales por cumplir en las dos semanas siguientes.
- 3) Planeaciones de lanzamiento de versión. Se hacen en períodos de tres a nueve meses para definir la visión del producto en el mediano plazo y los hitos que se deben cumplir

A partir de la literatura existente y de los principios del Scrum se han diseñado algunos marcos de referencia de trabajo (reference frameworks) para implementar estas metodologías en las empresas según su tamaño, y permitir que se vayan generando mutaciones en la medida que vayan escalando, ya que a mayor tamaño mayores son los retos de alineación y sincronización entre las personas para mantener la agilidad e innovación.

Los dos marcos de referencia de trabajo más conocidos son SAFe y LeSS framework. En consideración al hecho de que SAFe tiene mayor documentación y goza de una mayor adopción y reconocimiento en la industria, se ha tomado como marco de referencia para esta investigación.

Con este marco se estructurará la implementación de DevOPS en la organización SeguriData.

DESARROLLO DEL PROYECTO

2.1 Procedimiento y descripción de las actividades realizadas.

2.1.1 Situación Actual de la organización

Las responsabilidades del Área de QA, se enfocan en el ciclo de pruebas de un producto, de manera tradicional, donde se realiza lo siguiente:

- Análisis de requerimiento.

Se realiza el análisis a un requerimiento establecido, donde se comprende el funcionamiento del producto y en base a ello se definen las etapas posteriores.

- Elaboración del Plan de pruebas.

El plan de pruebas se realiza en base al análisis del requerimiento, ya que con ello se define lo siguiente:

- Ambiente de pruebas.
- Matriz de pruebas.
- Tiempos de ejecución de pruebas.
- Herramientas de apoyo para pruebas.

- Diseño de escenarios de Pruebas.

Con el análisis del requerimiento, se parte para la creación de la matriz de prueba, donde se ingresan todas las posibles casuísticas del producto para localizar incidencias o verificar el correcto funcionamiento del producto.

- Ejecución de Pruebas.

La ejecución de la matriz de prueba sobre el producto se realiza con el objetivo de validar lo mencionado en el requerimiento, con el fin de liberar un producto con estándares de control de calidad validados.

- Liberación de producto.

Una vez que se hayan realizado las pruebas y verificado que el producto no contiene defectos, se libera al equipo de operaciones para que se implemente a nivel productivo o se entregue al cliente en curso.

Esta manera tradicional de trabajo no es funcional, dadas las necesidades cambiantes de los clientes, por lo que surge la idea de un cambio, de manera que:

El proceso de Control de Calidad, deberá participar desde el nacimiento de una idea hasta la implementación del software, apoyando en todo el ciclo de vida del producto, ya que el control de calidad no debe limitarse solamente a la ejecución de Pruebas.

En función de las mejores prácticas para el Control de Calidad se define lo siguiente: Uso de DevOps como conjunto de prácticas de desarrollo de software que combinan la iteración de diferentes áreas, ya que fomenta la colaboración, comunicación e integración con el fin de interactuar de la mano, para la liberación de productos de forma ágil, con estándares de calidad elevados.

2.1.2 Practicas DevOps en una empresa de desarrollo de software.

DevOps es un conjunto de prácticas que enfatizan la colaboración, comunicación e integración entre los desarrolladores de software y los profesionales de operaciones de TI, automatizando el proceso de entrega de software y los cambios de infraestructura. El nombre DevOps nace de la unión de “Desarrollo” (Dev) y “Operaciones (Ops) “.

DevOps tiene como objetivo crear una cultura y un entorno en el que el diseño, las pruebas y el lanzamiento de software se puedan realizar de forma rápida, frecuente y eficiente, de mejor calidad y a un coste menor. DevOps no es solo una metodología, es una verdadera filosofía de trabajo.

La cultura DevOps se ha convertido en el camino que permite optimizar el rendimiento de un proyecto en un contexto en el que los cambios se producen a un ritmo más rápido que los propios desarrollos.

DevOps nos empuja hacia un único código de referencia que incluye software, middleware y entorno operativo. DevOps reúne aquellos que históricamente fueron sistemas diferentes, en términos de modo de gestión, ciclo de lanzamiento o modelo de control.

La propuesta de implementación de dichas prácticas va dirigida al área de Ingeniería y Desarrollo, así como a los miembros del área de Control de Calidad, y Operaciones, donde se definirá de forma práctica mediante lo siguiente:

- Integración de equipos.
- Actividades.
- Organigrama.
- Proyecto Piloto.
- Implementación de DevOps.

Con DevOps, más disciplinas trabajan juntas, en un modelo de entrega perfectamente integrado, para agregar valor al cliente.

2.1.3 Integración de equipos

Al adoptar una cultura de DevOps junto con prácticas y herramientas de DevOps, los equipos adquieren la capacidad de responder mejor a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y alcanzar los objetivos empresariales en menos tiempo.

Los equipos que adoptan la cultura, las prácticas y las herramientas de DevOps mejoran el rendimiento y crean productos de más calidad en menos tiempo, lo que aumenta la satisfacción de los clientes.

Aunque la adopción de prácticas de DevOps automatiza y optimiza los procesos con tecnología, todo comienza con la cultura interna de la organización y con las personas que participan en ella. El desafío de cultivar una cultura de DevOps requiere cambios profundos en la forma en la que las personas trabajan y colaboran. Pero cuando las organizaciones se comprometen a implementar una cultura de DevOps, pueden crear un entorno que facilite el desarrollo de equipos de alto rendimiento.

La integración de equipos respecto al Área de Ingeniería y Desarrollo y Control de Calidad, será con base en la asignación de productos, ya que en el momento en que exista la necesidad de liberación de un producto, se integre un equipo formado por un desarrollador y un ingeniero de pruebas, de forma que se planifique el ciclo de pruebas de cada producto en el que interactúen.

El objetivo es realizar un canal de comunicación y colaboración que permita la liberación de productos con estándares de calidad elevados.

Algunos puntos importantes a tomar en cuenta en la implementación de la cultura de DevOps:

COLABORACIÓN, VISIBILIDAD Y ALINEAMIENTO: Una buena cultura de DevOps se distingue, entre otras cosas, por la colaboración entre los equipos, que comienza con la visibilidad. Diferentes equipos, como el de desarrollo o el de operaciones de TI, deben compartir entre sí sus procesos de DevOps, sus prioridades y sus preocupaciones. Estos equipos también deben planear juntos el trabajo y alinear sus objetivos y los indicadores del éxito en relación con el negocio.

CAMBIOS EN EL ÁMBITO Y EN LA RESPONSABILIDAD: A medida que los equipos se alinean, asumen y participan en más fases del ciclo de vida, no solo las que son principales para su rol. Por ejemplo, los desarrolladores asumen responsabilidad no solo por la innovación y la calidad establecidas en la fase de desarrollo, sino también por el rendimiento y la estabilidad que sus cambios producen en la fase de uso. Al mismo tiempo, los operadores de TI se aseguran de incluir la gobernanza, la seguridad y el cumplimiento normativo en las fases de planeamiento y desarrollo.

CICLOS DE LANZAMIENTO DE VERSIONES MÁS CORTOS: Los equipos de DevOps mantienen la agilidad porque lanzan versiones de software en ciclos cortos. Los ciclos de lanzamiento de versiones más cortos facilitan el planeamiento y la administración de los riesgos, porque el progreso es incremental, lo que reduce el impacto en la estabilidad del sistema. El acortamiento de los ciclos de lanzamiento de versiones permite también a las organizaciones adaptarse y reaccionar a las necesidades cambiantes de los clientes y a la presión competitiva.

APRENDIZAJE CONTINUO: Los equipos de DevOps de alto rendimiento establecen una mentalidad de crecimiento. Aceptan el fracaso y responden rápido a los errores, e incorporan lo que aprenden a los procesos, de modo que mejoran continuamente, aumentan la satisfacción del cliente y agilizan la innovación y la capacidad de adaptación al mercado de forma constante. DevOps es un recorrido, por lo que siempre hay espacio para crecer.

2.1.4 Proyecto piloto.

La implementación de DevOps al proceso de Control de Calidad, se podrá demostrar o ejemplificar con la integración a un proyecto piloto, definiendo cada una de las mejoras que se desea mostrar con la integración de las nuevas prácticas.

2.1.5 Implementación DevOps.

Un aspecto importante y crítico para DevOps, es la automatización de todo el proceso, desde el despliegue hasta la validación del producto.

De modo que las pruebas automatizadas ya no son una opción, sino una necesidad latente para la empresa, de la mano del uso de Scrum.

La implementación de DevOps en la organización, permite desarrollar y probar softwares con mayor eficiencia, rapidez y calidad. Por ello se pretende realizar una automatización de las pruebas en DevOps, para optimizar los procedimientos desde la raíz.

Sin automatización no hay DevOps.

La automatización incluirá desde los desarrollos de software, así como las pruebas de control de calidad ya que estas permitirán localizar lo antes posible errores de código.

La implementación de automatización de pruebas definirá una mejor calidad de código, optimización en los tiempos de entrega y organización en ambientes de pruebas.

La automatización se podrá realizar principalmente para lo siguiente:

- Automatización de pruebas funcionales.
- Automatización de pruebas de regresión
- Automatización de pruebas de rendimiento.

En definitiva, la automatización es uno de los pilares fundamentales en los que se basan los principios de DevOps. Invertir en la automatización de los procesos, liberando a los equipos de tareas que no aportan valor y que les permitan orientar sus esfuerzos hacia la mejora del producto final, sería un buen

comienzo para las organizaciones que tienen como objetivo llegar a ser DevOps.

Pruebas manuales vs pruebas automáticas

Pruebas de software

Las pruebas de software o testing son un elemento fundamental en cualquier proyecto. La realización de pruebas de software es una parte esencial y fundamental para cualquier sistema, ya que garantiza añadir cierto grado de calidad al sistema antes de ser liberado para los usuarios finales.

De modo que establecer una metodología estratégica para llevarlas a cabo es algo que nunca debe pasarse por alto. Solamente de esta forma nos acercaremos a la excelencia de los estándares de calidad que el cliente necesita; en forma de resultados objetivos. El departamento de QA de la empresa Seguridata tiene definida una metodología, Proceso T.

Cuando se habla de pruebas en términos generales, los diferentes profesionales de este campo, tanto los analistas QA (Quality Assurance) como los testers propiamente dichos, hacen referencia a dos enfoques principales, y en numerosas ocasiones, existen fronteras difusas, fricciones, dudas y diferentes formas de interpretar los resultados.

■ Pruebas Manuales

Las ventajas que ofrecen las pruebas manuales vs las pruebas automatizadas suelen compararse constantemente en la búsqueda por la eficiencia y la calidad del desarrollo software. Pero ambas pueden (y deben) convivir para lograr resultados eficaces tanto en la detección de incidencias como en la prevención.

Las pruebas manuales, son ejecutadas directamente por uno o más testers, simulando las acciones del usuario final, apoyándose de las herramientas necesarias.

Las pruebas son imprescindibles, ya que nos permiten garantizar que las aplicaciones cumplen las funcionalidades que se esperan de ellas y las expectativas de calidad (no solo de código); ayudando a encontrar esos errores o defectos que aún no se han descubierto; reduciendo el costo del desarrollo,

el de propiedad para los usuarios; y desarrollar confianza en los clientes al evitar los molestos errores de regresión.

- Ventajas de pruebas manuales

Las pruebas manuales son indispensables para cada proyecto, ya que por medio de ellas se detectan la mayor parte de los defectos, y de esta forma se logra estabilizar el sistema que se está probando.

También permiten un análisis profundo por parte de un humano, lo cual es un mayor beneficio cuando se quiere mejorar la experiencia de usuario.

- Desventajas de pruebas manuales

Las repeticiones de pruebas manuales podrían dejar pasar defectos, ya que una acción repetitiva para una persona es cansada y tediosa.

Pueden llegar a consumir mayor tiempo que las automatizadas. Es necesario incluir pruebas de regresión cuando se está haciendo una actualización o ampliación de la funcionalidad del sistema.

- Pruebas automatizadas

Las pruebas automatizadas tienen como objetivo detectar fallas en el software evitando que se tenga que ejecutar las pruebas manualmente.

Las pruebas automatizadas son ejecutadas por testers con skills técnicos, y se apoyan de diversas herramientas para realizar scripts y así, ejecutar las pruebas automáticamente.

Es decir, se hará un listado de los casos de prueba a probar, utilizando alguna herramienta de automatización, para que luego, estos casos de prueba se reproduzcan automáticamente.

Este tipo de pruebas, brindan un mayor grado de confiabilidad cuando se aplican después de algún cambio al hardware o software donde se ejecuta el sistema, además, simulan la interacción humana con el software.

La automatización debe cimentarse en un proceso de pruebas manuales robusto.

- Ventajas de pruebas automatizadas

Las pruebas automatizadas son la opción más viable cuando se necesita realizar diversos test cases, de manera repetitivas y por un período de tiempo extenso. Pueden ejecutarse una y otra vez luego de ser creadas, más rápido que las pruebas manuales, idóneas para cambios en funciones nuevas, para garantizar que lo que no se ha modificado siga funcionando.

- Desventajas de pruebas automatizadas

Deben aplicarse hasta que un sistema se encuentre estable, es decir que haya pasado por el proceso de pruebas manuales

Capacidad de ejecución de pruebas

Es posible realizar un gran número de pruebas en un breve período de tiempo. Se pueden ejecutar en forma paralela y sobre distintos servidores, aumentando aún más su capacidad.

Adicional a las ventajas y desventajas, debemos considerar que las pruebas manuales no pueden desaparecer por completo pues hay temas a revisar que la automatización no resuelve, estos son de tipo visual como los colores, los tamaños, en general las precisiones que, hasta ahora, solo el ojo humano puede detectar.

2.1.6 Investigación de Herramientas de uso comercial y de uso libre para probar el software.

El objetivo de esta actividad es encontrar soluciones a las necesidades que no pueden satisfacerse con las pruebas manuales que actualmente se realiza en el área de QA en la empresa SeguriData Privada SA. DE C.V.

Esta actividad incluye recorrer foros, referencias y buscar antecedentes similares a la necesidad planteada. Luego se analizan las posibles soluciones. Esta actividad también incluye búsqueda de nuevas herramientas que asistan en la automatización.

Los equipos tienen muchas herramientas de DevOps para facilitar la adopción de una cultura de DevOps en su organización. La mayoría de los equipos confían en varias herramientas y, de este modo, crean cadenas de herramientas personalizadas que han adaptado a sus necesidades en cada fase del ciclo de vida de las aplicaciones. Si bien adoptar una herramienta o una tecnología específicas no es lo mismo que adoptar DevOps, cuando la cultura de DevOps está presente y los procesos están definidos, la gente puede implementar y optimizar las prácticas de DevOps si eligen las herramientas adecuadas.

Herramientas para automatización de pruebas

Existen muchas herramientas de DevOps para facilitar la adopción de la cultura en la organización. La mayoría de los equipos confían en varias herramientas y, de este modo, crean cadenas de herramientas personalizadas que han adaptado a sus necesidades en cada fase del ciclo de vida de las aplicaciones.

Si bien adoptar una herramienta o una tecnología específicas no es lo mismo que adoptar DevOps, cuando la cultura de DevOps está presente y los procesos están definidos, la gente puede implementar y optimizar las prácticas de DevOps si eligen las herramientas adecuadas.

Se necesita utilizar herramientas para mejorar la calidad, aumentando la productividad y la eficacia a la hora de realizar las pruebas.

Ahora bien, habiendo distintos tipos de Testing nos enfrentamos con distintos desafíos al tener que utilizar herramientas específicas que atiendan a los mismos.

Poder seleccionar las herramientas que respalden los procesos, es fundamental para el éxito de DevOps. Si las operaciones intentarán mantener el ritmo de los ciclos de desarrollo rápidos, necesitarán usar plataformas altamente flexibles y tratar su infraestructura como los equipos de desarrollo tratan el código.

Las implementaciones manuales son lentas y dan lugar a errores. El aprovisionamiento y la implementación de plataformas se pueden simplificar mediante la automatización.

Para poder elegir la mejor herramienta que cumpla con nuestras necesidades, se tomaron en cuenta varios factores, entre ellos, identificar puntualmente los procesos definidos para el área de QA. Además, se tomó en cuenta el conocimiento técnico de los Tester, ya que, para hacer uso de algunas herramientas para automatizar las pruebas, el Tester debe tener cierto conocimiento en programación.

Se encontró una herramienta open source que cumple con nuestras necesidades, pues la curva de aprendizaje es rápida y fácil de entender, ya que existe mucha información al respecto, además, permite trabajar con diferentes lenguajes de programación muy populares, entre ellos; Java, Python, Ruby, C#, etc.

Por otra parte, la herramienta seleccionada puede ser utilizada también por el área de Desarrollo de la organización para sus pruebas unitarias y funcionales, que en un proceso DevOps ayuda a minimizar los errores.

La herramienta seleccionada es Selenium, esta ha sido probada por grandes organizaciones con buenos resultados.

Selenium

Selenium está compuesta por diferentes herramientas o componentes de software, cada una con un enfoque diferente para el apoyo a la automatización de pruebas, proporcionando distintas soluciones para abordar los problemas de automatización.

Estas herramientas dan lugar a un amplio conjunto de funciones de prueba dirigidos específicamente a las necesidades de las pruebas de aplicaciones web de todo tipo, proporcionando operaciones muy flexibles para la localización de elementos de interfaz de usuario y la comparación de resultados de la prueba contra comportamiento de la aplicación real.

Una de las características clave de Selenium es el apoyo para la ejecución de las pruebas en múltiples plataformas de navegadores: Explorer, Mozilla Firefox, Google

Chrome, Safari, entre otros

La principal diferencia de Selenium y otras herramientas de prueba para aplicaciones en web, es que Selenium, utiliza un navegador real impulsado a través de JavaScript para reproducir los scripts de pruebas. Esto significa que tiene algunas características únicas que no están disponibles en ninguna otra herramienta de prueba web [Gheorghiu, 2005].

Selenium es la suite de pruebas de automatización de código abierto más grande, lo que ha hecho que las pruebas sean perfectas. Es ampliamente utilizado para probar entre los grandes personajes como Google, Netflix, Fitbit, etc.

En el siguiente gráfico se muestra cómo Selenium Framework se usa ampliamente entre varias compañías de diferentes escalas y tamaños.

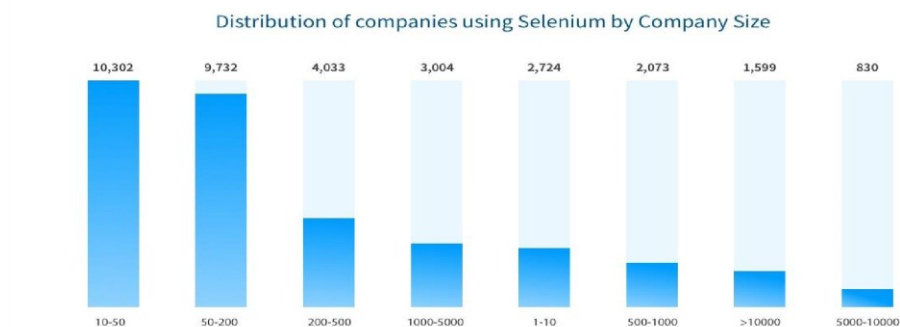


Ilustración 1 Imagen recuperada de: <https://www.browserstack.com/guide/seleniumframework>

Se pueden ejecutar las pruebas directamente en el navegador de destino, impulsar las interacciones en la página web requerida y volver a ejecutarlas sin ninguna entrada manual. Esto elimina las pruebas manuales repetitivas que consumen mucho tiempo y esfuerzo.

Selenium se ajusta a la idea de Agile y DevOps, que respaldan el flujo de trabajo de entrega continua.

Por lo tanto, Selenium es una de las herramientas favoritas para las pruebas, y para la organización SeguriData, es la herramienta que se ha seleccionado como punto de partida para la automatización de las pruebas, pues cumple con el requisito de pruebas rápidas y confiables, lo que ayuda a la empresa a ahorrar tiempo y dinero en las pruebas, teniendo la seguridad de que las pruebas serán ejecutadas y se detectarán errores más rápido, dando atención prioritaria a estos.

El grupo de herramientas (Suite) de Selenium que ayudan en la ejecución de pruebas, está constituido por Selenium Integrated Development Environment (IDE), Selenium Remote Control, Selenium WebDriver y Selenium Grid.

Estas herramientas permiten crear y ejecutar pruebas automatizadas sobre aplicaciones Web.

Estas pruebas automatizadas se utilizarán para pruebas funcionales, pruebas de regresión y para probar la aplicación sobre diferentes plataformas y navegadores Web.

Cada una de estas herramientas son el producto de proyectos que continúan en desarrollo. Estos son llevados a cabo por la Comunidad Open QA y utilizan la licencia Apache 2.0.

Selenium WebDriver

Permite utilizar un navegador de forma local o en remoto. No requiere un servidor especial ya que se inicia una instancia del navegador y se controla de esta manera. La funcionalidad principal de WebDriver es controlar el navegador. Incluso nos ayuda a seleccionar los elementos de la página HTML y realizar operaciones sobre ellos como hacer clic, rellenar campos de formulario, etc.

- Soporte del sistema operativo: Windows, Mac OS, Linux, Solaris.
- Soporte de navegador: Mozilla Firefox, Internet Explorer, Google Chrome 12.0.712.0 y superior, Safari, Opera 11.5 y superior, Android, iOS, HtmlUnit 2.9 y superior.

La API transfiere los scripts que están escritos en diferentes lenguajes de scripts al lenguaje incorporado de Selenium: Selenese, con la ayuda de enlaces. Estos enlaces específicos del lenguaje, como Selenium Java, Selenium Ruby, Selenium dotnet, etc. y la API se almacenan en la Biblioteca.

Mientras que el controlador, que es un módulo ejecutable, abre la instancia del navegador y ejecuta el script de prueba. Estos controladores pueden ser específicos del navegador, como Chromedriver para Google Chrome.

Los marcos como Cucumber y TestNG se utilizan para probar la conexión del cliente de WebDriver y el navegador / controlador. Estas pruebas también se pueden realizar en múltiples controladores con la ayuda de un Selenium Grid.

Selenium IDE

Selenium IDE es un complemento de Chrome y Firefox, que es capaz de registrar interacciones naturales en el navegador y generar su código en diferentes lenguajes de programación. Estos lenguajes de programación incluyen Java, Python, Ruby, C#.

Nos permite grabar diferentes casos de prueba dentro del IDE y reproducir el escenario de prueba en el navegador, donde el IDE puede reproducirlos y resaltar los errores en el código con color rojo.

Selenium IDE será útil para realizar las pruebas de regresión, donde puede usarse para crear prototipos de los errores, ya que el código puede no ser muy legible.

Por lo tanto, se recomiendan ampliamente las pruebas de navegador cruzado utilizando Selenium WebDriver.

Es muy fácil de entender y utilizar. No se necesita tener conocimientos de programación, aunque es importante estar familiarizado con los comandos especiales propios de Selenium.

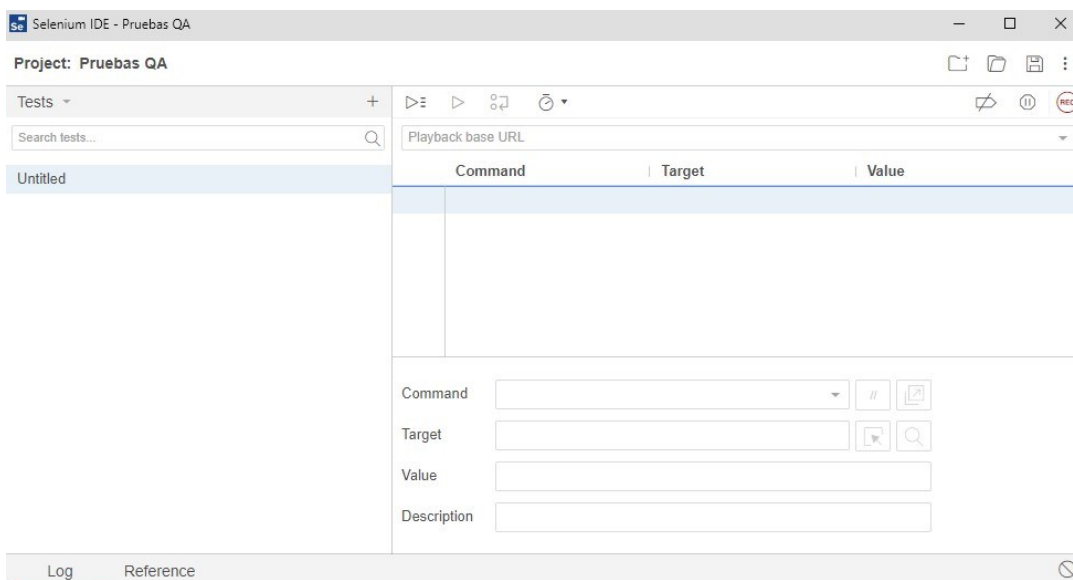


Ilustración 2 Selenium IDE en Chrome

2.1.7 Implementación de herramientas DevOps para la automatización de pruebas.

En esta primera etapa para la implementación se define cual será el proyecto piloto, donde se aplicará la automatización.

Considerando las características del proyecto piloto:

- Requiere agilidad en la respuesta
- Requiere varios ciclos de pruebas
- Se tienen definidas matrices de pruebas

Una vez que se selecciona el proyecto que cumple con las características anteriores, se definen los recursos que estarán interactuando con la herramienta, tanto del área de QA como de Desarrollo.

Y con esta información se define un cronograma con las actividades en torno al uso de la herramienta Selenium.

Los beneficios que esperamos al usar Selenium ejecutando pruebas automatizadas con mayor rapidez y frecuencia son:

- Disminución de costos
- Disminución de tiempo
- Detección de errores de funcionalidad

De manera que, simplemente ejecutando las pruebas que tenemos preparadas podemos obtener un rápido *feedback* sobre el estado de calidad del sistema.

Esto permite reducir no solo costos en el área de calidad, sino también nos ayuda a reducir costos de desarrollo y a dar una respuesta ágil al cliente.

La automatización brinda la capacidad de testear en paralelo, en forma desatendida y en distintas plataformas, pudiendo así, con los mismos costos, verificar cómo funciona la aplicación en , Chrome, Internet Explorer y en Firefox, por ejemplo. Si esto se pretende hacer de forma manual se duplicarían los costos, o sea, los mismos casos de prueba deberían ejecutarse en cada

plataforma, pero en cambio automatizando las pruebas es simplemente indicar que ejecute sobre un entorno u otro.

¿Por qué Selenium es tan popular para las pruebas de automatización?

Selenium Framework, que ha sido utilizado por más de 41 mil empresas en todo el mundo, disfruta de la mayor participación de mercado entre las otras herramientas de prueba de software contemporáneas. Un estudio de Enlyft (anteriormente iDatalabs) sugiere que Selenium tiene una cuota de mercado del 28,36%.

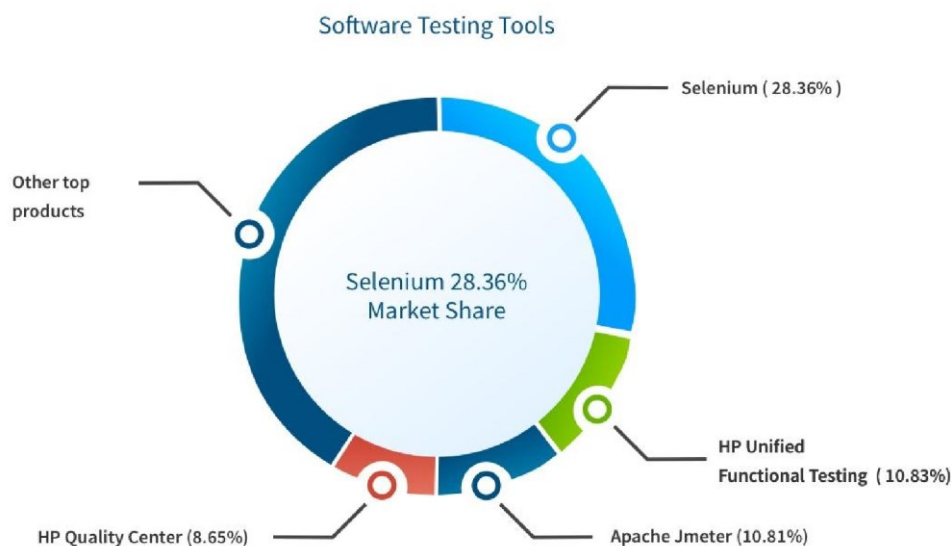


Ilustración 4 Fuente: <https://enlyft.com/tech/products/selenium>

La razón principal detrás de una popularidad tan abrumadora de Selenium es que es de código abierto.

Esto no solo nos ayudará a mantener los costos bajo control, sino que también garantiza que la empresa esté utilizando una herramienta que se actualizará continuamente.

Otras razones incluyen la flexibilidad multidimensional que ofrece en términos de lenguajes de script, sistemas operativos, navegadores e integración con otras herramientas.

Esto amplía la escala de cobertura y prueba, permitiendo el área de QA entregue un producto que sea altamente confiable y funcional.

En función de lo expuesto y realizando un análisis en la organización SeguriData, se determina que podemos tomar como proyecto piloto el Producto Vertical SeguriSign.

2.1.8 Implementación de Selenium para pruebas de software a Producto Vertical SeguriSign.

Actualmente la empresa SeguriData, ofrece varias soluciones para las necesidades de diferentes clientes.

Entre las soluciones que ofrece están:

- Documentos electrónicos

Agilizan y aseguran la firma de documentos o contratos electrónicos con valor legal.

- Cifrado de Información

Ayuda al cumplimiento del GDPR, superando al mismo tiempo los obstáculos y el gasto que implica esta estrategia de seguridad desarticulada.

- PSC

Están acreditados por la Secretaría de Economía. Certificados Digitales, Sellos de Tiempo, NOM 151 y Digitalización certificada como TLA.

- Onboarding Digital

Ofrecen soluciones y servicios para todos los tipos de organizaciones: bancos, gobierno, justicia y empresas. Es una solución integral para evitar el robo de identidad que, mediante el uso de biometría, garantiza al instante la autenticidad de la identidad del usuario.

2.1.9 De la evaluación de los productos, se toma la Decisión.

Se ha elegido al servicio de Firma de documentos electrónicos como proyecto piloto que se implementara a partir del segundo semestre del año 2020.

En la organización SeguriData se sabe cuánto puede costar la gestión de documentos en papel y los riesgos que conlleva, como la validez legal o la pérdida total de la información, o el problema de la distancia, por eso se cuenta con el desarrollo de una solución:

SeguriData cuenta con una Plataforma Tecnológica para la gestión de documentos y trámites con Firma Electrónica Avanzada, de manera segura y con valor probatorio.

Entre los componentes que hacen funcionar esta plataforma, se encuentra la Vertical SeguriSign, ya que permite la gestión de documentos electrónicos, donde uno de estos documentos son los recibos de nómina de los empleados, los cuales son firmados con firma electrónica avanzada, o bien, hacen uso de la firma biométrica.

El producto mencionado está formado por al menos 5 web services y 3 portales y es uno de los productos con mayor dinamismo en su actualización de versiones, porque se personaliza en función de las necesidades y herramientas que utilizan los clientes, por lo que la cantidad y la frecuencia de las pruebas es mucha.

¿Qué se espera de las pruebas del producto Vertical SeguriSign?

Se busca garantizar que la plataforma cuente con el más alto índice de calidad, en funcionalidad y usabilidad y que el producto cumpla con los siguientes criterios de aceptación:

- Legalidad en los documentos procesados.
- Accesibilidad a la información desde cualquier dispositivo.
- Certeza de la integridad y autenticidad de la información con el paso del tiempo.
- Validación desde cualquier visor de PDF.
- Respaldo por la Ley Federal de Firma Electrónica, Código de Comercio y Leyes Estatales de Medios Electrónicos.
- Cumplimiento con los flujos de negocio de los clientes

¿Cómo funciona la plataforma de firma?

Esta plataforma funciona bajo entorno web, y permite firmar documentos desde el portal web o desde dispositivos móviles Android y iOS.

También, emplea certificados digitales de cualquier autoridad certificadora (SAT, PSC- Prestador de Servicios de Certificación u otra).

Y algo muy importante es que se obtiene evidencia electrónica del documento firmado, cuándo y quienes firmaron, adicionando una constancia de conservación de mensajes de datos NOM 151 SCFI 2016

El producto Vertical SeguriSign está compuesto por los siguientes componentes:

- Web Services de Operaciones
- Web Services de Administración y Reportes
- Web Services de Autenticación
- Buzón Vertical SeguriSign
- SeguriSign Web services PDF
- Portal de Administración
- Portal de Agente
 - Portal de Firma

Los componentes específicos a probar de la Vertical SeguriSign, serán:

- Portal de Administración
 - Administrador de dominio
 - Administrador de Plataforma
- Portal de Agente
- Portal de Firma
 - Perfil 4 – Usuario estándar
 - Perfil 8 – Usuario de Servicio
 - Perfil 9 – Usuario Llaves externas
 - Perfil 13 – Usuario Llaves Importadas
 - Perfil 14 – Usuarios Invitados

Análisis de las pruebas

Actualmente, el área de calidad de la empresa SeguriData, cuenta con matrices de prueba de los diferentes portales. Aquí se especifican algunos datos como: Tipo de pruebas, Versión de la aplicación, Número de caso de prueba, Código de caso de prueba, Nombre de caso de prueba, Descripción para ejecutar cada caso de prueba de forma manual, Resultados esperados y Resultados actuales; entre otros.

Se busca automatizar el mayor número de casos de prueba posibles, aunque cabe destacar que existen casos de prueba que no es posible ejecutar más de una vez, pues existen algún tipo de condición que nos lo impide.

¿Qué se va a automatizar?

- Pruebas de regresión

Entendemos una regresión como la parte del proceso de pruebas en donde se tiene interacción con el área de Desarrollo al reportar bugs resultado de las pruebas. En las pruebas de regresión el área de Desarrollo soluciona los bugs reportados, sin embargo, pueden existir nuevos cambios en la aplicación que puedan afectar a las funcionalidades que ya funcionaban correctamente. Estas pruebas se deben aplicar en cada entrega.

- Repetición de un mismo caso de prueba con variantes

Existen casos de prueba en donde se deben hacer pruebas con diferentes datos., es decir, aplicar lo que llamamos el happy path y “n” variaciones más, pensando en usuarios finales sin experiencia.

- Automatizaciones de Happy Path

Reduce el esfuerzo humano. Nos evita realizar de forma manual muchos casos de prueba ya definidos en las matrices como happy path y nos ayuda a enfocarnos en las nuevas funcionalidades.

- Probar en varios ambientes

Permite realizar las mismas pruebas en diferentes ambientes de prueba. Es decir, se pueden utilizar el mismo set de pruebas para el ambiente de desarrollo o QA. Además, permite aplicar las pruebas a diferentes navegadores y dispositivos. Esto nos ayuda a reducir el esfuerzo humano y aumenta nuestra productividad para otros casos de prueba.

- Pruebas de aceptación de usuario

Nos ayudan para realizar pruebas de “Camino Feliz” y estas sean aceptadas por el usuario.

Esto nos garantiza que nuestro sistema tenga calidad, no solo comprobando que las pruebas de camino feliz son ejecutadas de manera exitosa, sino, demostrar y tener la seguridad que las pruebas de camino no feliz, - que son pruebas que usualmente el usuario no realizaría, pero si es posible que se presenten, muestren los mensajes de error correctos, de acuerdo a la lógica del negocio establecida.

- Pruebas repetitivas

Las aplicaciones en las que más conviene usar testing automatizado son entonces las que en algún sentido tienen mucha repetitividad, ya que será necesario ejecutar muchas veces las pruebas (ya sea porque es un producto que tendrá muchas versiones, que se continúe con el mantenimiento haciendo fixes y parches, o porque se debe probar en distintas plataformas).

Si se automatizan las pruebas de un ciclo de desarrollo, para el siguiente ciclo la prueba automatizada podrá volver a testear lo que ya se automatizó con bajo esfuerzo, permitiendo que el equipo de pruebas aumente el tamaño del conjunto de pruebas, aumentando el cubrimiento. Si no fuera así, terminaríamos teniendo ciclos de pruebas más grandes que los ciclos de desarrollo (y cada vez más grandes) u optaríamos por dejar cosas sin probar, con los riesgos que eso implica.

La siguiente gráfica muestra uno de los factores más importantes, y es la cantidad de veces que vamos a repetir la ejecución de un caso de prueba.

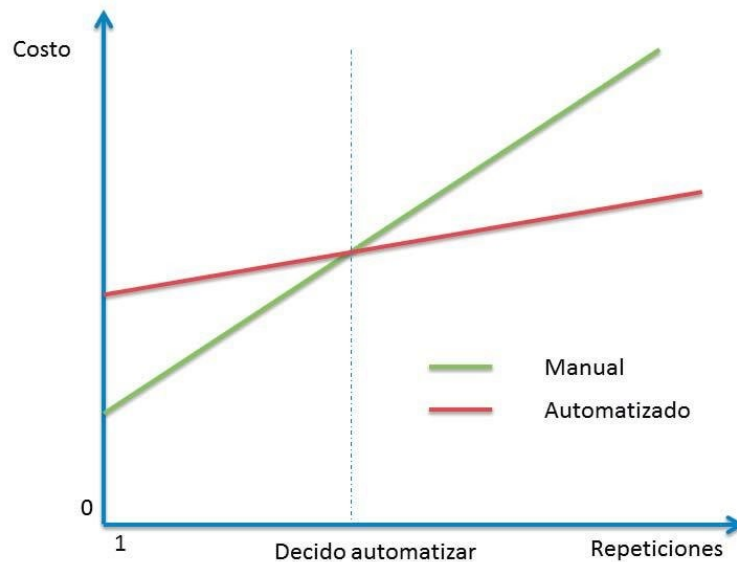


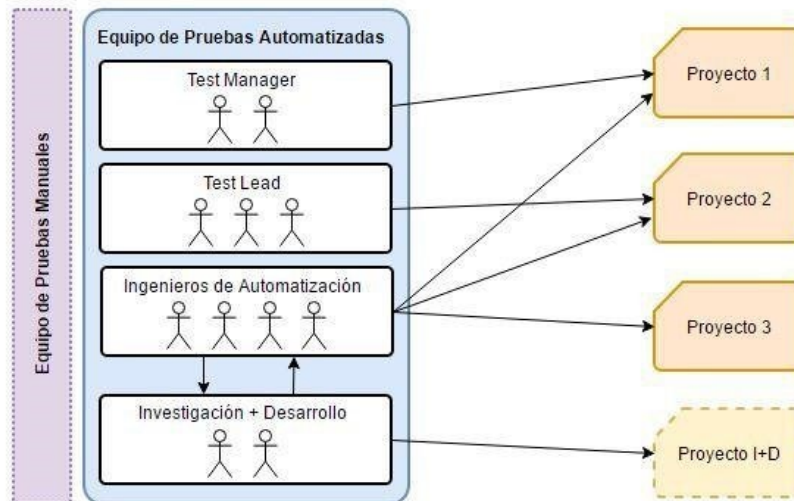
Ilustración 5 DECISIÓN DE AUTOMATIZAR O NO AUTOMATIZAR

Existen varios factores a tomar en cuenta para determinar que pruebas se van a ejecutar y cuáles no.

La cantidad de veces está determinada por muchas cosas: por el número de versiones de la aplicación que vamos a probar, o las distintas plataformas sobre las que la vamos a ejecutar, o incluso los datos, pues si un mismo caso de prueba lo tenemos que ejecutar muchas veces con distintos datos, eso también lo tendremos que tener en cuenta al decidir si automatizar o no.

¿Cómo se debería organizar el área de QA y para conseguir un mayor beneficio de las pruebas automatizadas a medio y largo plazo?

En la empresa SeguriData, se llevan a cabo un gran número de proyectos a día de hoy. Esto ocurre porque cualquier proyecto tiene fecha de inicio y fin, así que constantemente se crean y se cierran nuevos proyectos y los recursos asignados van rotando según los requerimientos de cada uno.



Si nos fijamos en cualquier empresa que valore la importancia de la calidad de sus productos y servicios, probablemente habrá contratado profesionales especializados en pruebas, ya sean externos o internos.

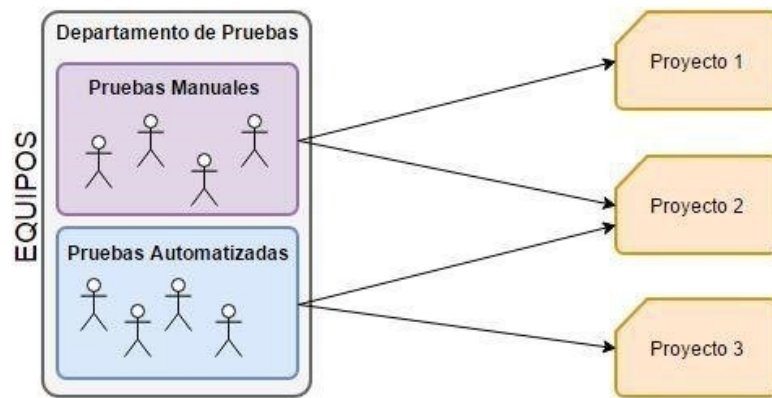
Según la organización y el tamaño de la empresa, estos recursos pueden formar parte del equipo de Testing y Calidad o estar contratados exclusivamente para proyectos específicos.

¿Qué se busca con la implementación de la automatización de las pruebas?

Los integrantes del área de QA, deberán formar parte del mismo equipo dentro de la empresa, a no ser que esta tuviera varias líneas de negocio extremadamente distintas.

En cuanto a las personas especializadas en automatización de pruebas, estas deberían considerarse como un equipo autónomo incluido dentro del equipo de *Testing*.

Esto ocurre debido a que las características de los procesos de automatización no son las mismas si las comparamos con las actividades de los Testers, Test Leads y Test Managers encargados de gestionar las pruebas manuales.



Un ejemplo de estas diferencias entre los equipos de pruebas manuales y automatizadas es el ritmo de cambio que existe actualmente. Las metodologías y herramientas utilizadas para el *testing* manual no evolucionan a la misma velocidad que las de automatización. Es por ello que debemos tratarlos a ambos como equipos independientes, aunque incluidos dentro del mismo grupo genérico.

Diseño y organización de casos de prueba

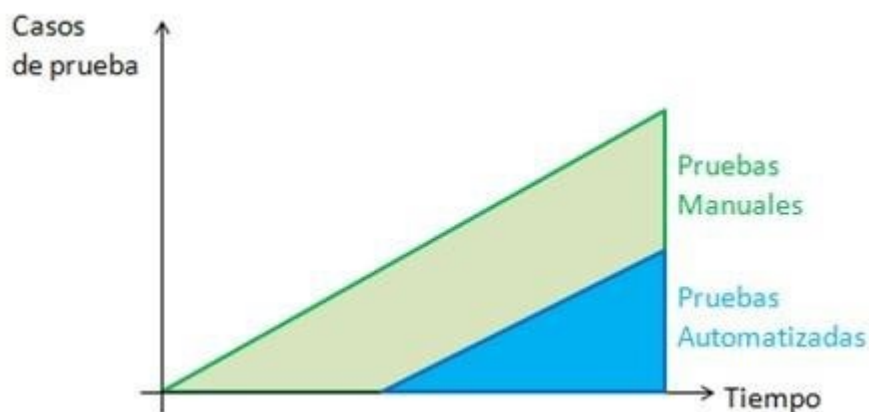
¿Probar solamente nuevas funcionalidades?

En la validación de una nueva funcionalidad, las pruebas manuales son de las mejores opciones para la detección de defectos. Ahora bien, también se puede decir que son lentas y, por tanto, repetirlas resulta bastante caro tanto a nivel económico como temporal.

Como se dispone de un tiempo y de unos recursos limitados para probar, solo se pone énfasis en probar las nuevas funcionalidades y de manera rápida las funcionalidades que ya existían.

Pero, ¿qué ocurre si el desarrollo de una nueva característica provoca un fallo en partes de la aplicación que ya habían sido desarrolladas y probadas y que anteriormente funcionaban de manera correcta? Es necesario realizar pruebas de regresión.

Precisamente para esto sirven las pruebas automatizadas.



Crear pruebas automatizadas permite ampliar la cobertura de pruebas, detectar los defectos antes y focalizar las pruebas manuales en lo que realmente interesa.

RESULTADOS

- Mejoras al proyecto

La automatización se llevó a cabo tomando en cuenta los siguientes aspectos:

- Análisis de la aplicación
- Preparación de la infraestructura
- Configuración del ambiente de prueba
- Crear los casos de prueba
- Complementación de las pruebas
- Ejecución y Depuración

A continuación, se describe a detalle cada aspecto aplicado al proyecto.

3.1. Análisis de la aplicación.

Se identifica que softwares y herramientas son idóneas para automatizar nuestra aplicación.

Se eligió a Selenium IDE y Selenium WebDriver como herramienta de automatización. Así mismo, se utilizó el framework TestNG, siendo éste un marco de pruebas automatizadas de código abierto; donde NG significa Next Generación.

3.1.1 Selenium Web Driver

Como se mencionó anteriormente, una de las grandes ventajas de trabajar con Selenium Web Driver, es que los scripts de prueba se pueden escribir en cualquiera de estos lenguajes de programación: Java, Python, C #, PHP, Ruby, Perl y .Net. Se utilizó esta herramienta de código abierto para automatizar las pruebas realizadas en los navegadores web.

3.1.2 Selenium IDE

Con esta extensión de Chrome o Firefox, se puede realizar la automatización de pruebas interactuando directamente con el navegador. Podemos crear diferentes Test y agruparlos en suites de pruebas. Bastaría con realizar la grabación de cada test simplemente ingresando a la URL deseada y dando clic a cada elemento del navegador, simulando así, cada caso de prueba deseado.

Sin embargo, esta herramienta se utilizó únicamente para obtener algunos elementos “Web Element” del navegador necesarios para poder codificarlos en Selenium Web Driver.

3.1.3 TestNG

Selenium se puede integrar con herramientas como TestNG y JUnit para gestionar casos de prueba y generar informes. En esta ocasión, se trabajó con TestNG como framework para la gestión de nuestras pruebas,

Con TestNG, haremos uso de annotations en los métodos para poder configurar nuestras ejecuciones de diferentes maneras, lo cual es un plus para poder organizar de mejor manera las pruebas.

3.2 Preparación de la infraestructura.

En esta etapa, se establecen las buenas prácticas de programación que vamos a utilizar, que lenguaje de programación utilizaremos, bajo que metodología se trabajara, que software/ IDE se utilizara, si se trabajara con algún patrón de diseño, etc.

La configuración de Selenium es bastante diferente de la configuración de otras herramientas comerciales. Para usar Selenium en el proyecto de automatización, se necesita instalar las librerías para el lenguaje de programación, en este caso Java. Además, se necesitarán los binarios del WebDriver para los navegadores en los que se desea automatizar y ejecutar la prueba.

Para poder utilizar WebDriver para la creación de scripts, existen algunos requisitos previos que deben estar en un lugar como la configuración básica del entorno. Se eligió el Lenguaje Java y el IDE Eclipse para el desarrollo de las pruebas, los cuales tienen los siguientes requisitos:

3.2.1 Instalación de Java.

Paso 1. Ir al sitio oficial de Oracle (www.oracle.com), entrar en la sección de “Downloads” y seleccionar “Java Platform (JDK)”.

Paso 2. Estando en la sección indicada en el paso 1, se mostrará una pantalla donde se elige la versión de «Java Platform (JDK)» que se desea instalar (Se debe tener en cuenta que según la versión de Java se instalará la versión de eclipse), después, aceptar el acuerdo de licencia y seleccionar la plataforma a la cual se desea descargar Java. Sería algo como lo siguiente:

Java downloads Tools and resources Java archive

The Oracle JDK 8 license changed in April 2019

The Oracle Technology Network License Agreement for Oracle Java SE is substantially different from prior Oracle JDK 8 licenses. This license permits certain uses, such as personal use, at no cost -- but other uses authorized under prior Oracle JDK licenses may no longer be available. Please review the terms carefully before downloading and using this product [here](#).

Commercial license and support are available for a low cost with [Java SE Subscription](#).

JDK 8 software is licensed under the [Oracle Technology Network License Agreement for Oracle Java SE](#).

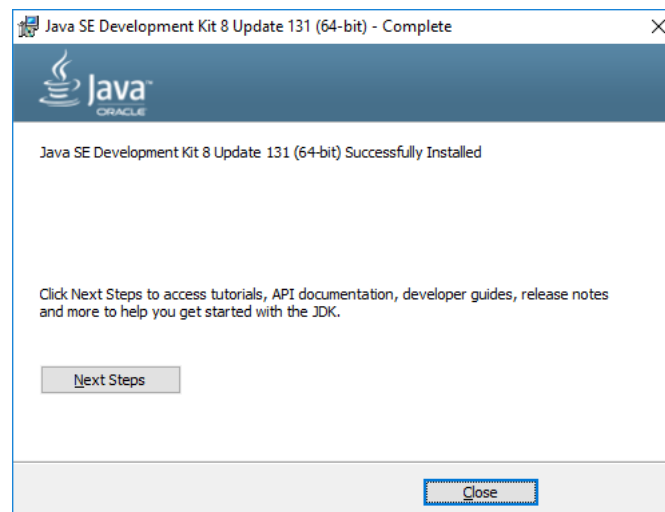
JDK 8u361 checksum

Linux macOS Solaris Windows

Product/file description	File size	Download
x86 Installer	135.96 MB	jdk-8u361-windows-i586.exe
x64 Installer	144.69 MB	jdk-8u361-windows-x64.exe

[Documentation Download](#)

Paso 3: Ejecutar el archivo descargado en el paso anterior, la instalación es muy similar a la de cualquier programa, simplemente damos en «Next» hasta que ya inicia la instalación, al final te dirá que Java SE Development Kit ha sido instalado correctamente.



Paso 4: Crear variable de entorno JAVA_HOME

Paso 5: Actualizar variable de entorno Path con directorio de JDK

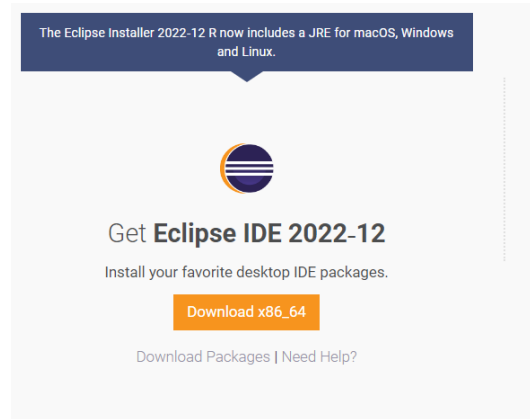
3.2.2 Instalación del IDE Eclipse.

Como entorno de desarrollo para nuestras pruebas automatizadas con Selenium Webdriver, utilizamos Eclipse. A continuación, se muestra la información necesaria para su instalación y configuración.

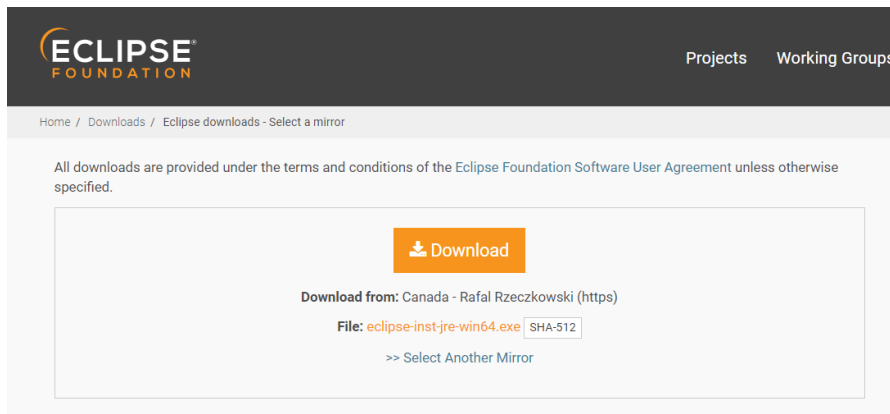
Paso 1: Abrir el navegador e ingresar al siguiente enlace:

<https://www.eclipse.org/downloads/>.

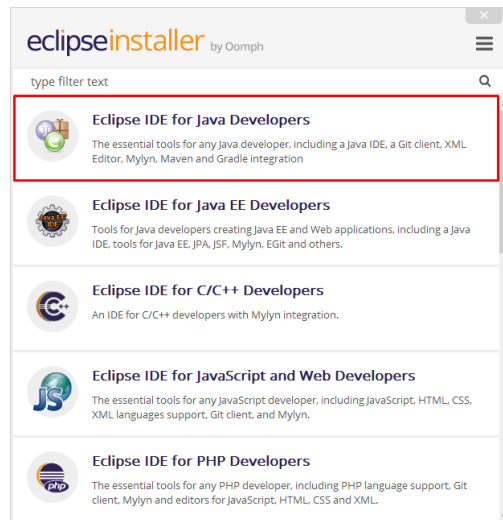
Se mostrará la siguiente pantalla, donde aparecerá la versión más reciente de eclipse con el botón “Download x86_64”. Dar clic en “Download x86_64”.



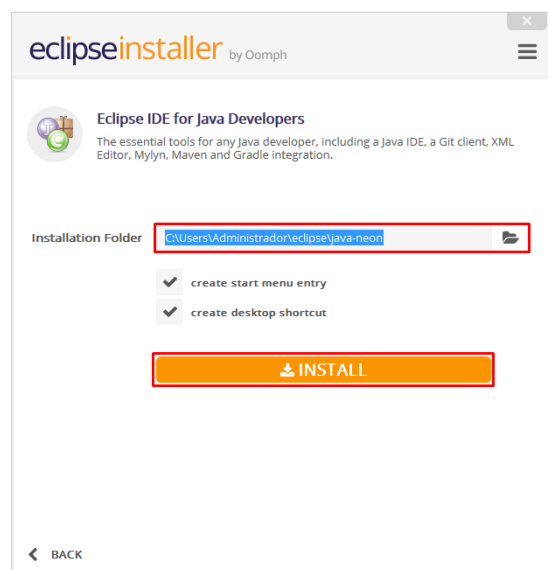
Paso 2. Después de dar clic en el botón “Download x86_64” se abrirá la siguiente pantalla y ahora presionamos el botón “Download” e inmediatamente se empezará a descargar el archivo .exe.



Paso 3: Una vez descargado el archivo .exe, se debe ejecutar, donde posteriormente se abrirá un menú de instalación con las opciones que tenemos para instalar como Java EE, Java SE, eclipse para C++, entre otros, en este caso seleccionaremos la opción “Eclipse IDE for Java Developers”.

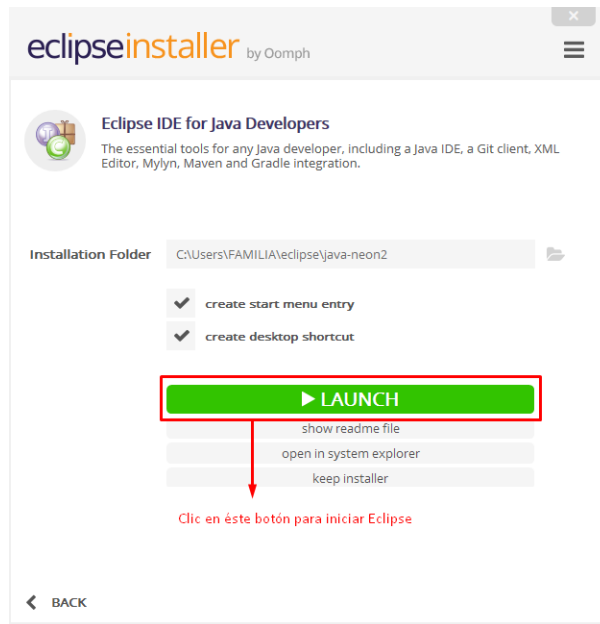


Paso 4: En el campo Installation Folder se establece la ruta de instalación de Eclipse IDE y le damos clic en el botón “Install”.

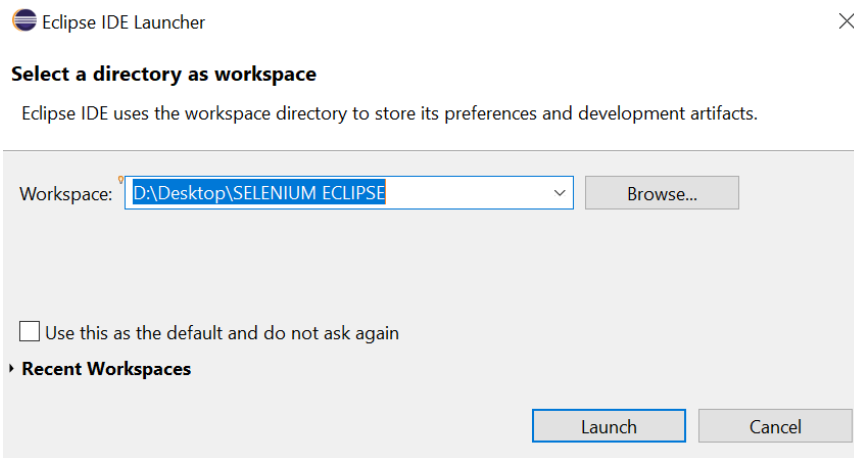


Al terminar la instalación de Eclipse para empezar a construir nuestras pruebas con Selenium Webdriver, se debe mostrar algo como en la imagen. Esto significa que estamos listos y que eclipse se instaló correctamente.

Presionar el botón “Launch”.



Paso 5: Finalmente, al ser primera vez que se ejecuta Eclipse, se pedirá una ruta para el workspace. Al tener la ruta seleccionada, dar clic en el botón “Launch”.



Con los pasos descritos anteriormente ya se tiene Java y Eclipse con lo cual, ya se podría empezar a configurar el ambiente para crear y ejecutar pruebas automatizadas con Selenium Webdriver.

3.2.3 Creación de proyecto Maven

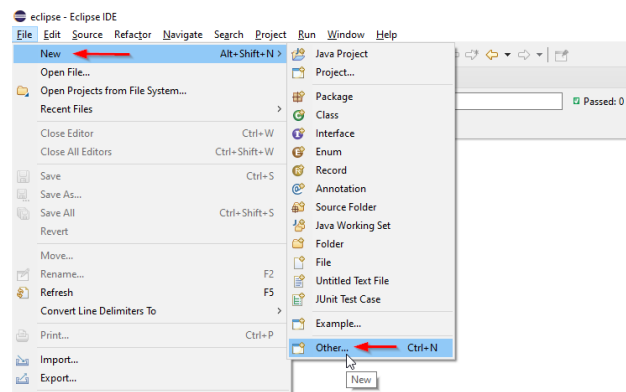
Maven, es una herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida, como por ejemplo en todas las fases de compilación y empaquetado y la instalación de mecanismos de distribución de librerías, para que puedan ser utilizadas por otros desarrolladores y equipos de desarrollo.

En el proyecto Maven, se estará gestionando el código, dependencias, librerías, etc.

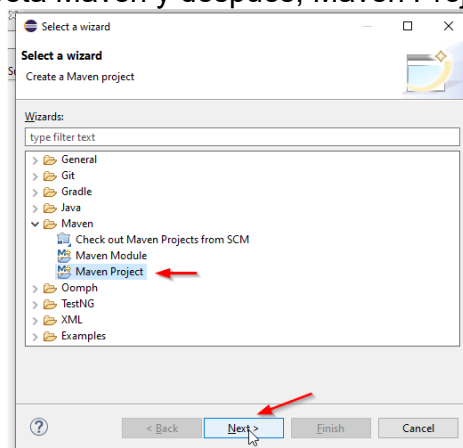
Las dependencias que se estarán importando a este proyecto serán Selenium y TestNG. Gracias a estas dependencias, se podrá ejecutar las pruebas de Selenium con clases de tipo TestNG.

Para crear el proyecto Maven, se deben seguir los siguientes pasos:

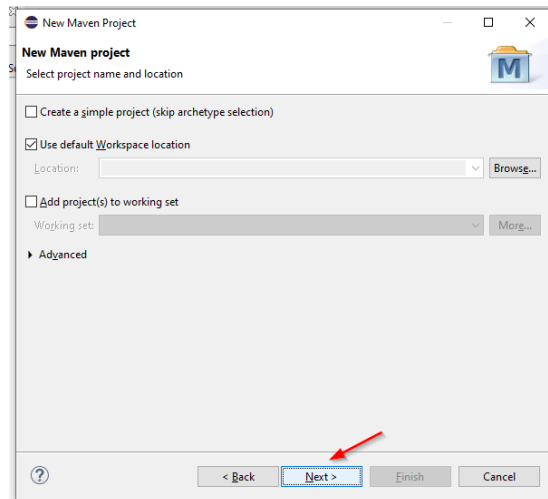
Paso 1. Crear nuevo proyecto dando clic en File -> New -> Other



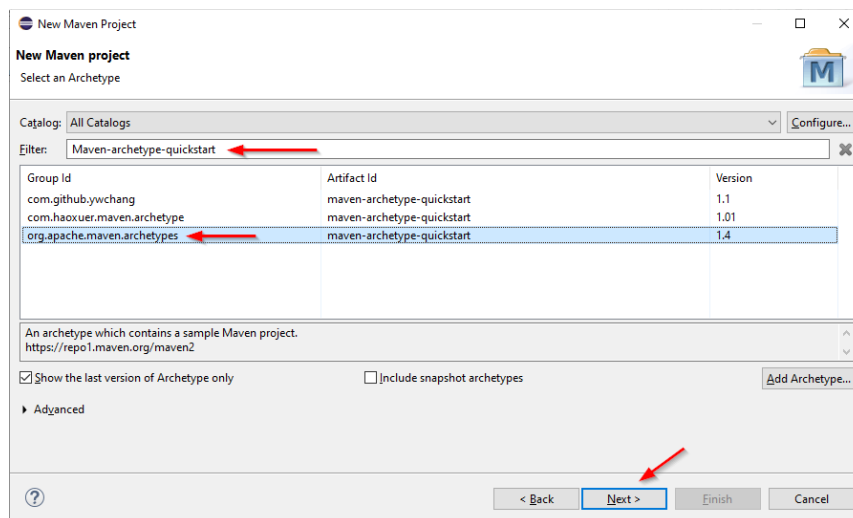
Paso 2. Seleccionar carpeta Maven y después, Maven Project. Dar clic en Next.



Paso 3. En la siguiente pantalla, dar clic en Next.



Paso 4. En Filter, escribir lo siguiente: **“Maven-archetype-quickstart”**. Después seleccionar del Group ID la opción de **org.apache.maven.archetypes**. Dar clic en Next.



Paso 5. En la siguiente pantalla, ingresar el **Group ID**, en este caso, es el nombre del proyecto. También ingresar el **Artifact ID**, en este caso, VerticalSSign. Finalmente, dar clic en Finish.

New Maven Project

Specify Archetype parameters

Group Id: PortalDeFirma

Artifact Id: VerticalSSign

Version: 0.0.1-SNAPSHOT

Package: PortalDeFirma.VerticalSSign

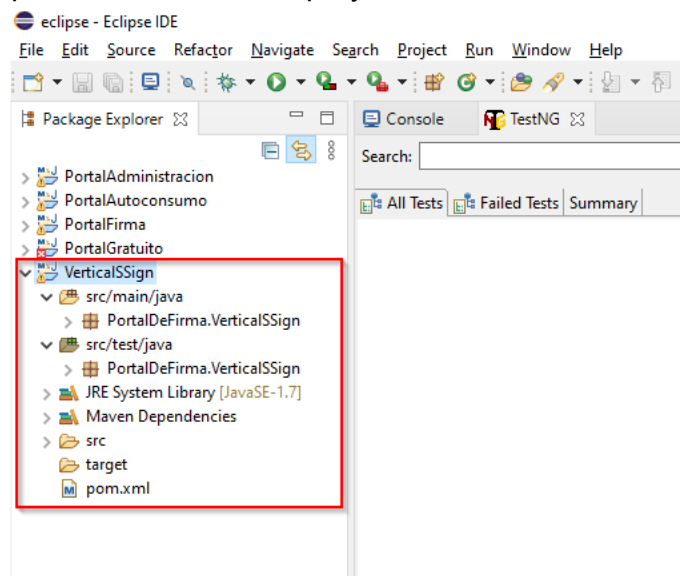
Properties available from archetype:

Name	Value

Advanced

< Back Next > Finish Cancel

De esta manera, queda estructurado el proyecto



3.2.4 Instalación de Dependencias en Maven

Para poder ejecutar las pruebas de Selenium en Java, necesitamos cargar tanto sus librerías para interactuar con los comandos de Selenium como las librerías necesarias para cargar los drivers respectivos para cada browser.

Las dependencias que se deben descargar en el proyecto Maven y que se utilizaran en el desarrollo de las pruebas, son las siguientes:

- TestNG

TestNG es un marco de prueba inspirado en JUnit y NUnit pero que presenta algunas funcionalidades nuevas que lo hacen más poderoso y fácil de usar. Admite pruebas configuradas por anotaciones, pruebas basadas en datos, pruebas paramétricas, etc.

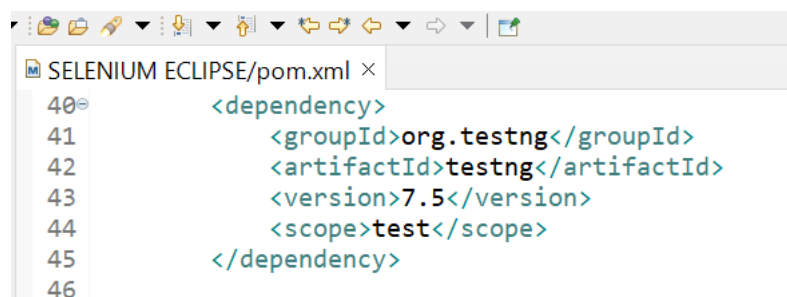
De acuerdo con <http://testng.org/doc/download.html> para instalar testng, se debe agregar la dependencia de testng al archivo pom.xml

Para eso, se deben seguir los siguientes pasos:

Paso 1. Ingresar al repositorio de dependencias Maven ingresando a la siguiente liga: <https://testng.org/doc/maven.html> y copiar la dependencia.

```
<dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>7.5</version>
  <scope>test</scope>
</dependency>
```

Paso 2. Ir al archivo POM.xml y pegar la dependencia dentro de la lista de dependencias. Guardar el archivo.



- Selenium

Selenium proporciona soporte para la automatización de navegadores web. Proporciona extensiones para emular la interacción del usuario con los navegadores, un servidor de distribución para escalar la asignación del navegador y la infraestructura para implementaciones de la especificación W3C WebDriver.

Para instalar la o las dependencias de Selenium para el proyecto, se deben seguir los siguientes pasos:

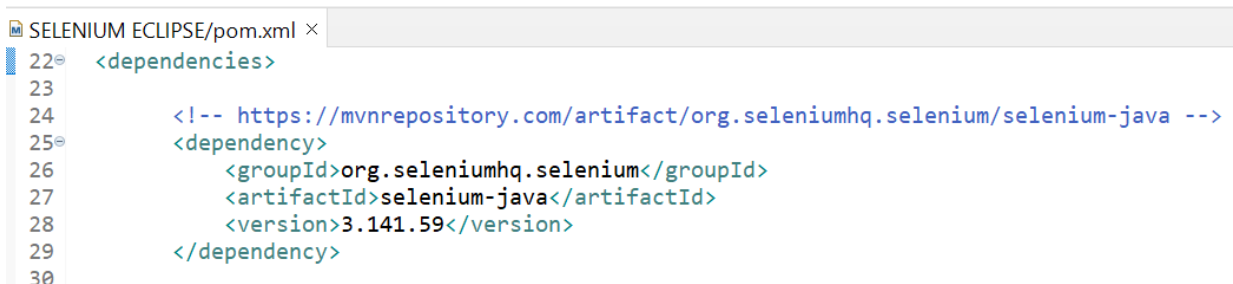
Paso 1. Ingresar al repositorio de dependencias Maven <https://mvnrepository.com/> y buscar la dependencia de Selenium, o bien ingresar directamente al repositorio con la dependencia de Selenium en la siguiente liga: <https://mvnrepository.com/artifact/org.seleniumhq.selenium>
Copiar la dependencia de Selenium.

La versión utilizada en este proyecto es la 3.141.59:

```
<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>3.141.59</version>
</dependency>
```

La dependencia de selenium-java admite la ejecución del proyecto de automatización con todos los navegadores compatibles con Selenium.

Paso 2. Ir al archivo POM.xml y pegar la dependencia dentro de la lista de dependencias. Guardar el archivo.



```
SELENIUM ECLIPSE/pom.xml x
22<dependencies>
23
24    <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
25    <dependency>
26        <groupId>org.seleniumhq.selenium</groupId>
27        <artifactId>selenium-java</artifactId>
28        <version>3.141.59</version>
29    </dependency>
30
```

Si se desea ejecutar pruebas solo en un navegador específico, se puede agregar la dependencia para ese navegador en el archivo pom.xml. Por ejemplo, se debe agregar la siguiente dependencia en su archivo pom.xml para ejecutar sus pruebas solo en Firefox:

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-firefox-driver</artifactId>
  <version>3.X</version>
</dependency>
```

De manera similar, si se desea ejecutar pruebas solo en Chrome, se debe agregar la siguiente dependencia:

```
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-chrome-driver</artifactId>
  <version>3.X</version>
</dependency>
```

▪ Extentreports

Se pueden crear informes bonitos, interactivos y detallados para las pruebas con ayuda de esta librería. Permite agregar eventos, capturas de pantalla, etiquetas, dispositivos, autores o cualquier otra información relevante e importante para crear un informe descriptivo e impactante que se pueda controlar por completo.

Para instalar la dependencia de Extentreports y poder generar reportes, se deben seguir los siguientes pasos:

Paso 1. Ingresar al repositorio de dependencias Maven <https://mvnrepository.com/> y buscar la dependencia de Extentreports, o bien ingresar directamente al repositorio con la dependencia de Extentreports para Selenium en la siguiente liga:

<https://mvnrepository.com/artifact/com.aventstack/extentreports/3.1.5>

La versión utilizada en este proyecto es la 3.1.5

```
<!-- https://mvnrepository.com/artifact/com.aventstack/extentreports -->
<dependency>
  <groupId>com.aventstack</groupId>
  <artifactId>extentreports</artifactId>
  <version>3.1.5</version>
</dependency>
```

Paso 2. Ir al archivo POM.xml y pegar la dependencia dentro de la lista de dependencias. Guardar el archivo.

```

46
47     <!-- https://mvnrepository.com/artifact/com.aventstack/extentreports -->
48     <dependency>
49         <groupId>com.aventstack</groupId>
50         <artifactId>extentreports</artifactId>
51         <version>3.1.5</version>
52     </dependency>
53

```

- LOG4J

Log4j es un marco de registro escrito en Java que proporciona una manera fácil de iniciar sesión en Selenium. En pocas palabras, el marco brinda información sobre todo lo que sucede durante la ejecución del software.

Log4j también proporciona información sobre cualquier cosa que pueda haber salido mal durante la ejecución o automatización del software. En general, Log4j documenta el resultado en forma de registros que se pueden examinar más adelante para fines como la auditoría de proyectos de Selenium a pequeña y gran escala.

Para descargar la dependencia, se deben seguir los siguientes pasos:

Paso 1. Ingresar al repositorio de dependencias Maven <https://mvnrepository.com/>, buscar la dependencia de log4j, y copiar las siguientes dependencias: log4j-api y log4j-core, donde:

- Apache Log4j Core

Implementación para Apache Log4J, una herramienta de registro altamente configurable que se enfoca en el rendimiento y la baja generación de basura. Tiene una arquitectura de complemento que lo hace extensible y admite el registro asíncrono basado en LMAX Disruptor.

```

<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.17.2</version>
</dependency>

```

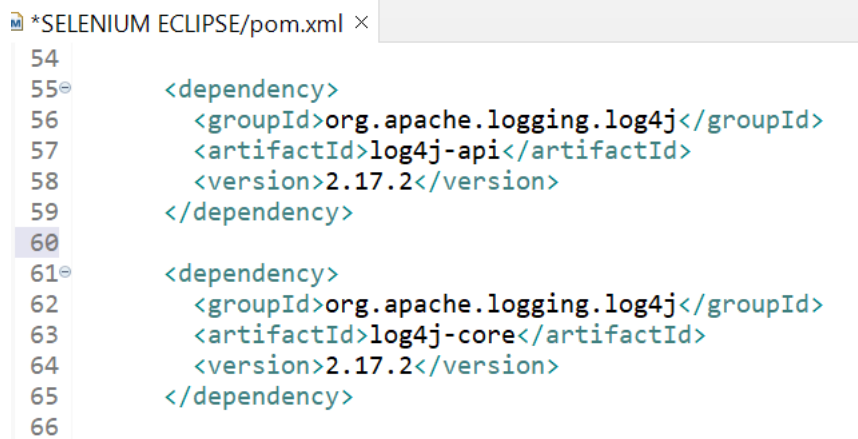
- Apache Log4j API

API para Apache Log4J, una herramienta de registro altamente configurable que se enfoca en el rendimiento y la baja generación de basura. Tiene una arquitectura de complemento que lo hace extensible y admite el registro asíncrono basado en LMAX Disruptor.

```
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-api</artifactId>
  <version>2.17.2</version>
</dependency>
```

La versión utilizada en el actual proyecto, es la 2.17.2

Paso 2. Ir al archivo POM.xml y pegar la dependencia dentro de la lista de dependencias. Guardar el archivo.



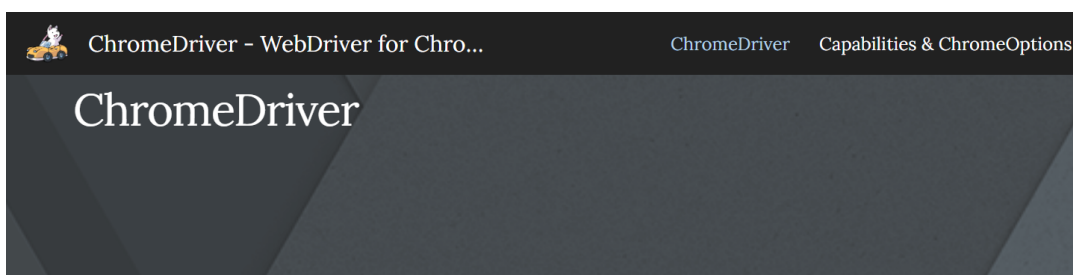
```
*SELENIUM ECLIPSE/pom.xml ×
54
55  <dependency>
56    <groupId>org.apache.logging.log4j</groupId>
57    <artifactId>log4j-api</artifactId>
58    <version>2.17.2</version>
59  </dependency>
60
61  <dependency>
62    <groupId>org.apache.logging.log4j</groupId>
63    <artifactId>log4j-core</artifactId>
64    <version>2.17.2</version>
65  </dependency>
66
```

3.2.5 Como ejecutar pruebas en Selenium en el Navegador Chrome

Binarios/Drivers

Para ejecutar el proyecto y controlar el navegador, se debe tener instalados los binarios de WebDriver específicos del navegador.

Para descargar los binarios para Chrome, se accede a la siguiente liga:
<https://chromedriver.chromium.org/>



WebDriver is an open source tool for automated testing of webapps across many browsers. It provides capabilities for navigating to web pages, user input, and more. ChromeDriver is a standalone server that implements the [W3C WebDriver standard](#). ChromeDriver is available for Chrome on Android and Chrome on ChromeOS).

You can view the current implementation status of the WebDriver standard [here](#).

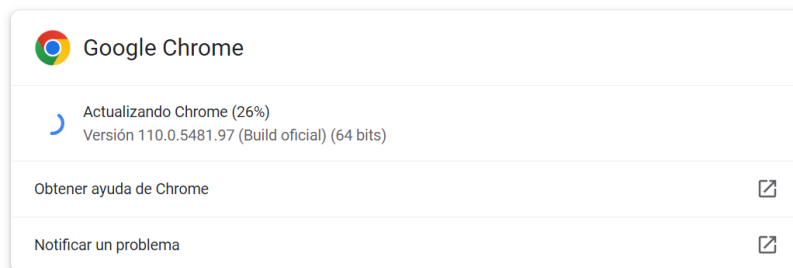
All versions available in [Downloads](#)

Importante:

Se debe descargar el binario del WebDriver compatible con nuestro sistema operativo y con la versión de nuestro navegador.

Es importante descargar la versión correspondiente a nuestro navegador, de otra forma, se pueden presentar errores impidiendo ejecutar las pruebas en el navegador.

Información de Chrome



Ver versión del navegador

En este caso, la versión de nuestro navegador es 110.0.5481.97, así que la versión a descargar del binario ChromeDriver debe ser la que más se adecue a ella.

All versions available in [Downloads](#)

- Latest **beta** release: [ChromeDriver 111.0.5563.19](#)
- Latest **stable** release: [ChromeDriver 110.0.5481.77](#)

Seleccionar versión de ChromeDriver

Index of /110.0.5481.77/

	Name	Last modified	Size	ETag
	Parent Directory	-	-	-
	chromedriver_linux64.zip	2023-02-08 08:07:21	7.05MB	ddb02c4f60356edf4f832df472872a44
	chromedriver_mac64.zip	2023-02-08 08:07:28	8.82MB	2b6c666b02384e07e0e8b0c122a2f647
	chromedriver_mac_arm64.zip	2023-02-08 08:07:36	8.00MB	b0f409e6fb4f12e30b7ecbf44c7280d6
	chromedriver_win32.zip	2023-02-08 08:07:44	6.78MB	5382dfd7e76113a3d95e1bb4e43bca7
	notes.txt	2023-02-08 08:08:00	0.00MB	a14c7343220c189ebfc670de877e27da

Elegir binarios según S.O

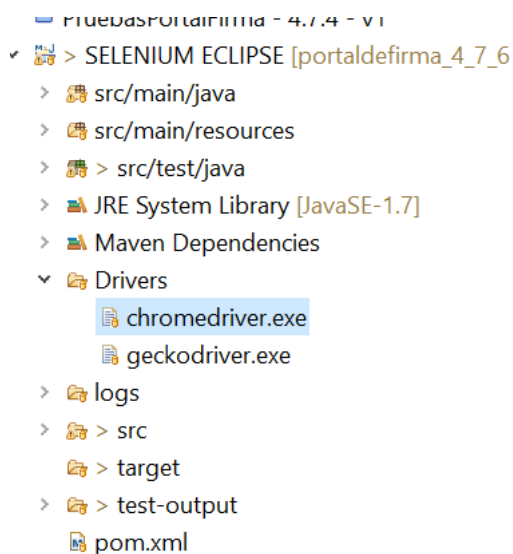
Después, se debe descomprimir el archivo y colocarlo en la RUTA o PATH del sistema.

chromedriver_win32

Nombre	Fecha de modificación	Tipo	Tamaño
 chromedriver	31/01/2023 03:04 p. m.	Aplicación	12,464 KB
 LICENSE.chromedriver	31/01/2023 03:04 p. m.	Archivo CHROMEDRI...	242 KB

ChromeDriver.exe

Una vez que se tiene el archivo descargado, se debe agregar al proyecto en una carpeta llamada “Driver”.



Navegadores admitidos por Selenium WebDriver

El marco Selenium también admite oficialmente los siguientes navegadores:

Navegador	Desarrollador	Versiones soportada
Chrome	Chromium	Todas las versiones
FireFox	Mozilla	54 y recientes
Internet Explorer	Selenium	6 y recientes
Opera	Opera Chromium / Presto	10.5 y recientes
Safari	Apple	10 y recientes.

3.2.6 Metodología de Programación con la que se trabajará

▪ Metodologías ágiles

¿Por qué se necesita automatizar el Software Testing al aplicar metodologías ágiles?

Todas las metodologías ágiles implican constantes iteraciones, en las cuales el código de programa se modifica constantemente.

En cada iteración, además de expandir las funcionalidades del sistema, podemos realizar Refactorizaciones para optimizar funcionalidad que ya desarrollamos y eliminar deuda técnica. Esto implica la necesidad de realizar constantemente pruebas regresión, dado que, al modificar código desarrollado previamente, debemos revisar que continúa funcionando adecuadamente.

Con frecuencia, esto implica:

- Constantes pruebas de regresión.
- Esfuerzo duplicado en pruebas.
- Todos en el equipo terminan haciendo pruebas manuales, inclusive desarrolladores, Testers y el cliente.

Entonces, ¿Se necesita la automatización de pruebas en metodologías ágiles?

La respuesta es un sí definitivo, dado que muchas veces se pueden hacer cambios en la base de código y refactorizaciones, inclusive varias veces al día.

Cuando el software es complejo y tiene muchas interfaces con otros sistemas, es imposible seguir el ritmo de las constantes iteraciones sin automatizar las pruebas de regresión.

¿Como puede ayudarme Selenium WebDriver para esto?

3.2.7 Patrón de diseño: Page object models (POM)

Dentro de la interfaz de usuario de la aplicación web, en este caso, la Vertical SeguriSign, hay áreas con las que interactúan las pruebas. Un objeto de página solo los modela como objetos dentro del código de prueba. Esto reduce la cantidad de código duplicado y significa que, si la interfaz de usuario cambia, la solución solo debe aplicarse en un lugar.

Page Object Model (POM), es un patrón de diseño que se ha vuelto popular en la automatización de pruebas para mejorar el mantenimiento de las pruebas y reducir la duplicación de código. Un objeto de página es una clase orientada a objetos que sirve como interfaz para una página de su AUT. Luego, las pruebas usan los métodos de esta clase de objeto de página cada vez que necesitan interactuar con la interfaz de usuario de esa página. El beneficio es que, si la interfaz de usuario cambia para la página, las pruebas en sí mismas no necesitan cambiar, solo el código dentro del objeto de la página debe cambiar. Posteriormente, todos los cambios para admitir esa nueva interfaz de usuario se ubican en un solo lugar.

Ventajas

- Hay una clara separación entre el código de prueba y el código específico de la página, como los localizadores (o su uso si está usando un mapa de interfaz de usuario) y el diseño.
- Hay un repositorio único para los servicios u operaciones que ofrece la página en lugar de tener estos servicios dispersos a lo largo de las pruebas.

En ambos casos, esto permite que las modificaciones requeridas debido a los cambios en la interfaz de usuario se realicen en un solo lugar.

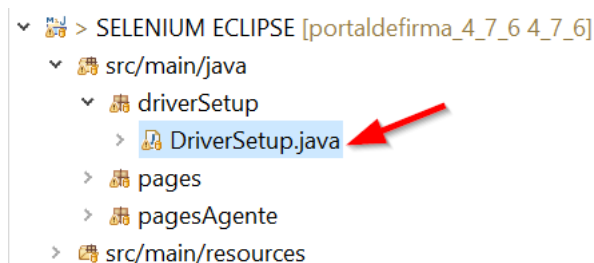
3.3 Creación de casos de prueba

Como se indicó anteriormente, se cuenta con una sólida matriz de pruebas, que es parte de la documentación que actualmente se tiene, la cual, incluye los diferentes casos de prueba a automatizar.

El proyecto estará estructurado de la siguiente manera, el cual, será explicado a detalle.

3.3.1 DriverSetup.java

Esta clase está ubicada en el paquete **driverSetup**, en el cual, se declarará el código que servirá para que se puedan ejecutar las pruebas de Selenium en el navegador deseado, en este caso, las pruebas podrán ser ejecutadas en el navegador de Google Chrome y Firefox.



Esto se puede determinar en las siguientes líneas de código (línea 22 y 36),

```
22  if(prop.getProperty("browser").equalsIgnoreCase("1")) { //FIREFOX
    -
    -
36  } else if(prop.getProperty("browser").equalsIgnoreCase("2")) {
    //FIREFOX
```

donde, se realiza la lectura de la propiedad “browser” ubicada en el archivo de propiedades config.properties, el cual es llamado en el método readConfig() (línea 45).

Código:

```
DriverSetup.java ×
1 package driverSetup;
2
3 import java.io.File;
4
5
6
7
8
9
10
11
12
13
14
15
16
17 public class DriverSetup {
18     public static Properties prop;
19
20     public static WebDriver setupDriver() {
21         readConfig();
22         if(prop.getProperty("browser").equalsIgnoreCase("1")) { //CHROME
23             String exePath = Paths.get("").toAbsolutePath().toString() + File.separator + "Drivers" + File.separator;
24             System.setProperty("webdriver.chrome.driver", exePath + "chromedriver.exe");
25
26             ChromeOptions options = new ChromeOptions();
27             options.addArguments("--incognito");
28             options.addArguments("--ignore-certificate-errors");
29             DesiredCapabilities capabilities = DesiredCapabilities.chrome();
30             capabilities.setCapability(ChromeOptions.CAPABILITY, options);
31
32             WebDriver driver = new ChromeDriver(capabilities);
33             return driver;
34
35         } else if(prop.getProperty("browser").equalsIgnoreCase("2")) { //FIREFOX
36             String exePath = Paths.get("").toAbsolutePath().toString() + File.separator + "Drivers" + File.separator;
37             System.setProperty("webdriver.gecko.driver", exePath + "geckodriver.exe");
38             WebDriver driver = new FirefoxDriver();
39             return driver;
40         }
41     }
42     return setupDriver();
43 }
44
45 public static void readConfig() {
46     try {
47         prop = new Properties();
48         FileInputStream fs = new FileInputStream(System.getProperty("user.dir")+
49             "\\src\\test\\java\\config\\config.properties");
50         prop.load(fs);
51
52     } catch (FileNotFoundException e) {
53         e.printStackTrace();
54     } catch (IOException e) {
55         e.printStackTrace();
56     }
57 }
58
59 }
```

3.3.2 ClasePage.java

Pages Object: Patrones de diseño en automatización

El patrón Page Object nos ayuda con la tarea de robustecer nuestras pruebas. Consiste en crear un objeto por cada conjunto de elementos significativos de la interfaz con la que interactuamos. Aunque el nombre nos sugiera (y generalmente suceda) que cada objeto debe representar una página de nuestra aplicación, lo cierto es que si dentro de la página tenemos componentes visuales que reutilizamos en otras partes, podemos construir un Page Object de ese elemento para poder reutilizarlo.

Como lo aplicamos en el proyecto

Un ejemplo de esto sería el Login, ya que aparece en muchos de nuestros casos de prueba.

Dicho objeto de página debe exponer métodos que permitan al usuario que escribe la prueba hacer las cosas que pudiera hacer un humano interactuando con ese componente o página.

La regla de oro es: esconder detalles estructurales de la página, y expresar las cosas de manera más procedimental. Luego, dentro de este conjunto de métodos podemos trabajar con nuestro navegador y la estructura de la página para realizar lo que queremos.

En lugar de usar la sintaxis larga «driver.findElement(by.locator ())» cada vez que acceda a un elemento determinado, podemos crear instancias de un objeto WebElement.

Como se mencionó anteriormente, POM nos sirve como repositorio único de los elementos web de nuestro portal web.

Se crearán varias clases, las cuales contendrán estos elementos web, los cuales podremos utilizar en diferentes casos de prueba.

Estas clases, deberán ser creadas dentro de la carpeta **src/main/java** dentro de un Package llamado “**page**”.

Ejemplos:

- PageActualizarContraseñaOlvidada.java
- PageActualizarInformacion.java
- PageAdminstradorDominio.java
- PageAdminstradorPlataforma.java

- Etc.

```
▼ > SELENIUM ECLIPSE [portaldefirma_4_7_6 4_7_6]
  ▼ src/main/java
    > driverSetup
    ▼ pages
      > PageActualizarContraseniaOlvidada.java
      > PageActualizarInformacion.java
      > PageAdministradorDominio.java
      > PageAdministradorPlataforma.java
      > PageAdministrarFirmantes.java
      > pageAgenteAltaManual.java
      > pageAgenteAltaMasiva.java
```

La estructura básica de las clases que contendrán los métodos y los objetos, se conformarán de la siguiente manera:

Paso 1. Crear una clase llamada PageActualizarInformacion.

```
public class PageActualizarInformacion {
```

Paso 2. Inicializar Properties prop, siendo este un objeto de propiedades que nos proporciona un método .getProperty que toma la clave de la propiedad como parámetro y devuelve el valor de la clave coincidente de un archivo .properties.

```
public static Properties prop;
```

Paso 4. Esta clase, nos permite crear mensajes para el seguimiento o registro de la ejecución de la aplicación.

```
private static Logger Logger =
LoggerManager.getLogger(PageActualizarInformacion.class);
```

Paso 3. En este código, se está indicando el constructor, mismo que funcionará para inicializar el webDriver y pueda identificar los webElement. Es decir, cada que se mande a llamar la clase PageActualizarInformacion, el constructor va a realizar lo que está dentro del método.

```
public PageActualizarInformacion(WebDriver driver) {
    PageFactory.initElements(driver, this);
}
```

Paso 3. Inicializar una variable de tipo WebElement, tomando en cuenta la siguiente sintaxis:

```
@FindBy(id = "idEmailEmployee")
private WebElement campoEmail;
```

Se deberán inicializar las variables que sean necesarias según se requiera.

```
PageActualizarInformacion.java x
1 package pages;
2
3 import java.io.FileInputStream;
22
23 public class PageActualizarInformacion {
24     public static Properties prop;
25     private static Logger logger = LogManager.getLogger(PageActualizarInformacion.class);
26
27     private static ExtentHtmlReporter htmlReporter = new ExtentHtmlReporter("extent.html");
28     private static ExtentReports extent = new ExtentReports();
29
30     public PageActualizarInformacion(WebDriver driver) {
31         PageFactory.initElements(driver, this);
32     }
33
34     @FindBy(id = "idEmailEmployee")
35     private WebElement campoEmail;
36
```

Como localizar los Web Element

Una vez identificados los casos de prueba que se desean automatizar, se crearan las clases a donde contengan los webs element de la página web que se está automatizando.

- Un WebElement representa un elemento del DOM (Document Object Model). Los WebElements se pueden encontrar buscando desde la raíz del documento utilizando una instancia de WebDriver o buscando en otra WebElement.
- En pocas palabras, es la forma de identificar elementos en una página.

Localizar Web Element

Un localizador es una forma de identificar elementos en una página. Es el argumento pasado a los métodos del elemento Finding.

Localizadores tradicionales:

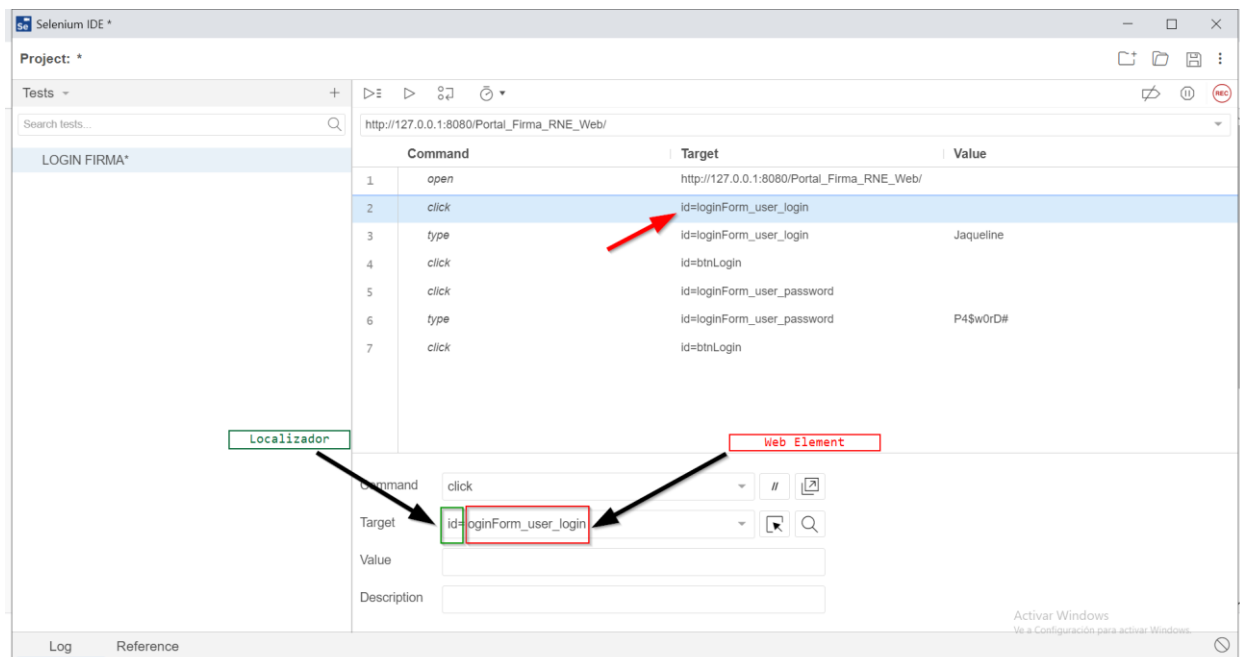
Localizador	Descripción
xpath	Localiza elementos que coinciden con una expresión XPath.
class name	Localiza elementos cuyo nombre de clase contiene el valor de búsqueda (no se permiten nombres de clase compuestos).
css selector	Localiza elementos que coinciden con un selector CSS.
id	Localiza elementos cuyo atributo ID coincide con el valor de búsqueda.

name	Localiza elementos cuyo atributo NOMBRE coincide con el valor de búsqueda.
link text	Localiza elementos de anclaje cuyo texto visible coincide con el valor de búsqueda.
partial link text	Localiza elementos de anclaje cuyo texto visible contiene el valor de búsqueda. Si coinciden varios elementos, solo se seleccionará el primero.
tag name	Localiza elementos cuyo nombre de etiqueta coincide con el valor de búsqueda

Para poder localizar de forma rápida un elemento web, en su mayoría, se utilizará la extensión Selenium IDE.

Una vez que se graba el test, se puede observar que, al dar clic en algún elemento web, como resultado se obtiene el localizador de ese elemento y el elemento, conformándose, por ejemplo, de la siguiente manera:

id=loginForm_user_Login

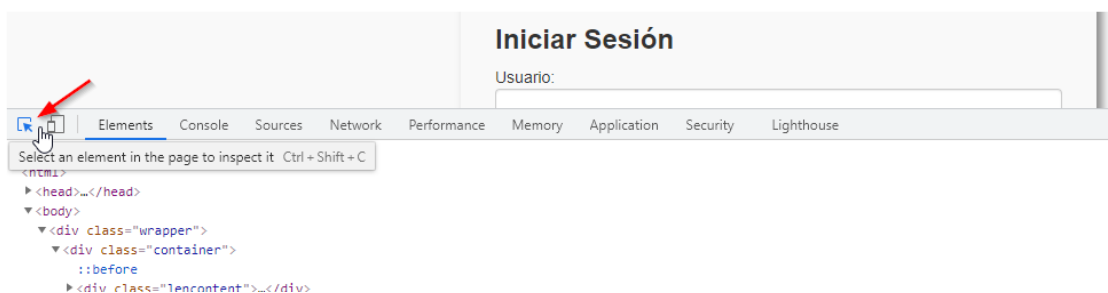


De igual forma, también se pueden identificar estos elementos directamente en el navegador de la siguiente manera:

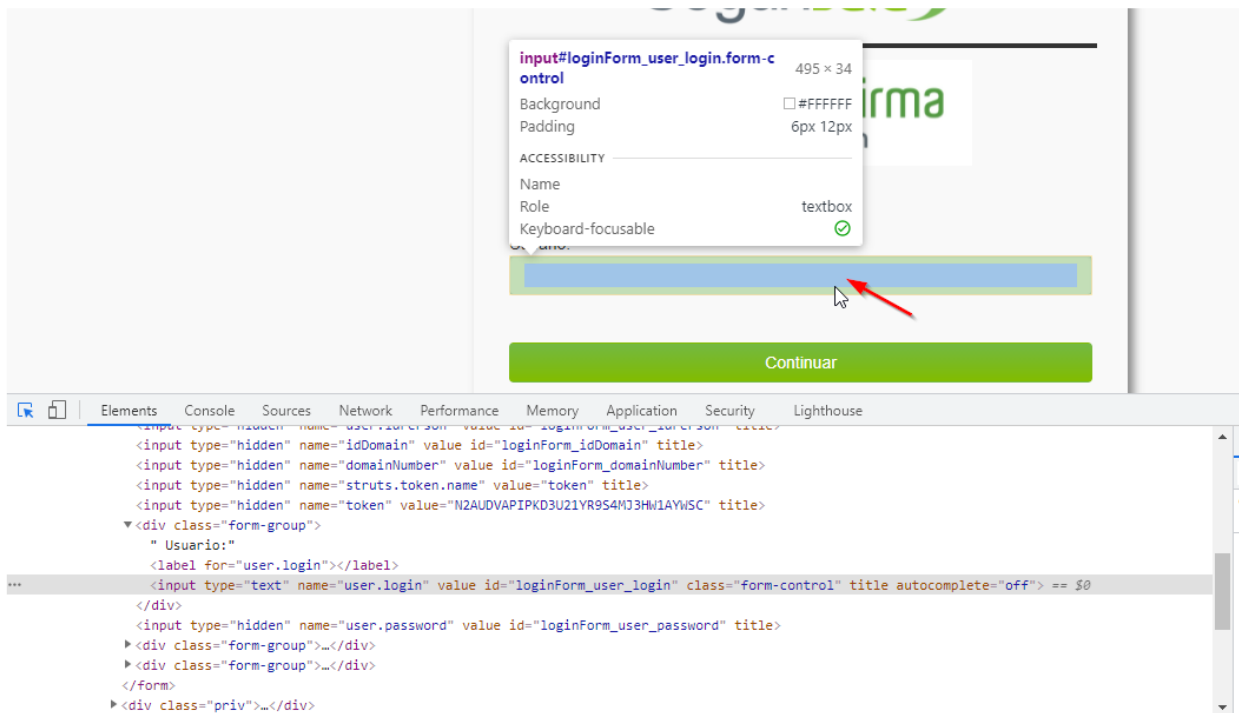
Paso 1. Ubicarse en la página que se estará probando, dar clic derecho y seleccionar la opción Inspeccionar.



Paso 2. Dar clic en el icono



Paso 3. Seleccionar el elemento que se desea inspeccionar, en este caso, el campo para ingresar el Usuario.



Al final, se remarca la línea donde se encuentra el elemento. En este caso, el elemento que se está buscando es el que se encuentra dentro de "id".



Crear métodos para los casos de pruebas

Dentro de las clases clasePageN.java, se deben incluir los métodos para cada caso de prueba. Estos se compondrán de los webElements que se identifiquen según la necesidad.

A continuación, se muestra un ejemplo.

Método: PageLogin.java

Este método, nos ayudará a iniciar sesión en muchos de los casos de prueba.

```
118 public void autenticacion(WebDriver driver, String user, String pass) throws InterruptedException {
119     readConfig();
120     if (prop.getProperty("tipoAutenticacion").equalsIgnoreCase("normal")) {
121         Logger.info("Se realiza Login Normal.");
122         fieldNombreElement.sendKeys(user);
123         botonEntrarField.click();
124         fieldpasswordElement.sendKeys(pass);
125         botonEntrarField.click();
126
127         if (driver.findElements(By.xpath("//*[@id=\"content\"]/div/h1")).size() > 0) {
128             String usuario = userLogeado.getText();
129             Logger.info("Se autentica el siguiente usuario: " + usuario);
130         } else {
131             if (driver.findElements(By.xpath("//button[@type='button']")).size() > 0) {
132                 driver.findElement(By.xpath("//button[@type='button']")).click();
133                 String usuario = userLogeado.getText();
134                 Logger.info("Se autentica el siguiente usuario: " + usuario);
135             } else {
136                 fieldpasswordElement.sendKeys(pass);
137                 botonEntrarField.click();
138                 String usuario = userLogeado.getText();
139                 Logger.info("Se autentica el siguiente usuario: " + usuario);
140             }
141         }
142     }
143
144     if (prop.getProperty("tipoAutenticacion").equalsIgnoreCase("dobleA")) {
145         Logger.info("Se realiza Login 2FA.");
146         fieldNombreElement.sendKeys(user);
147         botonEntrarField.click();
148         fieldpasswordElement.sendKeys(pass);
149         botonEntrarField.click();
150         Thread.sleep(5000);
151
152         readConfigMensaje();
153         String actual = textDialogoParaIngresarCodigoGoogle.getText();
154         String expected = prop.getProperty("Dialogo_Mensaje_De_Autenticacion");
155
156         if (expected.equals(actual)) {
157         } else {
158             Logger.error("El mensaje de validacion es diferente al esperado. " + "\nMensaje actual: " + e
159                 + "Mensaje esperado: " + actual);
160         }
161
162         readConfigMensaje();
163         String actual2 = validarUsuarioLogin.getText();
164         Logger.info("Se autentica el siguiente usuario: " + actual2);
165     }
166 }
```

```

167         if (prop.getProperty("tipoAutenticacion").equalsIgnoreCase("siemens")) {
168             Logger.info("Realizar Login para SIEMENS.");
169             readConfig();
170             fieldNombreElement.sendKeys(user);
171             botonEntrarField.click();
172             fieldpasswordElement.sendKeys(pass);
173             botonEntrarField.click();
174
175             if (driver.findElements(By.xpath("//div[@id='nonSiemensFederationServices']/div/div/div")).size()
176                 Logger.info("Se muestra Portal de SIEMENS.");
177                 Thread.sleep(3000);
178             } else if (driver.findElements(By.xpath("//button[@type='button']")[3])).size() > 0 {
179                 driver.findElement(By.xpath("//button[@type='button']")[3]).click();
180                 Logger.info("La sesión se encuentra caducada ");
181             }
182
183             customerOrPartner.click();
184             siemensID.click();
185             yesSiemens.click();
186             emailSiemens.sendKeys("pruebasjaqueline@gmail.com");
187             passwordSiemens.sendKeys("12121212Qw.");
188             botonEntrarSiemens.click();
189             codigo2FASiemens.click();
190             WebDriverWait wait = new WebDriverWait(driver, 20);
191             WebElement element = wait
192                 .until(ExpectedConditions.elementToBeClickable(By.xpath("//div[@id='content']/div/h1")));
193             element.click();
194             String usuario = userLogeado.getText();
195             Logger.info("Se autentica el siguiente usuario: " + usuario);
196         }
197     }

```

A continuación, se describe parte del código.

Entre las principales características de este método, es que se tiene la opción de iniciar sesión de acuerdo al tipo de configuración que ese encuentre en el archivo properties de configuración, es decir, se ejecutará la prueba de acuerdo a los valores que se encuentren en la property “tipoAutenticacion” ya sea “normal”, “dobleA” o “siemens”, si la configuración se encuentra en “normal”.

```

---
231
232#INDICAR TIPO DE AUTENTICACIÓN PARA INGRESAR AL PORTAL
233#Inicio de sesión simple = [normal]
234#Inicio de sesión 2fa = [dobleA]
235#Inicio de sesión 2fa-siemens = [siemens]
236tipoAutenticacion = normal

```

Properties de archivo config.properties

Si la configuración su encuentra en “normal”, solamente se ejecutará el código que este dentro de esa condición.

```
if (prop.getProperty("tipoAutenticacion").equalsIgnoreCase("normal")) {  
}
```

Importante, para poder obtener la información de las propiedades, se manda a llamar el método “readConfig()”. Que se encuentra en la misma clase (Por ejemplo, clasePage.java).

```
376  
377 public void readConfig() {  
378     try {  
379         prop = new Properties();  
380         FileInputStream fs = new FileInputStream(  
381             System.getProperty("user.dir") + "\\src\\test\\java\\config\\config.properties");  
382  
383         prop.load(fs);  
384     } catch (FileNotFoundException e) {  
385         e.printStackTrace();  
386     } catch (IOException e) {  
387         e.printStackTrace();  
388     }  
389 }  
390  
---
```

Como se observa en el código, al cumplirse la condición de que el tipo de autenticación es normal, se ejecutará al menos el siguiente bloque de código, donde se observa lo siguiente:

```
public void autenticacion(WebDriver driver, String user, String pass) throws InterruptedException {  
    readConfig();  
    if (prop.getProperty("tipoAutenticacion").equalsIgnoreCase("normal")) {  
        Logger.info("Se realiza Login Normal.");  
        fieldNombreElement.sendKeys(user);  
        botonEntrarField.click();  
        fieldpasswordElement.sendKeys(pass);  
        botonEntrarField.click();  
    }  
}
```

Al ser el tipo de autenticación normal, se imprime a nivel Log el siguiente mensaje de tipo INFO:

```
Logger.info("Se realiza Login Normal.");
```

También, se empiezan a interactuar en el navegador mediante las variables declaradas en un inicio, haciendo la llamada únicamente del objeto.

En caso de que sea un formulario, se utiliza el método `sendKeys`, mismo que se utiliza para escribir contenido automáticamente en un campo editable.

```
fieldNombreElement.sendKeys(user);
```

En caso de que sean botones, o algún otro elemento no editable, se utiliza el método `click()`, siendo este un método que emula una operación de clic del mouse para cualquier elemento interactivo de la interfaz de usuario (enlaces, botones, casillas de verificación, botones de opción).

```
botonEntrarField.click();
```

Se incluyeron algunas validaciones que nos ayudan a determinar el camino que se debe tomar en caso de que se encuentre algo diferente a lo esperado, esto mediante la localización de elementos mientras se van ejecutando las pruebas.

Por ejemplo

```
// SI SE INICIA SESIÓN Y SE ENCUENTRA EL ENCABEZADO PRINCIPAL, SE MOSTRARÁ
MENSAJE DE ÉXITO.
if (driver.findElements(By.xpath("//*[@id=\"content\"]/div/h1")).size() > 0) {
    String usuario = userLogeado.getText();
    Logger.info("Se autentica el siguiente usuario: " + usuario);

// SI SE INICIA SESIÓN Y MUESTRA EL BOTON PARA CERRAR LA SESIÓN ANTERIOR E
INICAR UNA NUEVA, YA QUE SE ENCUENTRA ABIERTA EN OTRO NAVEGADOR, LO QUE
SE REALIZARÁ ES DAR CLIC EN ESE BOTON, Y DESPUÉS, YA SE AUTENTICARÁ DE
FORMA CORRECTA, IMPRIMIENDO EL MENSAJE DE ÉXITO.
} else {
    if
(driver.findElements(By.xpath("//button[@type='button']")[3])).size()>0)
    {
        driver.findElement(By.xpath("//button[@type='button']")[3]).click
        ();
        String usuario = userLogeado.getText();
        Logger.info("Se autentica el siguiente usuario: " + usuario);
    }
}
```

Código:

```
127     if (driver.findElements(By.xpath("//*[@id=\"content\"]/div/h1")).size() > 0) {
128         String usuario = userLogeado.getText();
129         Logger.info("Se autentica el siguiente usuario: " + usuario);
130     } else {
131         if (driver.findElements(By.xpath("//button[@type='button']")[3])).size() > 0) {
132             driver.findElement(By.xpath("//button[@type='button']")[3]).click();
133             String usuario = userLogeado.getText();
134             Logger.info("Se autentica el siguiente usuario: " + usuario);
135         }
136     }
```

Otro ejemplo práctico que se implementó, y que nos ayuda para determinar si la prueba es correcta o no, es el uso de validaciones que nos ayudan a determinar si el mensaje recibido es el mismo que el mensaje actual.

Por ejemplo, se tiene el siguiente método “FirmaDocumentoPerfil4”.

Después de ingresar la contraseña para firmar el documento y al dar clic en “botonFirmar”, se espera un mensaje de validación al concluir la firma, el cual, se obtiene de la siguiente manera:

```
readConfigMensaje();
String actual = textDialogo.getText();
String expected = prop.getProperty("Dialogo_Firma_Documento");
```

Dónde, en la variable “actual”, se almacena el texto que se obtiene del elemento textDialog, y en la variable “expected” se asigna el valor que se encuentra en la property “Dialogo_Firma_Documento”, el cual, contiene el mensaje esperado.

Una vez que se tienen esos valores en las variables, se lanza una validación que nos indica que, si el mensaje esperado es igual al mensaje actual, se imprimirá en el Log un mensaje de éxito de tipo “info” indicando que el mensaje de validación es correcto, de otra forma, se imprimirá en el Log un mensaje de tipo “error”, el cual nos indica que el mensaje actual es diferente al mensaje actual, por lo tanto, el mensaje es incorrecto, por lo tanto, la prueba va a fallar.

```
if (expected.equals(actual)) {
```

Se comparte código completo:

```
157 public void FirmaDocumentoPerfil4(WebDriver driver) throws InterruptedException{
158     readConfig();
159     Logger.info("Firmar documentos perfil 4.");
160     WebDriverWait wait = new WebDriverWait(driver, 20);
161     WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("idShowPassword")));
162     element.click();
163     ingresarClavePrivada.sendKeys(prop.getProperty("Clave_Llave_Privada"));
164     botonFirmar.click();
165
166     readConfigMensaje();
167     String actual = textDialogo.getText();
168     String expected = prop.getProperty("Dialogo_Firma_Documento");
169
170     if (expected.equals(actual)) {
171         Logger.info("El mensaje de validacion es correcto: "+actual) ;
172     } else {
173         Logger.error("El mensaje de validacion es diferente al esperado.");
174         Logger.error("Mensaje actual: "+actual);
175         Logger.error("Mensaje esperado: "+expected);
176     }
177
178     botonOK.click();
179 }
```

3.3.3 EscenariosPruebas.java

Agruparemos los casos de prueba en escenarios de pruebas, es decir, podremos escribir varios casos de prueba en un solo escenario.

Estos escenarios de prueba, deberán ser creadas dentro de la carpeta **src/main/test**, agrupadas en un Package según corresponda.

En nuestro proyecto, tenemos los siguientes paquetes:

- Usuarios Perfil 4 (Portal de firma)
- Usuarios Perfil 8 (Portal de firma)
- Usuarios Perfil 9 (Portal de firma)
- Usuarios Perfil 13 (Portal de firma)
- Portal Administrador de dominio (Portal Administración)
- Portal Administrador de plataforma (Portal Administración)
- Portal de Agente

```
▼ [icon] > SELENIUM ECLIPSE [portaldefirma_4_7_6 4_7_6]
  > [icon] src/main/java
  > [icon] src/main/resources
  ▼ [icon] src/test/java
    > [icon] config
    > [icon] files
    > [icon] files.altaArchivos
    > [icon] perfil13Test
    > [icon] perfil4Test
    > [icon] perfil8Test
    > [icon] perfil9Test
    > [icon] portalAdministradorDominio
    > [icon] portalAdministradorPlataforma
    > [icon] portalAgente
    > [icon] test
  > [icon] JRE System Library [JavaSE-1.7]
  > [icon] Maven Dependencies
  > [icon] Drivers
  > [icon] logs
  > [icon] src
    > [icon] target
  > [icon] > test-output
    [icon] pom.xml
```

Y en cada paquete, se agregan los diferentes escenarios de prueba:

```
SELENIUM ECLIPSE [portaldefirma_4_7_6 4_7_6]
├── src/main/java
├── src/main/resources
├── src/test/java
│   ├── config
│   ├── files
│   ├── files.altaArchivos
│   ├── perfil13Test
│   └── perfil4Test
│       ├── baseTest.java
│       ├── TC01_P4_pruebasLogin.java
│       ├── TC02_P4_pruebasActualizarInformacion.java
│       ├── TC03_P4_pruebasBloqueoDesbloqueo.java
│       ├── TC04_P4_pruebasPrecertificacion.java
│       ├── TC05_P4_pruebasConsultaBandejas.java
│       ├── TC06_P4_pruebasSubirDocumentos.java
│       ├── TC07_P4_pruebasAgregarInvitados.java
│       └── TC08_P4_pruebasFirmaDocumentos.java
```

3.3.4 CasosPrueba.class

Se tomará como ejemplo el escenario de prueba “TC01_P4_pruebasLogin”, en el cual, se incluyen casos de prueba principalmente del Login e inicio de sesión.

A continuación, se describe algunas consideraciones que se deben tomar en cuenta y que se debe realizar.

PASO 1. Se debe indicar el paquete a donde se encuentra este escenario de prueba, así como también importar las librerías correspondientes. Nota: Estas librerías se pueden ir agregando conforme se vaya avanzando.

```
TC01_P4_pruebasLogin.java ×
1 package perfil4Test;
2
3 import java.io.FileInputStream;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6 import java.util.Properties;
7 import java.util.concurrent.TimeUnit;
8
9 import org.apache.logging.log4j.LogManager;
10 import org.apache.logging.log4j.Logger;
11 import org.openqa.selenium.WebDriver;
12 import org.testng.annotations.AfterTest;
13 import org.testng.annotations.BeforeTest;
14 import org.testng.annotations.Ignore;
15 import org.testng.annotations.Test;
16
17 import driverSetup.DriverSetup;
18 import pages.PageLogin;
19 import pages.pageCasosDePruebaFirma;
20
```

Paso 2. Dentro del método de la clase “TC01_P4_pruebasLogin”, se inicializa el método llamado DriverSetup (creado anteriormente en este documento [\(DriverSetup.java\)](#)).

Paso 3. Inicializar Properties prop, siendo este un objeto de propiedades que nos proporciona un método .getProperty que toma la clave de la propiedad como parámetro y devuelve el valor de la clave coincidente de un archivo .properties.

```
public static Properties prop;
```

```
Run All
3 public class TC01_P4_pruebasLogin extends baseTest {
3     static WebDriver driver = DriverSetup.setupDriver();
1     public static Properties prop;
2 }
```

Paso 4. Se crea el método setUpTest, el cual nos indica la URL en la que vamos a ejecutar las pruebas, va a maximizar la pestaña del navegador y se tendrá una espera implícita de 5 segundos.

```
22
23 @BeforeTest
24 public void setUpTest() throws InterruptedException {
25     readConfig();
26     driver.get(prop.getProperty("url"));
27     driver.manage().window().maximize();
28     driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
29 }
30
```

Para entender un poco el concepto de las esperas implícita, se describe lo siguiente:

Esperas implícitas

La espera implícita es el tiempo por defecto que se establece al inicio para buscar cualquier componente.

Por ejemplo, aquí se establece una espera implícita de 5 segundos:

```
driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
```

Por consiguiente, Selenium esperará 5 segundos cuando vaya a buscar cualquier elemento y en caso de no encontrarlo, se lanzará una excepción de tipo “Elemento no encontrado”.

Antes de continuar, se describe el siguiente concepto de TestNG, ya que a lo largo de estas pruebas, se observa varias anotaciones mismas de TestNG.

TestNG

Las anotaciones en TestNG son líneas de código que pueden controlar cómo se ejecutará el método debajo de ellos. Siempre están precedidas por el símbolo @.

Las anotaciones de uso común se enumeran a continuación:

- @BeforeSuite
- @AfterSuite
- @BeforeTest
- @AfterTest
- @BeforeClass
- @AfterClass
- @BeforeMethod
- @AfterMethod

En nuestro proyecto, utilizamos las siguientes:

@BeforeTest

Los métodos anotados con @BeforeTest se ejecutarán antes de que se ejecuten todos los métodos de prueba definidos en un elemento.

@AfterTest

Los métodos anotados con @AfterTest se ejecutarán después de que se ejecuten todos los métodos de prueba definidos en un elemento.

@Test

Los métodos anotados con @Test, se ejecutarán después de BeforeTest y antes de @AfterTest.

Para ejecutar los casos de prueba según un nivel de prioridad, se utiliza la siguiente anotación:

@Test (priority = 1)

Si se quiere ignorar un caso de prueba, se utiliza la siguiente anotación:

@Test

@Ignore

Para correr un caso de prueba o todos, dar clic en la opción correspondiente

```

Run All
3 public class TC01_P4_pruebasLogin extends baseTest {
4     static WebDriver driver = DriverSetup.setupDriver();
5     public static Properties prop;
6
7     @BeforeTest
8     public void setUpTest() throws InterruptedException {
9         readConfig();
10        driver.get(prop.getProperty("url"));
11        driver.manage().window().maximize();
12        driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
13    }
14
15    @Test(priority = 1)
16    Run | Debug
17    public static void loginPerfil4() throws InterruptedException {
18        test = extent.createTest("LoginPerfil4");
19        LoginPerfil4TC.loginPerfil4TC(driver);
20        autenticacion.autenticacion(driver, prop.getProperty("Usuario_Perfil_4"), prop.getProp
21        Logout.logout(driver);
22    }
23
24    @Test
25    @Ignore
26    Run | Debug
27    public static void loginUnicamente() throws InterruptedException {
28        test = extent.createTest("LoginPerfil4");
29        LoginPerfil4TC.loginPerfil4TC(driver);
30        // loginUnicamente.loginUnicamente(driver, prop.getProperty("Usuario_Perfil_4"), prop.get
31        autenticacion.autenticacion(driver, prop.getProperty("Usuario_Perfil_4"), prop.getProp
32        Logout.logout(driver);
33    }
34 }

```

Una vez aclarado esto, se crean los casos de prueba, los cuales incluyen diferentes objetos de los métodos construidos en las ClassPages, así como el llamado a la librería extentReport para que se pueda generar un reporte de las pruebas.

En este ejemplo, se hace el llamado a 3 métodos:

1. Método: loginPerfil4TC, incluye el nombre del caso de prueba
2. Método: autenticación, incluye la autenticación del usuario
3. Método: logOut, incluye la opción de cerrar sesión.

Quedando de la siguiente manera:

```

30
31 @Test(priority = 1)
32 Run | Debug
33 public static void loginPerfil4() throws InterruptedException {
34     test = extent.createTest("LoginPerfil4");
35     LoginPerfil4TC.loginPerfil4TC(driver);
36     autenticacion.autenticacion(driver, prop.getProperty("Usuario_Perfil_4"), prop.getF
37     Logout.logout(driver);
38 }

```

3.4 Complementación de las pruebas.

Identificar si algún caso de prueba necesita algún complemento especial para poder ser ejecutado con éxito.

En nuestro proyecto, no encontramos ningún inconveniente para poder automatizar las pruebas. Todo fue posible.

3.5 Ejecución y Depuración.

Se ejecutarán los casos de prueba y se encontraran errores en el sistema, después estos deben ser reportados al desarrollador correspondiente y después de la corrección, se ejecutará nuevamente el set de pruebas para asegurarse que todos los errores encontrados fueron depurados.

Siguiendo con el ejemplo del Login,

```
30
31  @Test(priority = 1)
    Run | Debug
32  public static void loginPerfil4() throws InterruptedException {
33      test = extent.createTest("LoginPerfil4");
34      loginPerfil4TC.loginPerfil4TC(driver);
35      autenticacion.autenticacion(driver, prop.getProperty("Usuario_Perfil_4"), prop.getPrope
36      Logout.logout(driver);
37  }
```

al ejecutar ese test, el navegador realiza lo siguiente:

1. Abrir un navegador según configuración: Chrome o Firefox
2. Se ingresa a la URL y se maximiza la pestaña.
3. Se generará reporte html del test
4. Se imprime en el Log el nombre del caso de prueba.
5. Se realiza la autenticación según configuración: normal, 2fa o siemens.
6. Se realiza Logout

3.5.1 Resultados de la prueba

Una vez ejecutado la prueba, se genera lo siguiente en consola, indicando que la prueba ha pasado, a su vez que va imprimiendo los mensajes de tipo LOG que fueron declarados en los métodos de las clases. Estos mensajes se van trazando también con la fecha y hora de ejecución.

Nos indica que se ha ejecutado 1 test, y no hubo errores.

```
Console x Results of running method TC01_P4_pruebasLogin.loginPerfil4 Problems Javadoc Declaration
<terminated> TC01_P4_pruebasLogin.loginPerfil4 [TestNG] C:\Users\bastida\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-
[RemoteTestNG] detected TestNG version 7.5.0
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
feb. 21, 2023 4:43:40 P. M. org.openqa.selenium.remote.DesiredCapabilities chrome
INFORMACIÓN: Using `new ChromeOptions()` is preferred to `DesiredCapabilities.chrome()`
Starting ChromeDriver 110.0.5481.77 (65ed616c6e8ee3fe0ad64fe83796c020644d42af-refs/branch-heads/5481@{#839}) on port 435
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
[1677019421.479][WARNING]: virtual void DevToolsClientImpl::AddListener(DevToolsEventListener *) subscribing a listener
feb. 21, 2023 4:43:41 P. M. org.openqa.selenium.remote.ProtocolHandshake createSession
INFORMACIÓN: Detected dialect: W3C
[INFO ] 2023-02-21 16:43:42 [main] pageCasosDePruebaFirma - -----
[INFO ] 2023-02-21 16:43:42 [main] pageCasosDePruebaFirma - Caso de prueba: loginPerfil4
[INFO ] 2023-02-21 16:43:42 [main] PageLogin - Realizar Login para SIEMENS.
[INFO ] 2023-02-21 16:43:48 [main] PageLogin - Se muestra Portal de SIEMENS.
[INFO ] 2023-02-21 16:44:24 [main] PageLogin - Se autentica el siguiente usuario: usuario_jcb1_p4@seguridata.com
[INFO ] 2023-02-21 16:44:25 [main] PageLogin - Se realiza logout.
PASSED: loginPerfil4

=====
Default test
Tests run: 1, Failures: 0, Skips: 0
=====

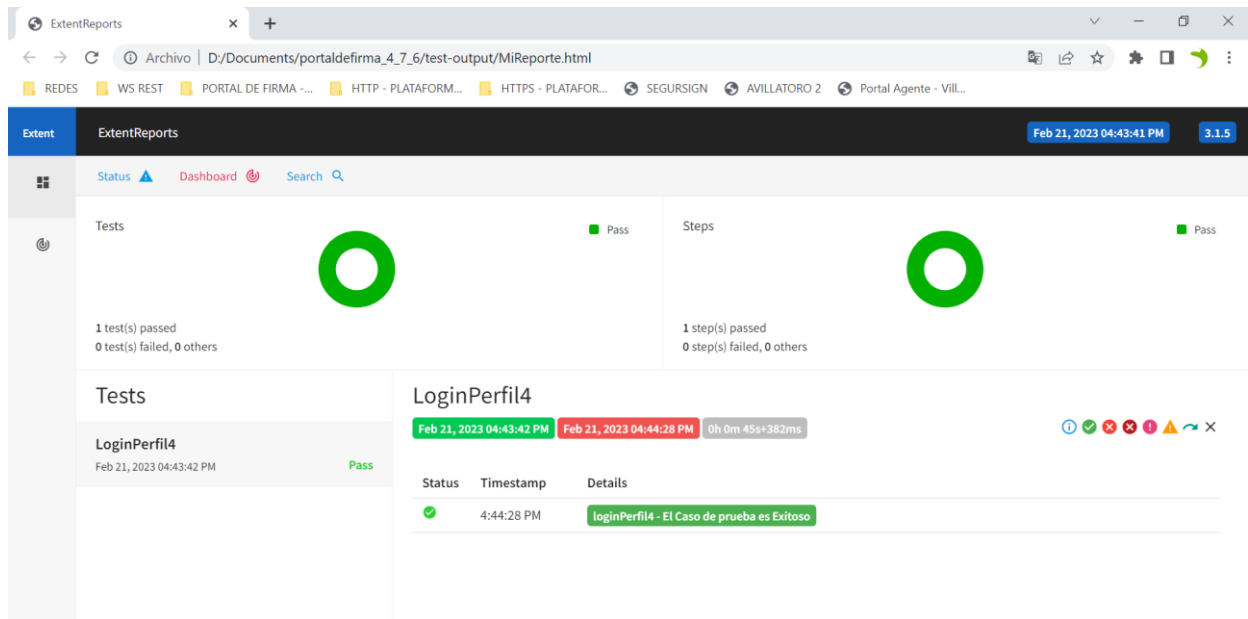
=====
Default suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
=====
```

En el reporte de TestNG, también se muestra el resultado indicando el test o los test que se han ejecutado, y que tiempo tardo cada uno en ejecutarse.

```
Console x Results of running method TC01_P4_pruebasLogin.loginPerfil4 x
Search:
Passed: 1
Failed: 0

All Tests Failed Tests Summary
▼ Default suite ( 1/0/0/0 ) (45.377 s)
  ▼ Default test ( 45.377 s)
    ▼ perfil4Test.TC01_P4_pruebasLogin
      loginPerfil4 (45.377 s)
```

A su vez, también se genera el reporte HTML de la librería Extent Report, el cual muestra el o los casos de prueba ejecutados, el tiempo que tardo cada test en ejecutarse, así como su resultado individual, si pasó o falló el test.



Se busca desarrollar un sistema de pruebas con la herramienta de automatización Selenium IDE y Selenium WebDriver para realizar los casos de prueba establecidos en la matriz de pruebas para los productos de software de la empresa SeguriData Privada SA. DE C.V denominados Portal de Firma SeguriSign, Portal de Administración SeguriSign y Portal de Agente SeguriSign.

La automatización de las pruebas a abarcado el 100%,

CONCLUSIONES

4.1 Conclusión

Durante el desarrollo del proyecto se analizó la situación actual de la organización SeguriData, identificando los procesos del departamento de Control de Calidad (QA), llegando a la conclusión que, QA deberá participar desde el nacimiento de una idea hasta la implementación del software, apoyando en todo el ciclo de vida del producto, ya que el control de calidad no debe limitarse solamente a la ejecución de Pruebas.

El conjunto de prácticas DevOps, enfatizan la colaboración, comunicación e integración entre los desarrolladores de software y los profesionales de operaciones de TI, incluyendo a los Tester, encargados de la ejecución de pruebas con herramientas de automatización, agilizando el proceso de entrega de software. DevOps tiene como objetivo crear una cultura entre los colaboradores y un entorno en el que el diseño, las pruebas y el lanzamiento de software se puedan realizar de forma rápida, frecuente y eficiente, de mejor calidad y a un coste menor

Para la implementación de prácticas DevOps, se realizó una investigación de herramientas para la automatización de pruebas, siendo Selenium, la herramienta elegida para comenzar las pruebas automáticas en el producto de la empresa “Vertical SeguriSign”.

Selenium es una herramienta Open Source, que permite ejecutar las pruebas en diferentes ambientes y plataformas, ofreciendo una solución completa para conseguir los objetivos planteados a medio y largo plazo.

4.2 Recomendación

La automatización de pruebas, nos garantizan pruebas fiables y rápidas, reduciendo la inestabilidad frente a futuros errores en desarrollo y facilitando la integración de nuevos test a medida que el proyecto y la inserción de nuevos requerimientos avanzan en el tiempo.

En mi opinión como empleado de la empresa con el perfil de Tester, y actualmente como Desarrollador Jr. con el perfil de QA Automation de pruebas,

las pruebas automatizadas han ayudado a reducir considerablemente el tiempo de ejecución de las pruebas, así como la detección de errores en una etapa de desarrollo temprano, ayudando así, a mejorar los tiempos de entrega, y lo más importante, a garantizar una mejor calidad del producto con el cliente.

COMPETENCIAS DESARROLLADAS Y/O APLICADAS

Entre las competencias y habilidades desarrolladas a lo largo de la estancia en la empresa, fungiendo como Tester en el área de calidad, y comunicándome siempre con los Desarrolladores de software, se encuentran las siguientes:

- Identificar errores que podrían ocurrir en el desarrollo de software
- Habilidad para evaluar software
- Habilidad para identificar riesgos en el desarrollo de software
- Habilidad para resolver problemas reales
- Listar errores y redacción de informes de calidad
- Llevar registros precisos de lo que se ha testado
- Trabajo en equipo
- Comunicación Oral y Escrita

Mientras que las competencias aplicadas, se encuentran las siguientes:

- Análisis, Diseño, Desarrollo e implementación de sistemas de información para la gestión de procesos y apoyo en la toma de decisiones, utilizando metodologías basadas en estándares internacionales.
- He desempeñado las actividades profesionales considerando los aspectos legales, éticos, sociales y de desarrollo sustentable.

FUENTES DE INFORMACIÓN.

Colaboradores de Wikipedia. (2020, 22 mayo). *Selenium*. Wikipedia, la enciclopedia libre. <https://es.wikipedia.org/wiki/Selenium>

Quijano, J. (2018a, octubre 8). *¿Qué pruebas debemos hacerle a nuestro software y para qué?* Genbeta. <https://www.genbeta.com/desarrollo/que-pruebas-debemos-hacerle-a-nuestro-software-y-para-que>

Sargent, A. (2020, 25 junio). *16 reasons why to use Selenium IDE in 2019 (and 2 why not)*. Automated Visual Testing | Applitools. https://applitools.com/blog/whyselenium-ide-2019/?utm_referrer=https%3A%2F%2Fwww.google.com%2F

Selenium IDE · Open source record and playback test automation for the web. (s. f.). Selenium. <https://www.selenium.dev/selenium-ide/>

Selenium Tutorial for Beginners: Learn WebDriver in 7 Days. (2020, 13 mayo). Guru. <https://www.guru99.com/selenium-tutorial.html>

¿Qué es DevOps? Explicación de DevOps | Microsoft Azure. (2018). Microsoft Azure. <https://azure.microsoft.com/es-mx/overview/what-is-devops/#tools>

Sin automatización no hay DevOps. (2019). Digital Business Assurance. <https://www.mtp.es/blog/devops-agile/sin-automatizacion-no-hay-devops>