AUTOMATIZACIÓN DE CASOS DE PRUEBA PARA LA APLICACIÓN "INDEMNIZACIONES"

Jorge Leonardo Chaparro Sarmiento.

Universidad Santo Tomás, Seccional Tunja.
Facultad de Ingeniería Electrónica.
IBM de Colombia. GBS. TAC-CM-QA.
2020

AUTOMATIZACIÓN DE CASOS DE PRUEBA PARA LA APLICACIÓN "INDEMNIZACIONES"

Jorge Leonardo Chaparro Sarmiento.

Lectores/Jurados: Ing. Camilo Ernesto Pardo Beainy.
Ing. Daniel Alejandro Rodríguez Caro.

Tutora: Ing. Angélica María Salazar Madrigal.
Cotutor: Ing. Carlos Alberto Cardona Coy.

Universidad Santo Tomás, Seccional Tunja.
Facultad de Ingeniería Electrónica.
IBM de Colombia. GBS. TAC-CM-QA.
2020

EXONERACIÓN DE RESPONSABILIDADES.

Este apartado se desarrolla con el propósito de exonerar a la Universidad Santo Tomás de Aquino, Seccional Tunja, de todas las responsabilidades y daños que se pueda causar a la empresa IBM por el proyecto desarrollado durante la pasantía, realizada durante el periodo de julio a diciembre de 2019, por el estudiante Jorge Leonardo Chaparro Sarmiento, identificado con el documento C.C. 1049655201 de la ciudad de Tunja.

ACEPTACIÓN DEL PROYECTO.

_____	_____
Jurado 1	Jurado 2
Ing. Camilo Ernesto Pardo Beainy.	Ing. Daniel Alejandro Rodríguez Caro.
Facultad de Ingeniería Electrónica.	Facultad de Ingeniería Electrónica.
Universidad Santo Tomás, Seccional Tunja.	Universidad Santo Tomás, Seccional Tunja.

TABLA DE CONTENIDOS.

1. INTRODUCCIÓN

La etapa de pruebas de calidad en el desarrollo de aplicaciones web es sumamente importante, ya que asegura que el producto que se entregará cumpla con los requerimientos que se exigieron desde un comienzo y contemple las modificaciones que poco a poco van surgiendo.

Las pruebas de calidad se basan en la ejecución de casos de prueba, los cuales son el conjunto de condiciones que se deben cumplir para generar determinado resultado, acorde con los requerimientos del proyecto.

Los avances tecnológicos permitieron desarrollar una estrategia que simplifica en gran medida las pruebas de calidad; dicha estrategia recibió el nombre de Automatización de Testing, la cual se basa en scripts debidamente programados que ejecutarán los casos de prueba, mientras que disminuyen los recursos económicos, humanos y de tiempo.

La Automatización de Testing resultó ser bastante útil en las pruebas de regresión, éstas están compuestas por algunos casos de prueba específicos que se deben ejecutar con cada actualización que se le haga al proyecto, con el fin de verificar que las funcionalidades principales de la aplicación se mantengan intactas.

En el presente documento se hace énfasis en el funcionamiento de *Indemnizaciones*: una aplicación desarrollada por la empresa IBM, explicando detalladamente el proceso de formulación y ejecución de sus casos de prueba, y aún más importante, ya que es el enfoque principal del proyecto: la automatización de la ejecución

de tales casos de prueba, además de la sustentación del impacto que el desarrollo del proyecto tendrá para la empresa.

2. PLANTEAMIENTO DEL PROBLEMA

En IBM, el proyecto "*Seguros Bolívar*" ha estado en desarrollo por más de tres años. En la actualidad, las aplicaciones que componen el proyecto (*ARL*, *Venta Digital de Autos*, *SOAT Digital*, *FileNet*, *Oficina Virtual* e *Indemnizaciones*) siguen sujetas a mejoras y cambios que podrían llegar a afectar su funcionalidad; por lo que constantemente tales aplicaciones deben pasar por específicas pruebas de Quality Assurance (QA o "aseguramiento de calidad" en español).

Una de sus aplicaciones más complejas en el ámbito de pruebas de calidad es *Indemnizaciones*, ya que es netamente funcional (llenar formularios, cargar documentos y enviar solicitudes) y se puede comportar de cuarenta y cuatro formas distintas. Realizar el Testing manual de la aplicación en su totalidad requiere que un programador invierta más de 10 horas, valioso tiempo que podría aplicarse en otras labores, además de que, para un trabajador se vuelve desgastante y tedioso el hecho de ejecutar la misma tarea por un largo periodo de tiempo y durante varios días. Este desgaste podría generar consecuencias como errores y baja eficiencia en la ejecución de sus labores, de forma que tampoco se puede asegurar la calidad en la ejecución manual de los casos de prueba.

Generar los scripts que automaticen las pruebas de calidad, reducirá los recursos económicos, humanos y de tiempo que actualmente se invierten en la etapa de pruebas de calidad para la aplicación *Indemnizaciones*.

3. JUSTIFICACIÓN

Automatizar las pruebas de regresión para la aplicación *Indemnizaciones* es un proyecto que debe realizarse.

El primer beneficiario con la implementación del proyecto será el Tester, ya que no tendrá que invertir más de 10 horas ejecutando labores repetitivas y tediosas, para obtener un resultado posiblemente mediocre. Con la automatización de testing, únicamente tendrá que ejecutar los scripts de automatización, esperar a que hayan finalizado, y reportar las incidencias detectadas en caso de que las haya.

Ahora, viéndolo desde un enfoque más significativo, con la automatización de testing, los recursos que invierte IBM en la actualidad (destinados al testing manual) serán reducidos, tanto humana (no será necesario que un programador realice las pruebas de forma manual) como económicamente (no será necesario pagar para que alguien realice las pruebas), además, al ser menor el tiempo (y asegurando la calidad) en que se ejecutan las pruebas de forma automática, los proyectos podrán ser entregados con anticipación, beneficiando así a ambas partes del negocio: IBM y Seguros Bolívar, siendo este último el cliente.

4. OBJETIVOS

4.1 OBJETIVO GENERAL

Automatizar la ejecución de los cuarenta y cuatro (44) casos de prueba funcionales correspondientes a la aplicación "Indemnizaciones" para el proyecto de Seguros Bolívar, utilizando la herramienta "Selenium IDE" y programación en Java.

4.2 OBJETIVOS ESPECÍFICOS

- Documentar las funciones que cumple el equipo de gestión de calidad de proyectos dentro de la empresa IBM.
- Comparar la ejecución manual y automatizada de pruebas de QA, para así determinar la importancia de estas últimas, antes de la liberación de un proyecto a la etapa de producción.
- Definir una herramienta de proporcione más beneficios en la automatización de pruebas para la aplicación *Indemnizaciones*.
- Comprender mediante la utilización de Selenium IDE, su funcionamiento y valor agregado en la automatización de pruebas para cualquier aplicación web.
- Implementar la herramienta Selenium WebDriver bajo el lenguaje y entorno de programación *Java*, para llevar la automatización de pruebas a un nivel más técnico y especializado.

5. GLOSARIO

En esta sección se hace referencia a información que, aunque no esté relacionada directamente con el proyecto a desarrollar, es útil para comprender el entorno laboral en el que fue desarrollado. También se explican algunas abreviaturas o siglas presentes en el documento.

- IBM.

La sigla IBM significa “International Business Machine”. Es una empresa que se dedica a la comercialización de software y hardware para computadores, y ofrece servicios relacionados con la informática, desde infraestructura y alojamiento en internet, hasta consultoría (siendo la empresa más grande en este ámbito a nivel mundial) [1].

IBM facilita a sus clientes los métodos para hacer frente a los problemas empresariales mediante una adecuada utilización de las tecnologías de la información. La oferta de hardware, software, servicios y financiamiento de IBM es la más completa del mercado, lo que permite a la compañía ofrecer soluciones tecnológicas a cualquier tipo de cliente, desde usuarios particulares hasta instituciones y grandes empresas de cualquier sector de actividad.

La sede principal de IBM se encuentra en Nueva York, Estados Unidos, pero opera en otros 169 países y cuenta con más de 386.000 empleados.

Aproximadamente el 50% de los ingresos de la compañía proviene de prestación de servicios, mientras que el otro 50% restante se obtiene del área de software y la venta de hardware [2].

- GBS.

La sigla se refiere a “Global Business Services”, que traduce al español “Servicios de Negocios Globales”. GBS es una unidad de negocio de IBM centrada en la administración y estrategia de consultaría, integración de sistemas y desarrollo de aplicaciones, servicios y herramientas web.

La unidad está organizada en seis líneas de servicios principales: Análisis y Estrategias de Negocios, Experiencia Interactiva, Administración de Servicios, Mercado Medio, Aplicaciones de Servicios de SAP y Servicios de Microsoft [3].

- CAI.

La sigla hace referencia a “Cloud Application Innovation”, que traducida al español es “Aplicación e Innovación de la Nube”. Es una de las divisiones de GBS, que se encarga netamente al desarrollo de aplicaciones, servicios y herramientas web, utilizando tecnologías como Angular, Bus de integración (propios de IBM y Oracle), IBM Cloud, DB2, entre otras.

- TAC – CM – QA.

La sigla se refiere a “Technology Assembly Center – Configuration Management – Quality Assurance”, en español “Centro de Montaje de Tecnología – Administración de Configuración – Aseguramiento de Calidad”. Es el equipo del cual se hizo parte para el desarrollo de la pasantía. La mayor parte de las funciones que estaban bajo responsabilidad del pasante (Jorge Chaparro, redactor del presente documento) estaban relacionadas con el Aseguramiento de Calidad.

- URL.

La sigla se refiere a “Universal Resource Locator” o “Uniform Resource Locator”. Es la dirección utilizada para enviar un buscador web a una específica página web. No existen URLs iguales, cada página web tiene una su propia y única URL [4].

- PM.

Su cifra en inglés significa “Project Manager”, o “Jefe de Proyecto” al traducirlo al español. Como se puede inferir de su nombre, el PM es la persona que está a cargo de un proyecto, siendo él quien define y presenta el proyecto ante el cliente y el equipo de trabajo, establece objetivos, asigna roles y tareas, propone e implementa soluciones ante posibles adversidades y entrega el proyecto. El PM es el intermediario entre el equipo de trabajo y el cliente, y busca satisfacer a ambas partes, idealmente bajo términos justos [5].

6. MARCO TEÓRICO

6.1 AMBIENTES DE TRABAJO PARA SOFTWARE

Al momento de desarrollar un software, no es posible cambiar el proyecto directamente sobre el ambiente productivo porque todo el software podría verse afectado.

Para realizar cambios parciales y que el cliente pueda probarlos antes de aplicarlos en el producto final, se utilizan los entornos o ambientes de trabajo para software:

- Entorno o ambiente de desarrollo:

Es el entorno de trabajo para desarrolladores individuales o pequeños equipos de desarrolladores, en el que pueden probar cambios radicales en el código sin afectar de forma adversa al resto del equipo de desarrollo. Estos entornos suelen estar ubicados directamente en las estaciones de trabajo de cada desarrollador [6].

- Entorno o ambiente de integración:

Un entorno común donde todos los desarrolladores hacen "commits" (acción de guardar o subir los cambios al repositorio remoto, es decir, realizar una actualización de los cambios [7]) al código. La meta de este entorno es combinar y validar el trabajo del equipo completo del proyecto para que pueda ser testeado antes de ser promovido al entorno de testing. Es posible que el ambiente de desarrollo e integración sean uno mismo [6].

- Entorno o ambiente de pruebas o de testing:

Es un entorno muy semejante al de producción, de forma que pueda usarse como una simulación de este último, con el fin de testear las actualizaciones, para asegurar que

las mismas no corrompan la aplicación existente en los servidores de producción. Este ambiente puede utilizarse también para la capacitación de usuarios [6].

Existen dos tendencias que hacen referencia a cómo deben ser gestionados los ambientes y datos de prueba para que el proceso de creación de software y, en especial, el proceso de pruebas sea lo más óptimo posible:

TEM (Test Environment Management): este modelo propone la creación de una administración única y centralizada de los ambientes de prueba, haciendo monitoreo, mantenimiento y resolviendo las situaciones que se presenten.

TDM (Test Data Management): este modelo asegura que las bases de datos de los ambientes de prueba contengan todo lo necesario para hacer buenas pruebas, procurando velocidad, eficiencia y el despliegue continuo de software [8].

- Ambiente de producción:

Es el entorno donde trabajan los usuarios finales y se utilizan los datos de negocio. Debe estar aislado de los otros entornos [6].

6.2 TESTING

El testing es una actividad desarrollada para evaluar la calidad del producto, y para mejorarlo al identificar defectos y problemas. El testing de software consiste en la verificación dinámica del comportamiento de un programa sobre un conjunto finito de casos de prueba, apropiadamente seleccionados a partir del dominio de ejecución que usualmente es infinito, en relación con el comportamiento esperado [9].

Un caso de prueba podría definirse como un conjunto de entradas de prueba, condiciones de ejecución, y resultados esperados desarrollados con un objetivo particular, tal como el de ejercitar un camino en particular de un programa o el verificar que cumple con un requerimiento específico.

Cuando se plantea un caso de prueba, deben considerarse ciertos elementos fundamentales: flujo (sentencia de pasos por ejecutar), datos de entrada, estado inicial, valor de respuesta esperado y estado final esperado.

Del concepto “caso de prueba” se deriva otro: “idea de prueba”, que es una breve declaración de algo que debería ser probado. Por ejemplo, si se está probando la función de la raíz cuadrada, una idea de prueba sería el “probar un número que sea menor que cero”. La idea es chequear si el código maneja un caso de error [10].

Volviendo a lo referente a Testing, el programa se prueba ejecutando solo unos pocos casos de prueba dado que por lo general es física, económica o técnicamente imposible ejecutarlo para todos los valores de entrada posibles. Esos casos de prueba son elegidos siguiendo alguna regla o criterio de selección. Se determina si un caso de prueba ha detectado un error o no, comparando la salida producida con la salida esperada para la entrada correspondiente.

La especificación de un software, en términos más generales, debería incluir otros aspectos (o requerimientos no funcionales) más allá de la funcionalidad. Por ejemplo, suelen especificarse requisitos de seguridad, desempeño, tolerancia a fallas, usabilidad, etc. Por lo tanto, al decir que un programa es correcto si verifica su especificación, en realidad se están incluyendo todos esos aspectos además de la funcionalidad. En

consecuencia, al testear un programa deberían seleccionarse casos de prueba con el objetivo de testear todos esos aspectos y no solo la funcionalidad. Es decir, no solo se debería intentar responder preguntas tales como “¿El sistema permite cargar una transacción solo en las condiciones especificadas?", sino también otras como “¿El sistema permite cargar hasta 10.000 transacciones por minuto?", “¿El sistema permite que solo los usuarios autorizados carguen transacciones?", etc [11].

Existen dos metodologías clásicas para hacer testing:

- Testing estructural o de caja blanca:

Testear un software siguiendo esta estrategia implica que se tiene en cuenta la estructura del código fuente del programa para seleccionar casos de prueba (es decir, el testing está guiado fundamentalmente por la existencia de sentencias tipo “if”, “case”, “while” y demás). En muchas ocasiones se pone tanto énfasis en la estructura del código que se ignora la especificación del programa. Como los casos de prueba se calculan de acuerdo con la estructura del código, no es posible generarlos sino hasta que el programa ha sido terminado. Peor aún, si debido a errores o cambios en las estructuras de datos o algoritmos (incluso sin que haya cambiado la especificación), puede ser necesario volver a calcular todos los casos. Se dice que el Testing Estructural prueba lo que hace el programa y no lo que se supone que debe hacer. Está bien, el programa funciona, pero ¿cumple con el requerimiento del cliente? [12].

- Testing basado en modelos o de caja negra:

Significa ejecutar el software sin considerar ningún detalle sobre cómo fue implementado. Esta estrategia se basa en seleccionar los casos de prueba analizando la especificación o modelo del programa, en lugar de su implementación.

Como los casos de prueba se calculan partiendo del modelo, es posible comenzar a “testear" casi desde el comienzo del proyecto. Además, los casos de prueba calculados con técnicas de MBT (Model-Based Testing, o Testing basado en modelos) son mucho más resistentes a los cambios en la implementación que aquellos calculados con técnicas de testing estructural.

Se dice que el MBT prueba lo que se supone que debería hacer el programa, y no lo que realmente hace [12].

Para asegurar una mayor calidad en el producto que se entregará, lo que se sugiere es combinar ambas técnicas: la caja blanca y la caja negra. Sin embargo, esto muchas veces no es posible, ya que puede que el equipo de QA no tenga acceso a los recursos primarios (código fuente) brindados por el equipo de desarrollo, por lo que la técnica de testing estructural no pueda ponerse en práctica, y el único recurso para formular los casos de prueba sea el MBT.

En la figura 1 se muestra un esquema que puede tomarse como base para el diseño de casos de prueba:

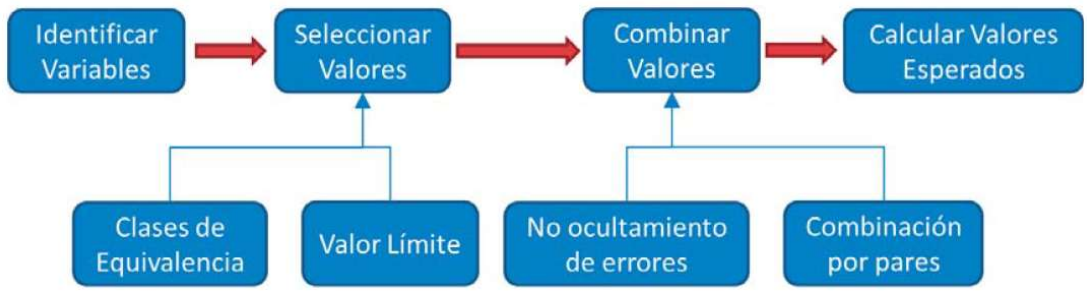


Figura 1. Proceso básico de diseño de pruebas [10].

La primera fase: identificar variables (las cuales pueden ser ingresadas por pantalla por el usuario, ser parámetros del sistema, ser estados de base de datos, entre otras), consiste en definir los componentes que afectarán el cómo se desarrollen los flujos de la aplicación.

La segunda fase: seleccionar valores, consiste en definir los valores que se asignarán a las variables de las cuales se habló previamente. Es importante considerar en esta fase a las "clases de equivalencia", estas son conjuntos de valores que harán que la aplicación se comporte de la misma forma. Considerando eso, podría decirse que al elegir un representante cualquiera de cada clase de equivalencia será suficiente para probar el total de las entradas. Dos valores no pertenecen a la misma clase cuando disparan un flujo distinto en el sistema, disparan un mensaje de error diferente, cambian la apariencia o contenido de la interfaz de usuario, cambian las opciones disponibles para otras variables, disparan distintas condiciones para una regla de negocio, cambian valores por defecto, cambian el comportamiento por distintas configuraciones o distintos países, entre otros.

La tercera fase: combinar de valores, es decir, mezclar de distintas formas los valores que pueden asignarse a las variables, pero de forma eficiente, lo que quiere decir, que con una mínima cantidad de combinaciones se pueda encontrar la mayor cantidad de defectos en la aplicación.

Por último, la fase de calcular valores esperados consiste en definir los resultados de acuerdo con las especificaciones del sistema, o con el conocimiento que se tenga del negocio. En otras palabras, consiste en diseñar el oráculo de prueba [10].

Es importante considerar también que, en el Testing, las pruebas que se ejecutarán se clasifican en tres tipos:

- Pruebas funcionales:

Este tipo de pruebas se basa en las funcionalidades de un sistema, descritas en la especificación de requisitos. Se caracterizan por la completitud funcional (el grado en el que las funcionalidades cubren todas las tareas y objetivos del usuario especificados), corrección funcional (capacidad del producto o sistema para proveer resultados correctos con el nivel de precisión requerido) y pertenencia funcional (capacidad del producto de software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados). Las pruebas funcionales suelen estar asociadas a las técnicas de diseño de pruebas de caja negra, ya que tienen en cuenta el comportamiento externo del software.

- Pruebas no funcionales:

Suelen utilizar técnicas de diseño de caja negra y deben definirse en las especificaciones del producto. Las pruebas no funcionales están enfocadas en las pruebas de carga (medición del comportamiento del sistema al aumentar el número de peticiones que se le realizan en cierto periodo de tiempo o simultáneamente), pruebas de rendimiento (en las que se mide la velocidad de procesamiento y tiempo de respuesta del sistema), pruebas de volumen (capacidad del sistema para procesar gran cantidad de datos), pruebas de esfuerzo (se sobrecarga al sistema y se analiza la capacidad de recuperación), pruebas de seguridad (intentar ingresar con accesos no autorizados, ataque de denegación de servicios, etc), pruebas de estabilidad, eficiencia y robustez (medición

de la respuesta del sistema a los errores de funcionamiento), pruebas de compatibilidad (funcionamiento del sistema bajo distintos sistemas operativos, plataformas de hardware, etc) y pruebas de usabilidad (facilidad de uso, efectividad y satisfacción) [13].

- Pruebas estructurales:

Permiten medir la totalidad de las pruebas mediante la evaluación de tipo estructura. En estas pruebas se aplican las técnicas de diseño de caja blanca [14].

6.3 HERRAMIENTAS PARA PLANIFICACIÓN Y GESTIÓN DE PRUEBAS

Las herramientas de gestión ayudan al probador a documentar y evaluar los casos de prueba. Tienen como objetivo proporcionar mecanismos que permitan realizar de una manera controlada la documentación, el mantenimiento de las pruebas y la gestión de resultados. Dentro de las herramientas de gestión no sólo se encuentran las que gestionan las pruebas, sino que también están las que gestionan incidencias.

Las herramientas de planificación permiten al probador escoger una estrategia de prueba, así como tener una visión general de los casos de prueba del proyecto.

Actualmente, Jira es la herramienta que se utiliza para los proyectos desarrollados por CAI, en GBS, IBM, Colombia. Cuenta con interfaz web, en la cual es posible gestionar incidencias, crear y asignar tickets, modificar su tipo (épica, historia, tarea, bug, etc), severidad (alta, media, baja, etc), cargar documentos, agregar comentarios y descripciones, además puede enlazarse con otras aplicaciones externas (correo electrónico, páginas web creadas, etc).

6.4 AUTOMATIZACIÓN DE TESTING

Antes de explicar a lo que es automatización de testing, es necesario conocer el concepto de “Pruebas de regresión”; si bien no es para lo único que se utiliza la automatización de pruebas, tal vez sea la opción más popular y la más usada.

Las pruebas de regresión son un subconjunto de las pruebas planificadas que se seleccionan para ejecutar periódicamente, por ejemplo, ante cada nueva liberación del producto. Tienen el objetivo de verificar que el producto no ha sufrido regresiones.

Tampoco es válido pensar que las pruebas de regresión se limitan a verificar que se hayan arreglado los bugs que se habían reportado, pues es igual de importante ver que lo que antes andaba bien ahora siga funcionando.

Generalmente cuando se diseñan las pruebas para determinadas funcionalidades, ya se definen cuáles son las pruebas que serán consideradas dentro del set de pruebas de regresión. O sea, las que serán ejecutadas ante cada nueva liberación del producto, o en cada ciclo de desarrollo [10].

Las pruebas que se aplican en un proyecto, aunque éste sea uno pequeño, pueden llegar a ser muy numerosas, por lo que para realizar pruebas como son las de regresión, implicaría el gasto de una gran cantidad de recursos y tiempo, normalmente más del que un proyecto se puede permitir para realizar las pruebas.

Las pruebas automáticas aportan al proyecto velocidad, eficiencia, precisión y resistencia, ya que con la automatización se incrementa la velocidad al ejecutar las pruebas y, al no ejecutarse manualmente por un probador o tester, éste se puede dedicar a

realizar otras tareas del proyecto. Las pruebas manuales pueden presentar defectos al ser ejecutadas de nuevo por los testers, ya que al probar de nuevo el sistema se tiende a relajar el nivel de búsqueda de errores, mientras que la herramienta utilizada para automatizar nunca se cansa ni se distrae.

Las pruebas se generan mediante la creación de scripts que ejecutarán el software de manera automática. Para la creación de estos scripts se encuentran en el mercado varias herramientas, de pago y libres. Éstas funcionan con diferentes lenguajes de programación, lo que permite una gran diversidad a la hora de elegir una de ellas para realizar la automatización de las pruebas del proyecto [14].

6.5 HERRAMIENTAS PARA LA AUTOMATIZACIÓN DE TESTING

El objetivo de estas herramientas es la creación de scripts en diferentes lenguajes de programación, que permitan ejecutar las pruebas funcionales de forma automática.

Algunas herramientas para la automatización de testing son:

- Selenium (libre):

Es un conjunto de herramientas para automatizar los navegadores web a través de muchas plataformas que permitirán la creación de conjuntos de prueba sobre aplicaciones web. Permite la creación de scripts mediante una gran variedad de lenguajes de programación como Java, Ruby, Python, etc [15]. De esta herramienta se hablará más a detalle posteriormente.

- QTP - Quick Test Professional (pago por licencia):

Es el módulo de automatización de la empresa HP. Permite la automatización de casos de prueba. Los scripts se programan en Visual Basic Script [16].

- SoapUI (libre/pago por licencia):

Permite probar, simular y generar código de servicios web partiendo del contrato de estos en formato WSDL, y con vínculo SOAP sobre http [17].

- Cucumber (libre):

Permite la automatización de la prueba de aceptación para aplicaciones web. Está basada en BDD (Behavior Driven Development) y el lenguaje de generación de scripts es Ruby (esencialmente, pero puede modificarse) [18].

La tabla 1 compara de forma puntual las características más significativas de las herramientas mencionadas previamente:

Característica	Selenium	QTP	SoapUI	Cucumber
Licencia	Es Open Source	Necesaria	Algunas características requieren licencia	Es Open Source
Costo	Sin costo, al ser open source	Alto, ya que se requiere licencia	Depende de las funciones que se quieran realizar	Sin costo, al ser open source
Consumo de recursos de cómputo durante la ejecución de los scripts.	Bajo consumo	Alto consumo	Alto consumo	Bajo consumo
Experiencia en programación requerida	Requerida para Selenium WebDriver. No requerida para Selenium IDE	Se requieren conocimientos básicos.	Se requiere conocimiento en estructuras XML y WSDL	Se requieren conocimientos básicos de Ruby, y del lenguaje de programación secundario.

Sistemas operativos compatibles	Windows, Linux, Solaris OS X	Windows	Windows, Linux, MAC	Windows, Linux, MAC
Lenguajes de programación compatibles	Java, C#, Ruby, Python, Perl, PHP, JavaScript	VB Script	WSDL, XML	BDD, Ruby (esencialmente), Java, C#, Python, .NET.
Se enfoca al testing de	Páginas y aplicaciones web.	Repositorios empresariales web.	Servicios web.	Páginas y aplicaciones web.

Tabla 1. Comparación de las características de algunas herramientas utilizadas para la automatización de Testing [15] [16] [17] [18].

6.6 SELENIUM

Selenium es un conjunto de herramientas de software para automatizar navegadores web y aplicaciones web. Selenium es Open Source y es por esto por lo que se está convirtiendo en un recurso ampliamente utilizado, por encima de otros con similar funcionalidad. Selenium es compatible con muchos lenguajes de programación (Java, JavaScript, Phyton, PHP, etc), sistemas operativos (Windows, Mac, Linux, etc) y buscadores (IE, Chrome, Safari, Firefox, etc).

Algunas de las herramientas más relevantes con las que Selenium cuenta son:

- Selenium IDE: Es una herramienta mediante la cual se pueden ejecutar de forma automática casos de prueba que previamente fueron ejecutados de forma manual por el tester.
- Selenium RC: “Selenium Remote Control”, se usa para crear códigos en distintos lenguajes de programación con el fin de hacer testing a aplicaciones web. Hay que iniciar un servidor, proveer instrucciones al Core, y el Core las proveerá al

buscador. Al no existir una comunicación directa entre el script y el buscador, el sistema RC se vuelve más lento que otros.

- Selenium WebDriver: Es una interfaz de programación para crear y ejecutar casos de prueba mediante elementos, objetos localizadores, y métodos de WebDriver. Al ser una API (interfaz de programación de aplicaciones), no tiene un IDE (entorno de desarrollo integrado) incorporado, por lo que es necesario utilizar uno para desarrollar los scripts, como Netbeans, Eclipse, entre muchos otros. Puede que haya limitaciones de WebDriver cuando se trabaje con espacios para subir archivos, descargas, pop-ups, y barreras de diálogos.
- Selenium Grid: Se usa para ejecutar múltiples comandos de prueba en múltiples equipos. Se basa en una arquitectura de maestro y nodos. Los nodos pueden trabajar con diferentes sistemas operativos, diferentes buscadores, o diferentes lenguajes de programación. Esto se usa cuando realmente no se sabe en qué condiciones está trabajando el cliente; así que Grid permite que las aplicaciones funcionen sobre cualquier ambiente.

Es necesario contemplar las siguientes consideraciones antes de comenzar a desarrollar los scripts de automatización:

- El tiempo de ejecución del Script que se realice siempre será mucho más rápido que los tiempos de carga de la página que se vaya a testear. Ya que el primero, por más extenso que sea, se ejecutará en milisegundos. En cambio, la página requiere mucho más tiempo (varía dependiendo al servidor de la página que se

vaya a probar, y por supuesto, las condiciones de la red a la cual se está conectado) para cargar todos sus elementos y componentes. Es por esto por lo que en el Script deben existir funciones que sincronicen la ejecución de los comandos con el estado de la página Web.

- Existen ocho formas distintas para buscar elementos en una página web. Se requieren funciones distintas de WebDriver para ejecutar la determinada búsqueda. Y el valor que toma el atributo de cada búsqueda puede encontrarse en el HTML de la página.
- Existen distintos tipos de comandos en Selenium: comandos de acción (interactúan directamente con los elementos de la aplicación web. Algunos de ellos son: “open”, “set window size”, “click”, “type”, etc.), comandos de acceso (examinan el estado de una aplicación, y almacenan este estado en una variable. Algunos de ellos son: “store”, “store value”, etc) y comandos de acierto (Verifica el estado actual de un elemento de la aplicación web con un valor esperado. Hay tres tipos de comandos de acierto, el primero es el de Acierto (assert), que se asegura de terminar la ejecución del caso de prueba en caso de que un valor en cierto elemento no sea el esperado. El segundo es el de verificación, este solamente pone como “Fallido” una comparación, pero no detiene la ejecución del caso de prueba. El tercero es el de esperar, este también fija como “fallido” una comparación en caso de que no aparezca un elemento particular, pero tampoco detiene la ejecución del caso de prueba).

Para desarrollar el proyecto en cuestión, se utilizarán únicamente de las herramientas Selenium IDE y Selenium WebDriver, por lo que se hará énfasis solo en estas dos:

Se debe instalar Selenium IDE para Chrome (en caso de que éste sea el navegador que se utilizará, también hay versiones para otros navegadores). Entonces, aparecerá el símbolo de la extensión en la sección destinada para éstas, tal como se muestra en la figura 2:

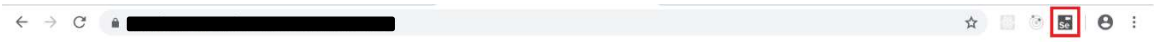


Figura 2. Extensión de Selenium IDE para Google Chrome.

Al darle clic, se abrirá en pantalla el pop-up mostrado en la figura 3:

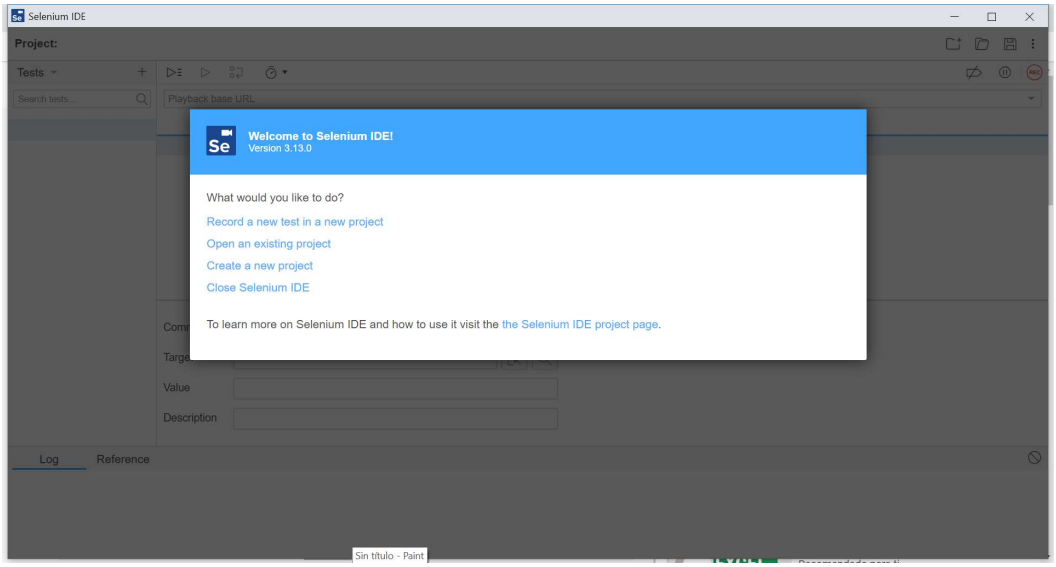


Figura 3. Menú de inicio para Selenium IDE.

Se selecciona la opción de crear un nuevo proyecto, en caso de que efectivamente se quiera hacer eso. Se debe dar un nombre al proyecto. Luego, una vez creado, se comienzan a añadir los casos de prueba, estos deben tener un nombre y se debe indicar la URL de la página web en donde comenzará el testeo.

Una vez hecho esto, se comenzarán a grabar los comandos que realice el usuario, los cuales podrán ser replicarlos automáticamente. Cuando finalice la ejecución manual del caso de prueba, se debe detener la grabación, al hacer esto, todos los comandos que se hayan realizado quedarán guardados en el panel del caso de prueba en cuestión.

Si es necesario modificar algún caso de prueba, no tiene que grabarse desde cero, se pueden reanudar grabaciones desde cualquier punto, además de agregar, editar o eliminar comandos de forma manual.

Se da la posibilidad de exportar el proceso como lenguaje de programación (desde Selenium IDE se puede exportar como Java, JavaScript y Python), para que sea un script el que ejecute el caso de prueba.

El correcto funcionamiento de los scripts generados desde Selenium IDE requieren el uso de la herramienta Selenium WebDriver, que es básicamente un conjunto de librerías y dependencias [19] [20] que deben incluirse en el archivo "pom.xml". Además, debe instalarse Chrome Driver [21] para que Selenium WebDriver pueda controlar el navegador Google Chrome.

En este punto se tienen las dependencias necesarias para poder correr un script de Selenium [22].

6.7 PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA

La programación orientada a objetos se enfoca en los elementos de un sistema, sus atributos y las interacciones que se producen entre ellos para diseñar aplicaciones informáticas. Los elementos abstractos del modelo orientado a objetos se denominan

"clases" [24]. La programación orientada a objetos se basa en la programación de clases. Y un programa se construye a partir de un conjunto de clases [23].

En la programación orientada a objetos, siempre debe existir una clase principal (que contenga la función *main()*). Además, pueden existir clases secundarias (específicas de la aplicación que se está desarrollando), las cuales pueden ser llamadas en la clase principal. Cada clase realiza funciones particulares, permitiendo construir las aplicaciones con gran modularidad e independencia entre clases. Las clases en Java se agrupan en *packages,* que son librerías de clases. También se pueden definir nuevas clases basadas en clases ya existentes, lo que permite reutilizar código previamente desarrollado; esto es una propiedad llamada "Herencia" [23].

Una vez que se ha declarado una clase, se pueden crear (instanciar) objetos a partir de ella. La instanciación de los objetos debe hacerse de la siguiente forma:

Clase = new Objeto();

El operador "new" se encarga de asignar un espacio en memoria para el objeto "Objeto".

La información de un objeto se almacena en atributos, los cuales son características propias de los objetos de la clase. Estas características pueden ser de tipo caracter (string, char), numérico entero (byte, short, int, long), numérico decimal (double y float) y booleano (boolean).

Además de definir los atributos de un objeto, es necesario definir los métodos que determinan su comportamiento. Toda clase debe tener un método especial denominado "constructor". Este método tiene el mismo nombre de la clase y se ejecuta cada vez que

se instancia un objeto de la clase. Para diferenciar entre los atributos del objeto y los identificadores de los parámetros del método constructor, se utiliza la palabra *this*. De esta forma, los parámetros del método pueden tener el mismo nombre que los atributos de la clase [24].

Es posible agregar a un método el operador *throws Exception()*, y es significativamente útil cuando se quiere indicar la sección del programa (o del método, en este caso) que presenta fallas, inconsistencias, problemas y demás situaciones que no permiten la correcta y total ejecución del código [25].

6.8 SÍNTESIS DEL MARCO TEÓRICO

El marco teórico expuesto anteriormente, al ser un poco extenso, puede generar la pérdida de ideas en el lector, por lo que en la figura 4 se presenta un mapa conceptual que reúne la información más relevante:

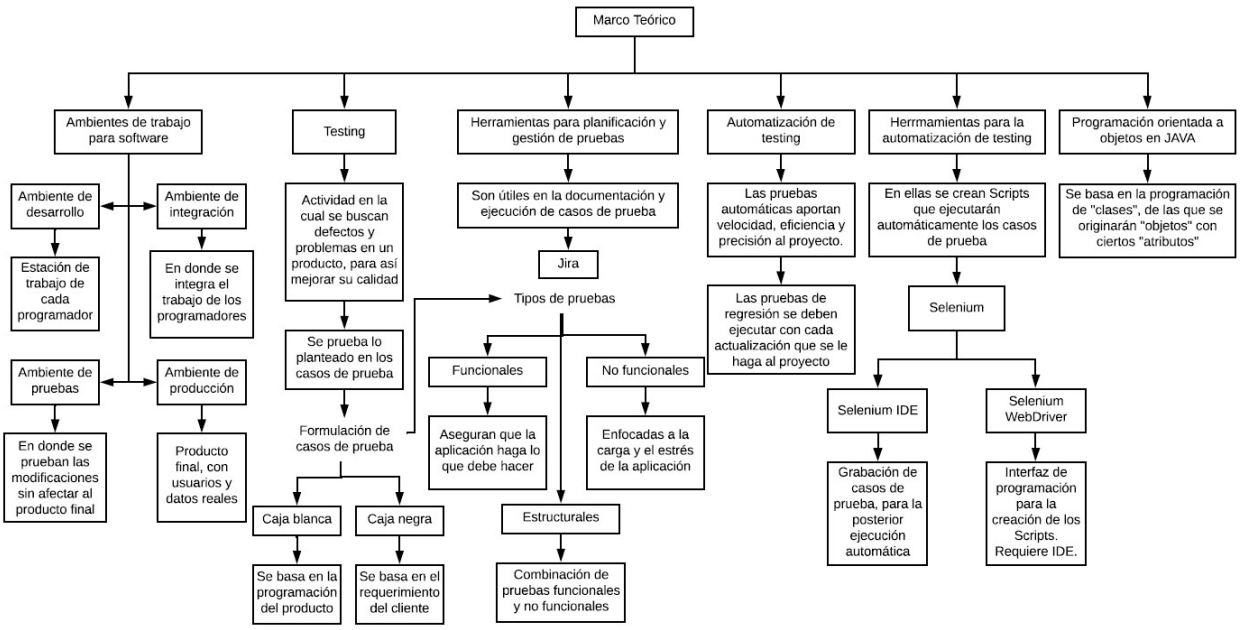


Figura 4. Mapa conceptual que resume la información contenida en el marco teórico.

7. METODOLOGÍA

El desarrollo del proyecto constará de siete etapas (se denominarán “capítulos” posteriormente) principales, la primera se trata de conocer el entorno de la aplicación “Indemnizaciones”, esto es identificar su estructura, los flujos que deben cumplirse, cómo deben llenarse los formularios, qué validaciones deben hacerse y cómo se comportan sus servicios, tanto de front (interfaz visual por el usuario) como de back (procesos que se realizan y no son presenciados por el usuario).

La segunda etapa consiste en el planteamiento de casos de pruebas (qué se va a probar) para la aplicación.

La tercera etapa consiste en realizar pruebas de QA (ejecución de los casos de prueba) de forma manual, tanto de front (verificar que se cumplan los flujos, los servicios funcionen, las validaciones se apliquen, los componentes de la página se muestren correctamente, entre otros) como de back (los documentos cargados a la aplicación se almacenan en los registros correspondientes de base de datos).

La cuarta etapa consiste en definir si es viable realizar la automatización de las pruebas de QA. Además de definir qué casos de prueba se pueden automatizar (partiendo desde el punto de que no cualquier página web puede automatizarse).

Después de definir qué se va a automatizar, se debe buscar una herramienta/software/extensión, que permita realizar la automatización, y tener claro por qué es superior a las demás. Esta es la quinta etapa.

Al haber definido la herramienta que se utilizará (Selenium), se comenzará con la sexta etapa: automatizar la ejecución de los casos de prueba viables, utilizando Selenium

IDE (Integrated Development Environment) [26] y se comprobará su correcto funcionamiento.

La séptima y última etapa consiste en utilizar programación en Java sobre el editor de texto “Eclipse”, para generar los scripts o códigos que ejecuten las pruebas de QA de forma automática, además de comprobar su correcto funcionamiento.

Según lo mencionado anteriormente, se hará uso de la investigación aplicada [27], ya que mediante herramientas (Selenium IDE y programación en Java) se busca dar solución a una problemática específica (automatizar la ejecución de casos de prueba para la aplicación “Indemnizaciones” del proyecto Seguros Bolívar).

8. DESARROLLO

8.1 CAPÍTULO 1: APLICACIÓN “INDEMNIZACIONES”

Como bien se mencionó antes, esta sección se centra en brindar una contextualización general sobre el funcionamiento de la aplicación “Indemnizaciones”.

La vista inicial que se tendrá al ingresar con la URL de la página será la siguiente:

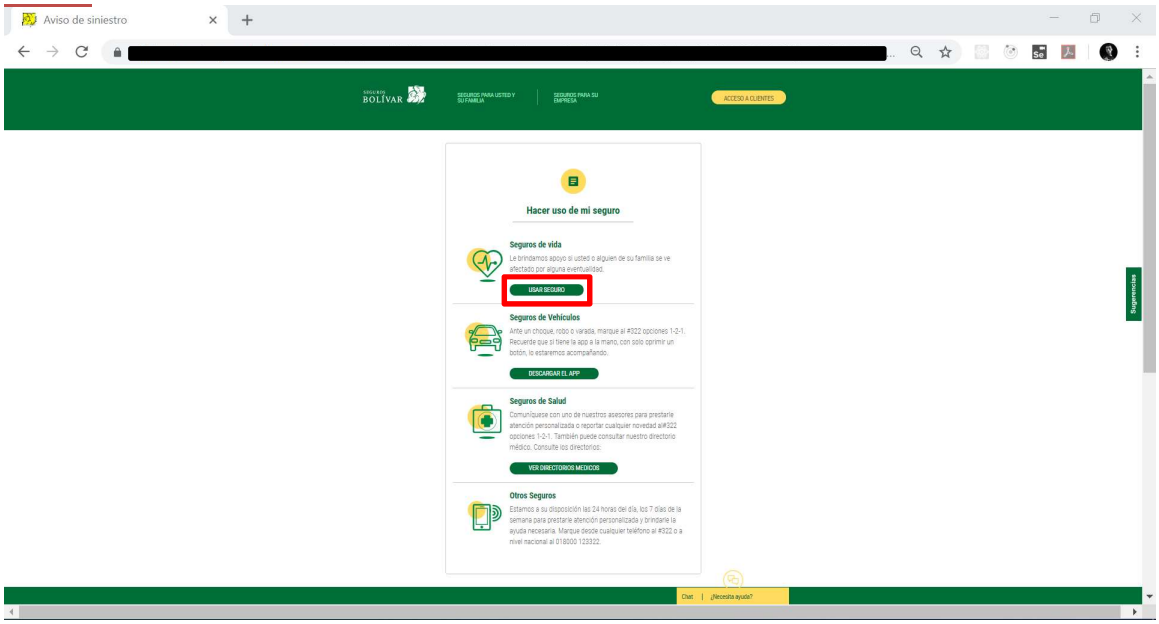


Figura 5. Página principal de la aplicación “Aviso de siniestro”.

La aplicación “Indemnizaciones” se refiere a la sección “Seguros de vida” mostrada en la figura 5, así que, para continuar con el flujo, se debe dar clic al botón “Usar seguro” posicionado dentro de la sección en cuestión, lo cual generará un redireccionamiento a la siguiente pantalla:

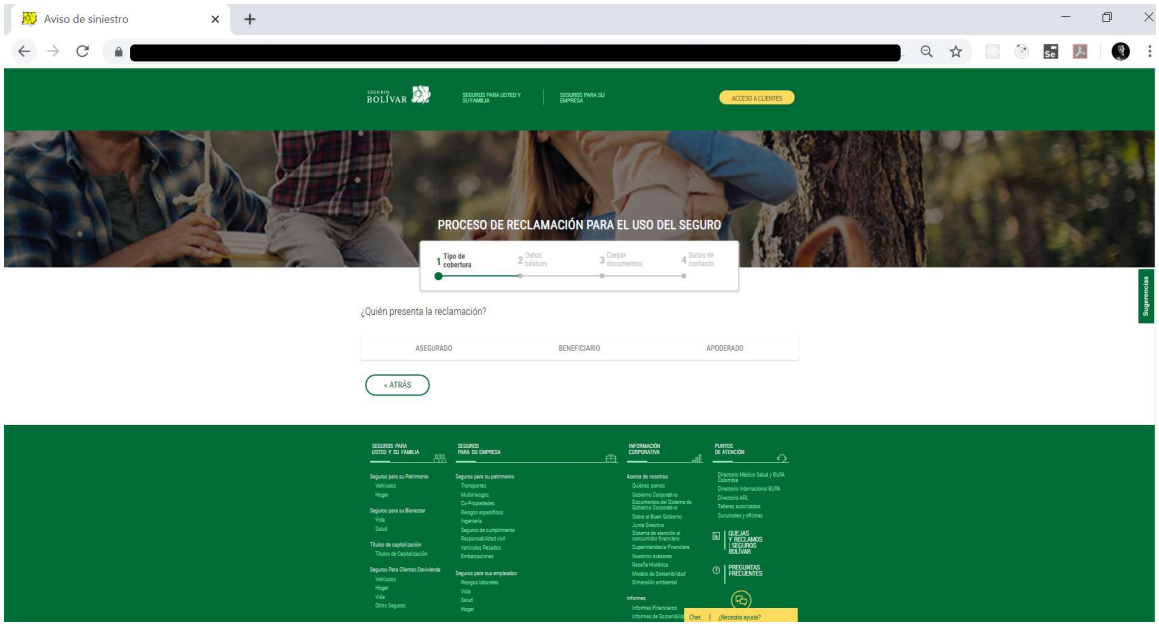


Figura 6. Página principal de la aplicación “Indemnizaciones”.

El componente principal de la pantalla mostrada en la figura 6 es una sección “¿Quién presenta la reclamación?” (tipo de solicitud) compuesto de tres botones: “Asegurado”, “Beneficiario” y “Apoderado”; al dar clic sobre ellos, la pantalla cambiará a como se muestra respectivamente en las figuras 7, 8 y 9.

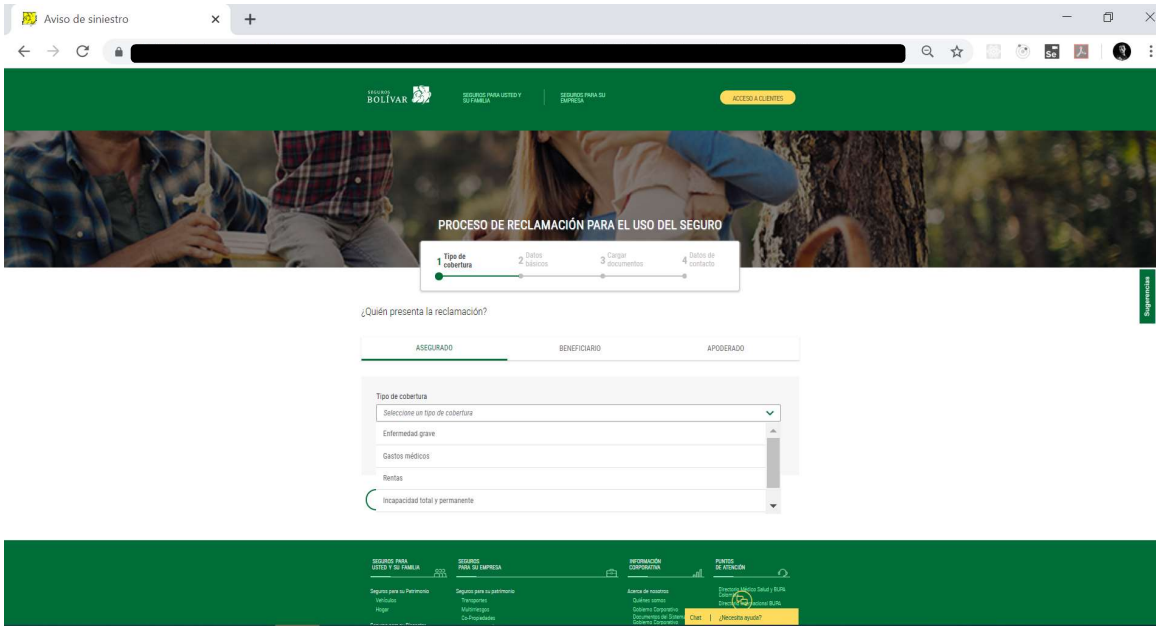


Figura 7. Comportamiento de la aplicación al dar clic en el botón “Asegurado”.

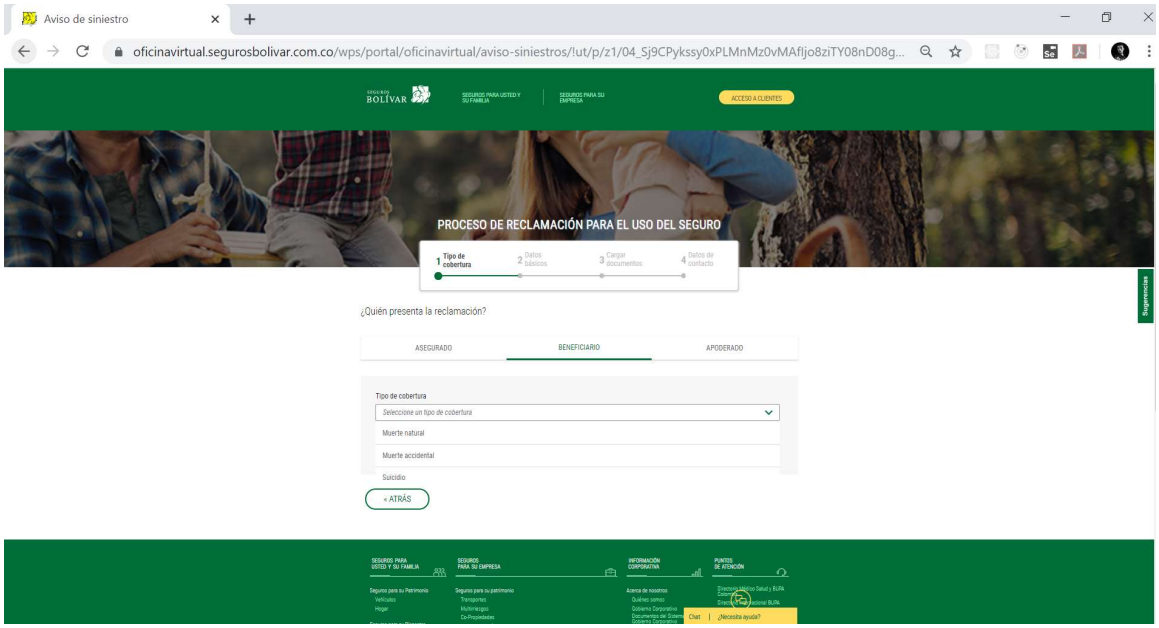


Figura 8. Comportamiento de la aplicación al dar clic en el botón “Beneficiario”.

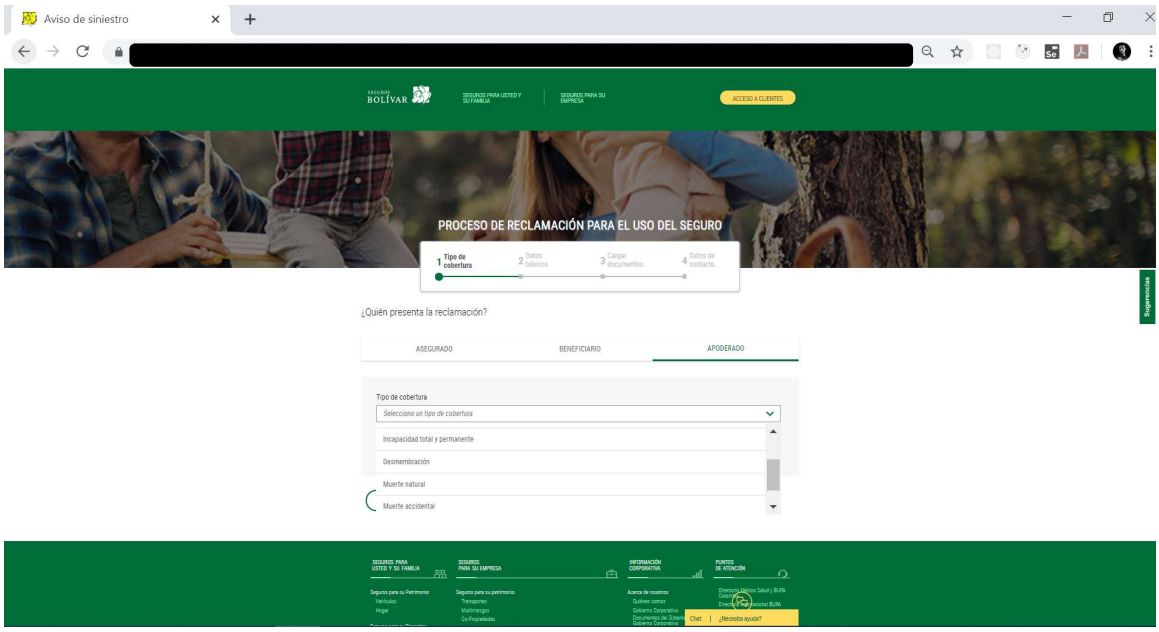


Figura 9. Comportamiento de la aplicación al dar clic en el botón “Apoderado”.

Para los tres botones sucede algo similar: aparece en pantalla una lista desplegable “Tipo de cobertura”. Sin embargo, los componentes que aparecen en ella son distintos dependiendo del tipo del tipo de solicitud que se haya seleccionado. Para el caso de “Asegurado”, los componentes de la lista son “Enfermedad Grave”, “Gastos Médicos”,

“Rentas”, “Incapacidad Total y Permanente” y “Desmembración”. Para “Beneficiario”, los componentes que se muestran en la lista son “Muerte Natural”, “Muerte Accidental” y “Suicidio”. Mientras que para “Apoderado”, se muestran en la lista los componentes “Enfermedad Grave”, “Gastos Médicos”, “Rentas”, “Incapacidad Total y Permanente”, “Desmembración”, “Muerte Natural”, “Muerte Accidental” y “Suicidio”.

Es entonces cuando la aplicación se comenzará a comportar de forma distinta para cada uno de los posibles flujos.

Se explicará a detalle un flujo para cada tipo de solicitud.

- Para una solicitud de “Asegurado”:

Suponiendo que se haya seleccionado el tipo de cobertura “Enfermedad Grave”:

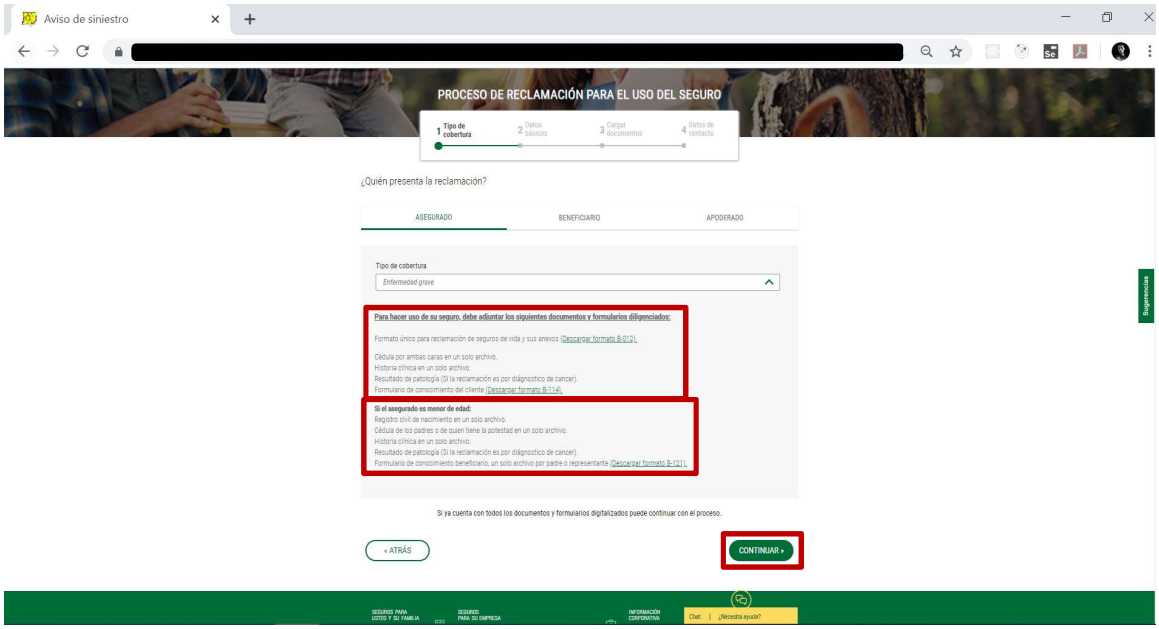


Figura 10. Comportamiento de la aplicación bajo un flujo con tipo de solicitud “Asegurado” y tipo de cobertura “Enfermedad Grave”.

La figura 10 muestra el listado de documentos que deberán considerarse para realizar la solicitud. Este listado de documentos varía dependiendo del tipo de cobertura

que se haya seleccionado. Además, está dividido en dos secciones, la primera de ellas está compuesta de los documentos que se deben considerar siempre, y la segunda está compuesta de los documentos que se deben considerar en caso de que la persona que esté realizando la solicitud, en este caso, el "asegurado", sea una persona menor de edad.

Para continuar con el flujo, se debe dar clic al botón "continuar". Lo cual cargará la pantalla mostrada en la figura 11:

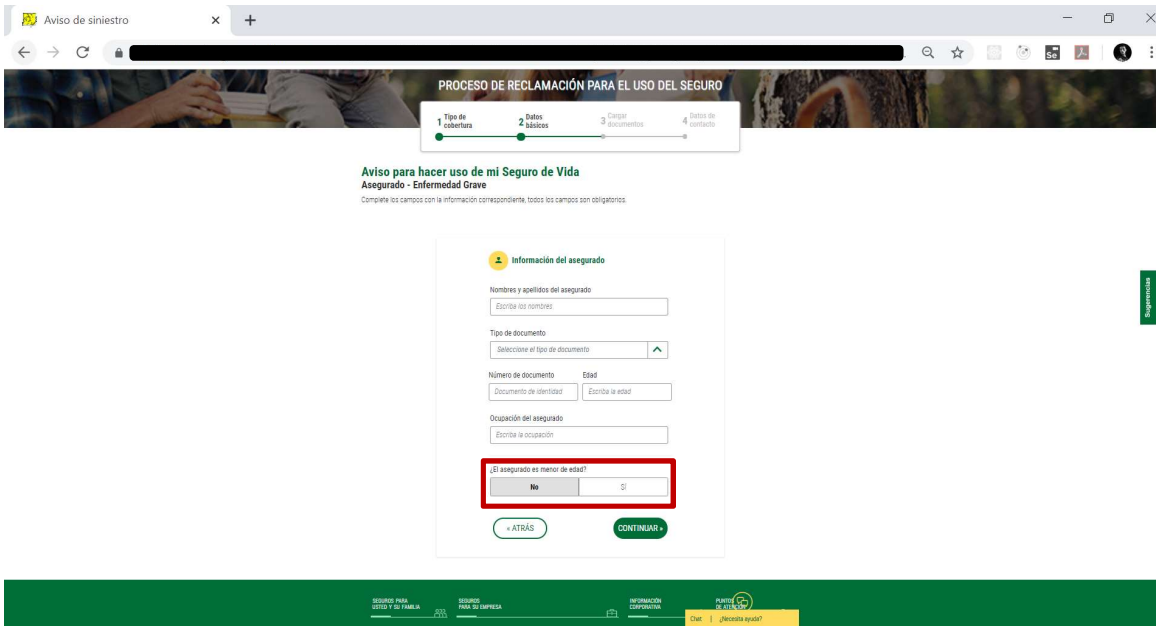


Figura 11. Formulario "Información del asegurado".

El formulario está compuesto por varios elementos: cuatro campos de entrada de texto destinados a ingresar el nombre, número de documento, edad y ocupación del asegurado, una lista desplegable en la cual se indica el tipo de documento del asegurado (los componentes de esta lista provienen de un servicio propio de Seguros Bolívar) y, por último, un *radio-button* mediante el cual se indica si el asegurado es o no menor de edad. Este último componente hará que la aplicación se comporte de forma diferente según la opción seleccionada.

Para continuar con el flujo, se debe llenar el formulario mostrado en la figura 11 y dar clic al botón “continuar”, que dependiendo de lo que se haya seleccionado mediante el *radio-button*, redireccionará a alguna de las siguientes pantallas:

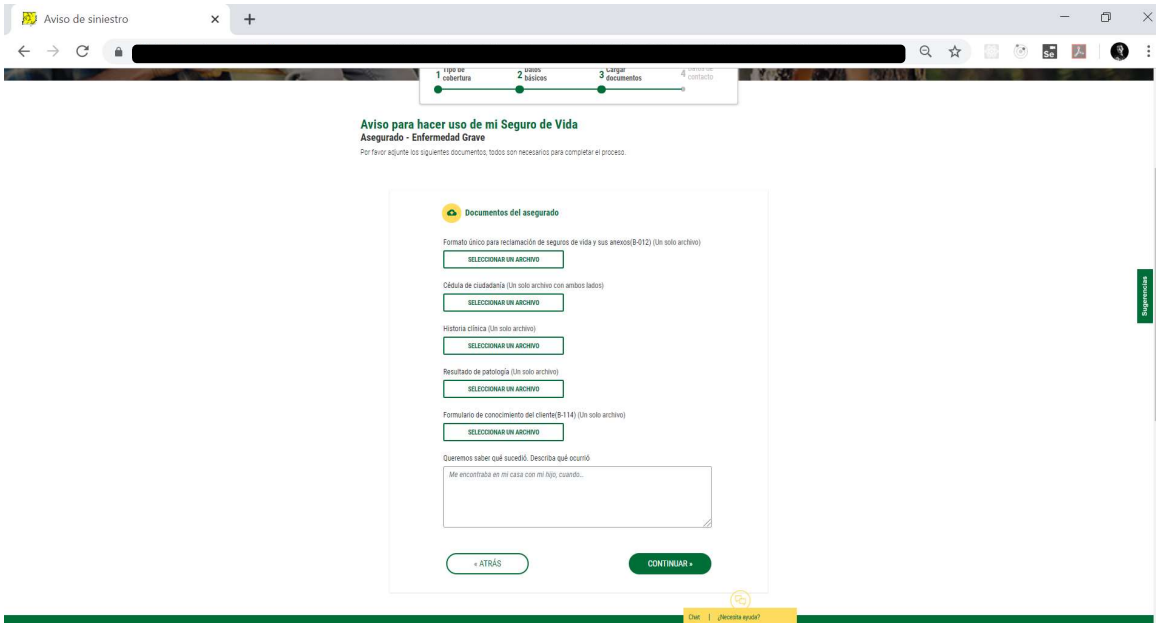


Figura 12. Documentos de asegurado, si éste no es menor de edad.

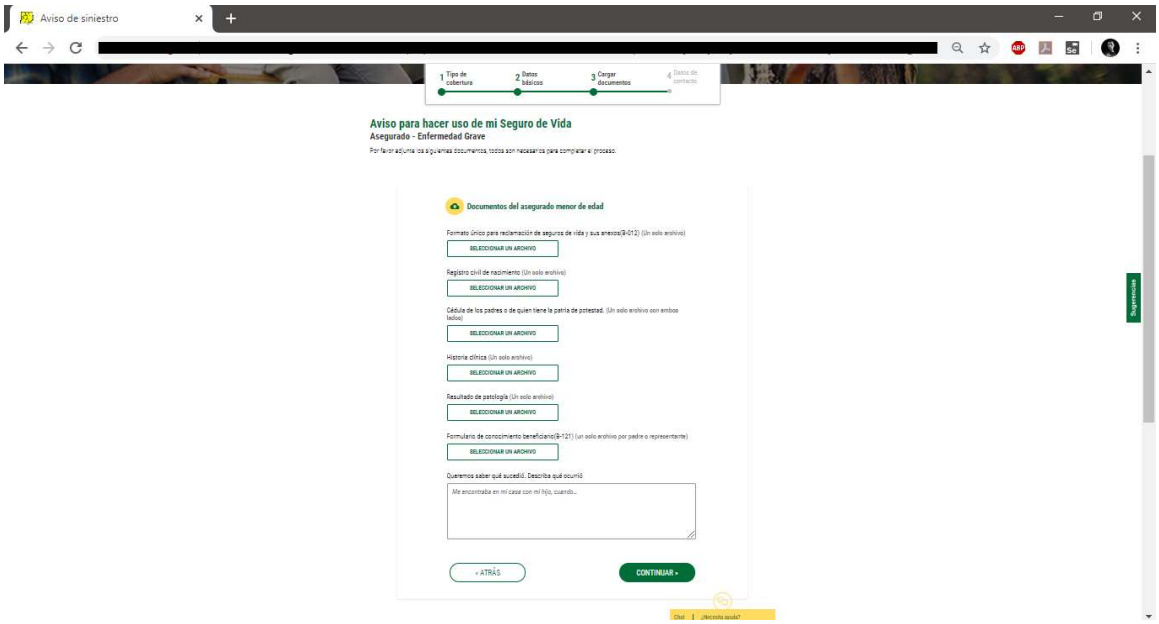


Figura 13. Documentos de asegurado, si éste es menor de edad.

Para el caso de la figura 12, los campos para cargar documentos deben coincidir con los documentos que se solicitan en la primera sección de la figura 10. Mientras que para el caso de la figura 13, los campos para cargar documentos deben coincidir con la totalidad de documentos (primera y segunda sección) que se solicitan en la figura 10.

Para ambos casos, debe añadirse una pequeña descripción del suceso, mediante el campo de entrada de texto posicionado bajo los campos de carga de documentos. El flujo continúa una vez se hayan cargado los documentos necesarios, y se dé clic en el botón "continuar". Entonces la pantalla cambia a la que se muestra en la figura 14:

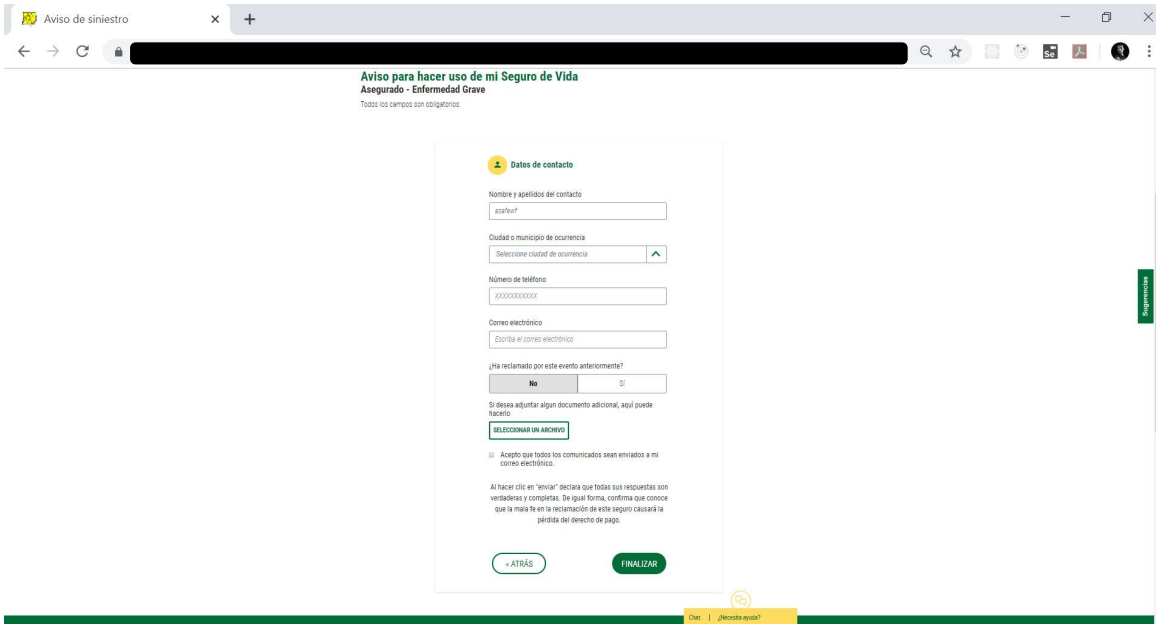


Figura 14. Formulario de datos de contacto.

El cual está compuesto de distintos elementos: tres campos de entrada de texto para ingresar el nombre (el cual se llena por defecto con la información proporcionada en el formulario "información del asegurado" de la figura 11, sin embargo, es posible editarlo en el formulario actual), número de teléfono y correo electrónico; una lista desplegable para indicar la ciudad o municipio de ocurrencia (los componentes de esta

lista se obtienen de un servicio propio de Seguros Bolívar); un *radio-button* para indicar si anteriormente ya se había realizado o no una solicitud bajo la misma situación (lo que sea seleccionado no afectará al flujo posterior), un campo de carga de documento en caso de que se desee adicionar información extra (no es obligatorio) y un *check-box* mediante el cual se aceptan los términos y condiciones de la aplicación. Una vez se ha llenado el formulario, se da clic en el botón "finalizar" y si el proceso se realizó correctamente, aparecerá pop-up mostrado en la figura 15.

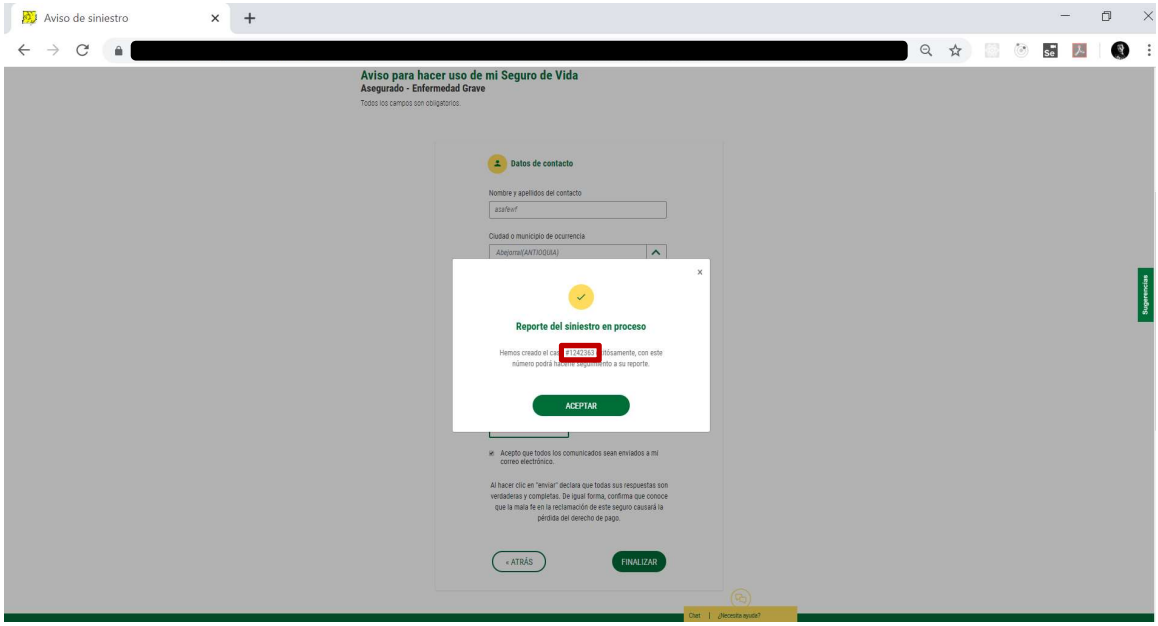


Figura 15. Pop-up informativo del éxito de la solicitud.

El mensaje contiene un número de caso único, correspondiente a la solicitud que se acaba de realizar. En caso de que los servicios estén fallando, o haya ocurrido alguna inconsistencia a la hora de realizarse la solicitud, el mensaje del pop-up cambiará y se mostrará el mensaje de la figura 16:

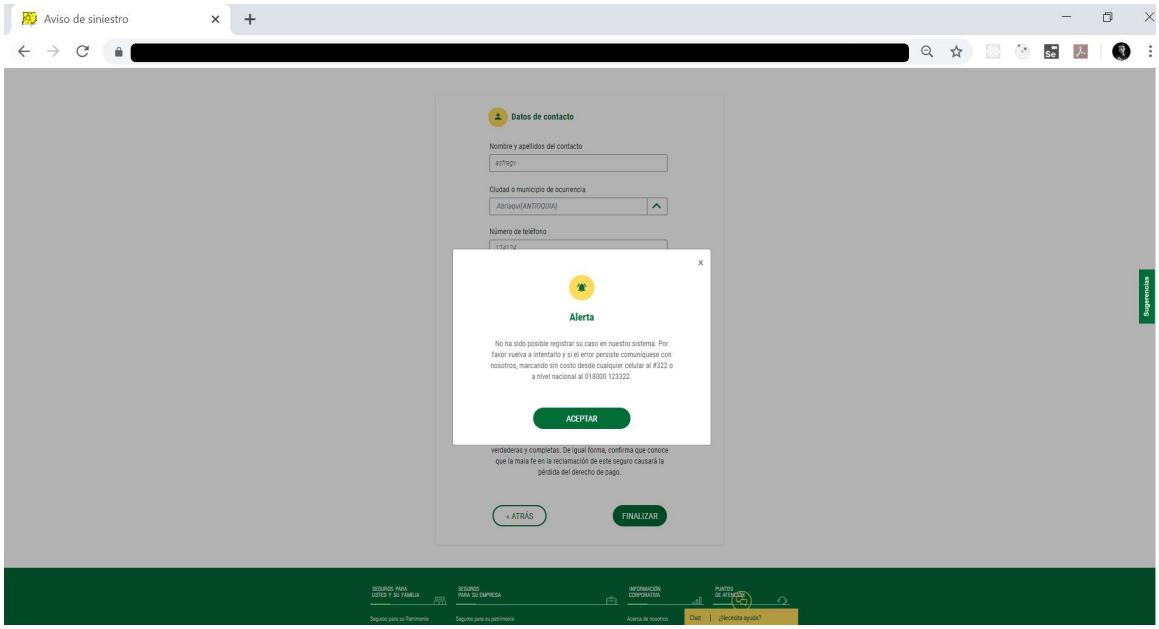


Figura 16. Pop-up informativo de la falla en el proceso de solicitud.

En ambos casos, al dar clic al botón “Aceptar” dentro del pop-up, redireccionará a la página inicial de la aplicación “Aviso de siniestros” mostrada en la figura 5.

- Para una solicitud de “Beneficiario”:

  Suponiendo que se haya seleccionado como tipo de cobertura “Muerte Natural”:

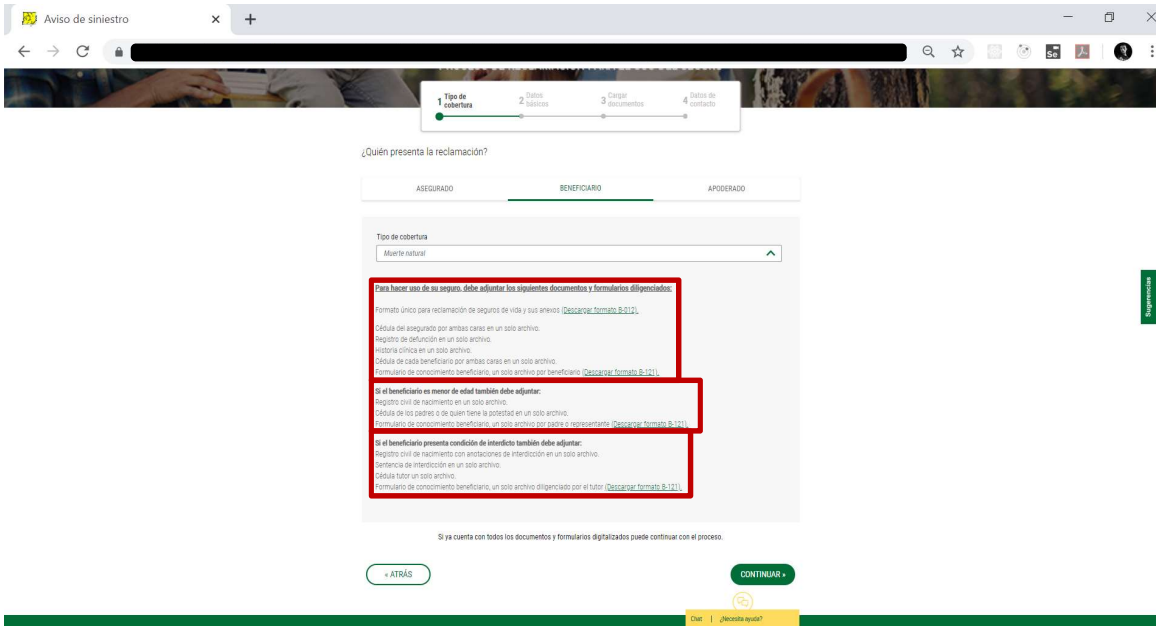


Figura 17. Comportamiento de la aplicación bajo un flujo con tipo de solicitud “Beneficiario” y tipo de cobertura “Muerte Natural”.

Como puede apreciarse en la figura 17, justo cuando se selecciona el tipo de cobertura se arroja el listado de documentos que serán necesarios para realizar la solicitud. A diferencia del tipo de solicitud de “Asegurado” en donde el listado se dividía en dos secciones (figura 10), para los tipos de cobertura de “Beneficiario” el listado se dividirá en tres secciones, la primera corresponderá a los documentos que siempre se tendrán en cuenta, la segunda para los documentos que se requerirán en caso de que quien realice la solicitud sea menor de edad, y la tercera para los documentos que se requerirán en caso de que quien realice la solicitud se encuentre en condición de interdicto.

Una vez mostrado el listado de documentos, se da clic en el botón “continuar” que mostrará el formulario de la figura 18:

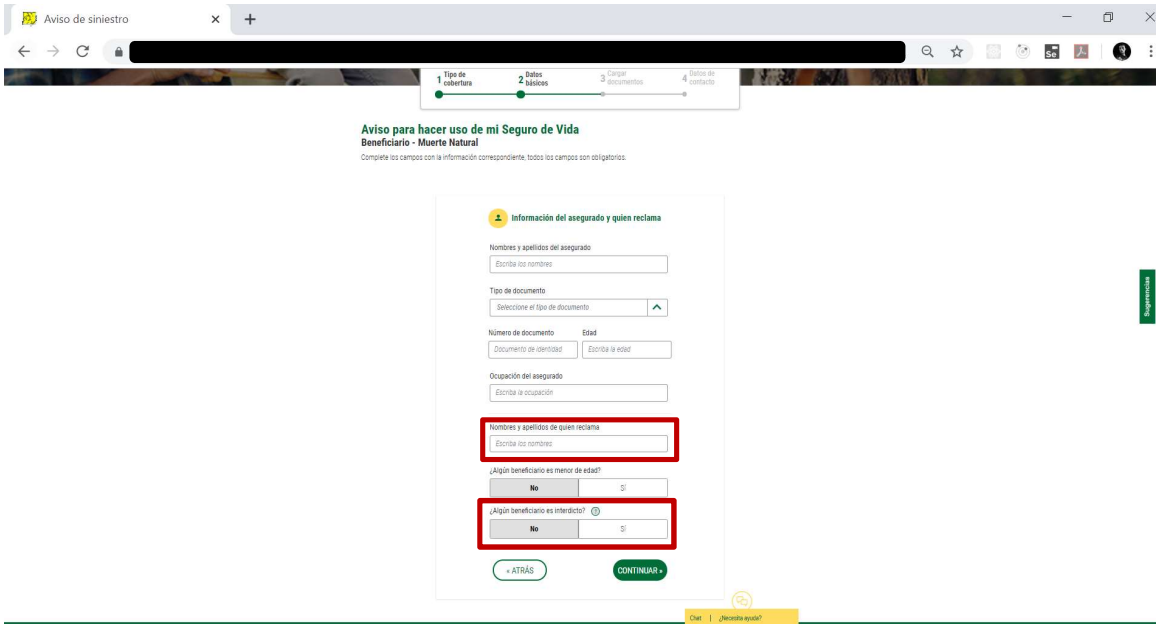


Figura 18. Formulario “Información del beneficiario”

El formulario de beneficiario cuenta con los mismos componentes que el formulario de asegurado (figura 11), pero con dos componentes más: un campo de entrada de texto en el cual se indica el nombre de la persona que está presentando la solicitud y un *radio-button* mediante el cual se indica si el beneficiario presenta o no condición de interdicto.

Entonces, a partir de este punto, el flujo podrá resultar en cuatro formas distintas: el beneficiario no es menor de edad ni presenta condición de interdicto, el beneficiario no es menor de edad, pero presenta condición de interdicto, el beneficiario es menor de edad, pero no presenta condición de interdicto, y el beneficiario es menor de edad y además presenta condición de interdicto. Por lo que al dar clic al botón “continuar” se mostrará la pantalla de carga de documentos, similar a la mostrada en la figura 12 o 13, con los espacios para cargar documentos correspondientes al flujo que se haya seguido, el campo de entrada de texto para describir la situación ocurrida y el botón de “continuar”,

que al darle clic mostrará la pantalla del formulario de datos de contacto mostrada en la figura 14, el cual al ser llenado y dar clic al botón “finalizar”, mostrará un pop-up comunicando el éxito (figura 15) o inconsistencia (figura 16) en el proceso de solicitud. Y retornará a la pantalla de inicio de “Aviso de siniestros” (figura 5).

- Para una solicitud de “Apoderado”:

Para este tipo de solicitud, es posible que el listado de documentos se presente de dos formas distintas dependiendo del tipo de cobertura que se seleccione: dividida en dos secciones como se muestra en la figura 10 o dividida en tres secciones como se muestra en la figura 17. El primer caso aplica para los tipos de cobertura “Enfermedad Grave”, “Gastos Médicos”, “Rentas”, “Incapacidad Total y Permanente” y “Desmembración” (ya que para estas no está la opción de que el apoderado presente o no condición de interdicto, por lo que no hay documentos que se requieran relacionados a esta situación), y el segundo caso aplica para los tipos de cobertura “Muerte Natural”, “Muerte Accidental” y “Suicidio” (ya que para estas sí aplica la condición de interdicto para el apoderado, por lo que se tienen que considerar otros documentos).

De igual manera, cuando se continúa con el flujo mediante el botón de “continuar”, se mostrará en pantalla el formulario “Información de apoderado”, el cual podrá tener dos interfaces distintas:

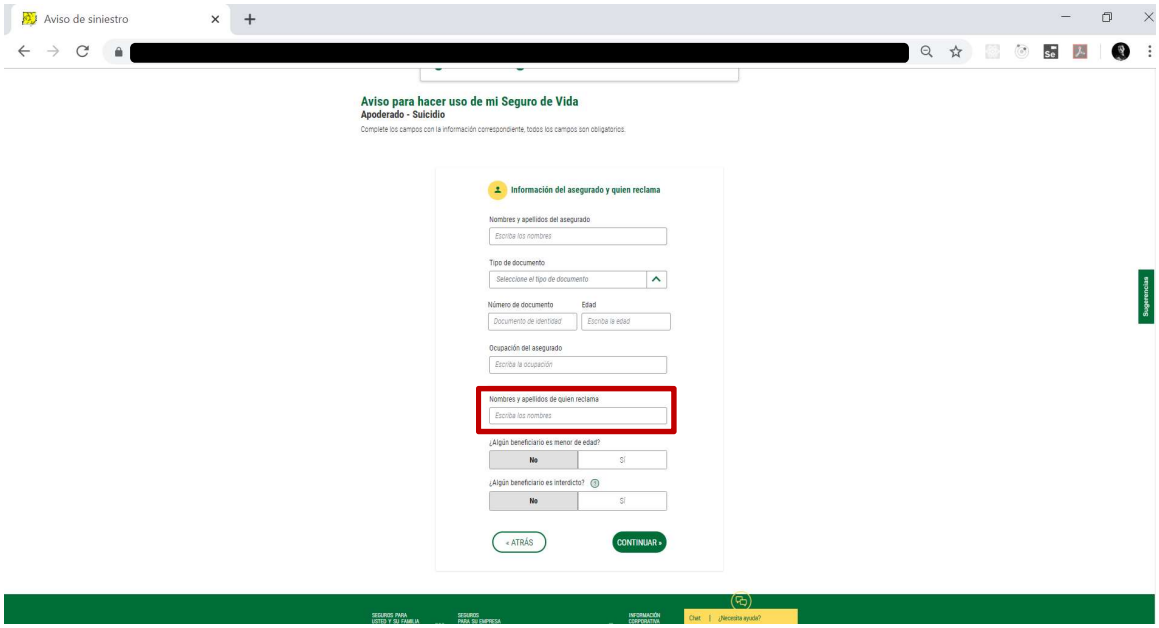


Figura 19. Formulario "Información de apoderado" para los tipos de cobertura "Enfermedad Grave", "Gastos Médicos", "Rentas", "Incapacidad Total y Permanente" y "Desmembración".

La primera presentación, mostrada en la figura 19, es bastante similar a la mostrada en la figura 11, pero como puede apreciarse, se adiciona el campo de entrada de texto para ingresar el nombre del apoderado.

El segundo tipo de interfaz, para los tres tipos de cobertura restantes, es igual al mostrado en la figura 18.

Una vez llenado el formulario y pulsado el botón "continuar" aparecerá la pantalla para cargar los documentos, similar a la mostrada en la figura 12 o 13, pero con los campos de los documentos correspondientes al flujo que se esté ejecutando en ese momento. Luego, continuando con el flujo, se mostrará el formulario de datos de contacto (figura 14), el cual, al llenarse, arrojará la respuesta exitosa (figura 15) o fallida (figura 16) en el proceso de solicitud y volverá a la página principal de "Aviso de siniestros" (figura 5).

8.2 CAPÍTULO 2: PLANTEAMIENTO DE LOS CASOS DE PRUEBA

Al no tener acceso al código fuente de la aplicación de indemnizaciones, los casos de prueba se plantearon basados en el modelo de caja negra.

- Casos de prueba para el front-end:

Estos casos de prueba se centran en la interfaz de usuario de la aplicación, es decir, lo que es visible para quienes la vayan a utilizar. Por lo que los casos de prueba se caracterizarán de la siguiente forma:

- ✓ Posicionamiento: Son pruebas que se realizan prácticamente de forma visual, en donde se verifica que la ubicación de todos los elementos que componen la aplicación sea correcta. Debe hacerse en todas las pantallas de la aplicación.
- ✓ Redireccionamiento: Son pruebas enfocadas a los botones de la aplicación que generarán un redireccionamiento de la página. Un ejemplo puede ser el botón “continuar” que redirecciona de la pantalla “Información del asegurado” a la pantalla de “Documentos del asegurado”; o el botón “atrás” que realiza el direccionamiento inverso.
- ✓ Validación de campos de entrada: Son las restricciones que se deben aplicar a los campos en los que el usuario deberá proporcionar determinada información. Las validaciones que se deben realizar dependen del campo en cuestión, algunos ejemplos son: para los campos de “Número de documento”, “Teléfono” y “Edad” sólo debería permitirse el ingreso de caracteres tipo numérico; los campos para cargar documentos

sólo permiten archivos con determinadas extensiones (archivos .jpg, .doc, .docx, .pdf, etc); la información ingresada en los campos de correo electrónico debe cumplir con la estructura de prefijo, seguido del caracter "@", seguido de un sufijo, después un "." y finalizado con caracteres tipo letra; que los campos para ingresar nombres, apellidos, ocupaciones y descripciones permitan el ingreso de todo tipo de caracter; entre otros. Además, debe validarse la obligatoriedad de los campos, es decir, que si uno o varios campos obligatorios están vacíos, no se pueda continuar con el flujo.

- ✓ Estado de los servicios: Se centra en los servicios externos a la aplicación, pero que están ligados a ella, estos servicios son: la visualización del listado de los documentos solicitados (servicio de Seguros Bolívar que se llama cuando se ha seleccionado el tipo de cobertura), la visualización de los componentes del campo "Tipo de Documento" y "Ciudad o Municipio de ocurrencia" (servicios de Seguros Bolívar que se llama cuando se da clic a la lista desplegable correspondiente) y el servicio de envío de la solicitud (que puede ser fallida o exitosa, por lo que deben considerarse ambos casos de prueba).

- Casos de prueba de tipo "responsive":

Son similares a los casos de prueba de front-end, pero se plantean para dispositivos móviles. No es necesario realizar las pruebas desde un dispositivo móvil, ya que es posible hacerlo desde un computador, manipulando la aplicación web desde su

ventana de comandos (ingresando con el comando CTRL + Shift + i) como se muestra en las figuras 20 y 21:

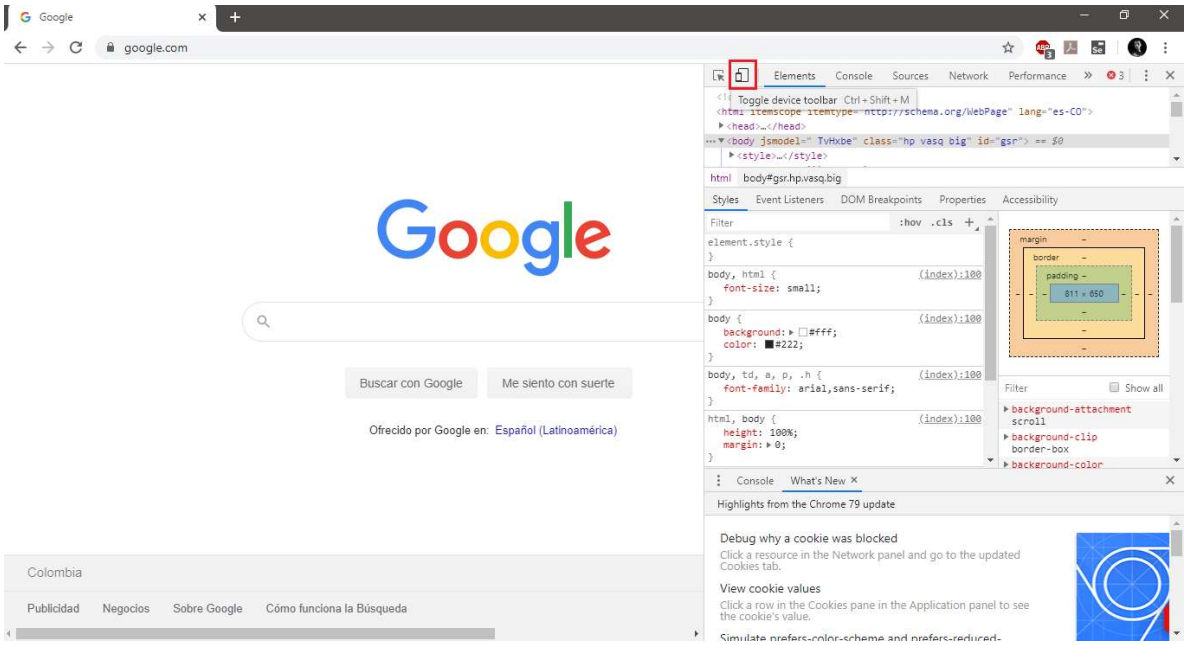


Figura 20. Ventana de comandos en el navegador Google Chrome.

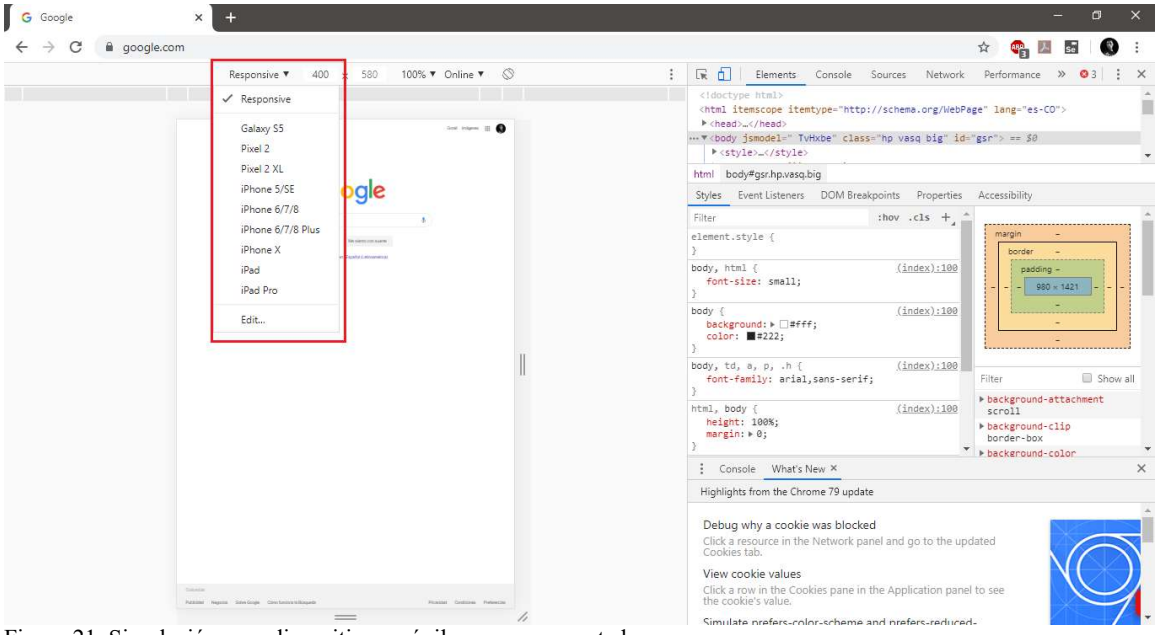


Figura 21. Simulación para dispositivos móviles en un computador.

- Casos de prueba para el back-end:

Estos casos de prueba se centran en verificar que los documentos que se cargan en el proceso de solicitud se almacenen correctamente en su registro correspondiente en base de datos. Un ejemplo de esto puede ser verificar que el archivo cargado en el campo “Registro de defunción” se almacene en base de datos en el registro “Registro de defunción”. Son de “back-end” porque el usuario que esté utilizando la aplicación solo se encargará de subir los documentos, pero no tendrá conocimiento sobre el registro en donde quedaron almacenados, ya que este es un proceso que no corresponde a la interfaz visual de usuario.

Algo que se debe recalcar es que, para ingresar a la base de datos, es necesario utilizar la VPN de Seguros Bolívar con un usuario y contraseña específicos. Cuando se esté conectado a ésta, solo se podrá acceder a la página de la base de datos, las demás páginas estarán bloqueadas.

Los casos de prueba que se formulen deben documentarse debidamente. Actualmente, IBM utiliza Microsoft Excel para cumplir con este fin. En la figura 22, se muestra el formato utilizado con el planteamiento de un caso de prueba:

IBM — Ejecución, Seguimiento — TAC

Fecha Inicio	10 de septiembre de 2019
Especialista(s) de Pruebas:	Jorge Chaparro Sarmiento
Solicitud Herramienta:	Indemnizaciones Seguros Bolivar
Aplicativo	FI
Fecha Fin	

Porcentaje Ejecución — 0% — 100% — Análisis, Bloqueado, Completado

ESTADOS	TOTAL	Porcenta
Análisis	0	0%
Bloqueado	0	0%
Completado	0	0%
Completado Obs.	0	0%
Error	0	0%
Fuera de alcance	0	0%
Pendiente	1	100%
Total	1	100%

Alcanc

ID	Nombre Caso de Prueba	Descripción	Resultado Esperado	Observaciones	Estado	Ciclo	Defecto	Estimad o
CP01	Verificar el posicionamiento de componentes en el Home	Estando en el home de la página, todos los componentes deben interactuar de la forma correcta	La ubicación de cada uno de los componentes es la correcta		Pendiente	1		

Casos de Pruebas TF | CP01

Figura 22. Formato para almacenar los casos de prueba planteados.

8.3 CAPÍTULO 3: PRUEBAS DE QA MANUALES

Las pruebas de QA no son más que la ejecución de los casos de prueba que se plantearon anteriormente, en este caso, de forma manual.

Suponiendo un caso de prueba que se haya formulado, el cual se centraba en comprobar el redireccionamiento de cierto botón, entonces la prueba de QA para este caso de prueba consistía en dar clic al botón correspondiente y verificar que se redireccione a la página deseada. Si esto sucede, el estado de la prueba de QA se consideraba como completado y entonces era posible cerrar el ciclo de pruebas. Si, por el contrario, el botón no aparecía, o redireccionaba a una página errónea, o se encontraba mal posicionado, o demás posibles defectos, el estado del caso de prueba se consideraría como bloqueado, fallido u otro, dependiendo de la situación, entonces se cerraría el primer ciclo de pruebas, adjuntando información tal como se muestra en la figura 23:

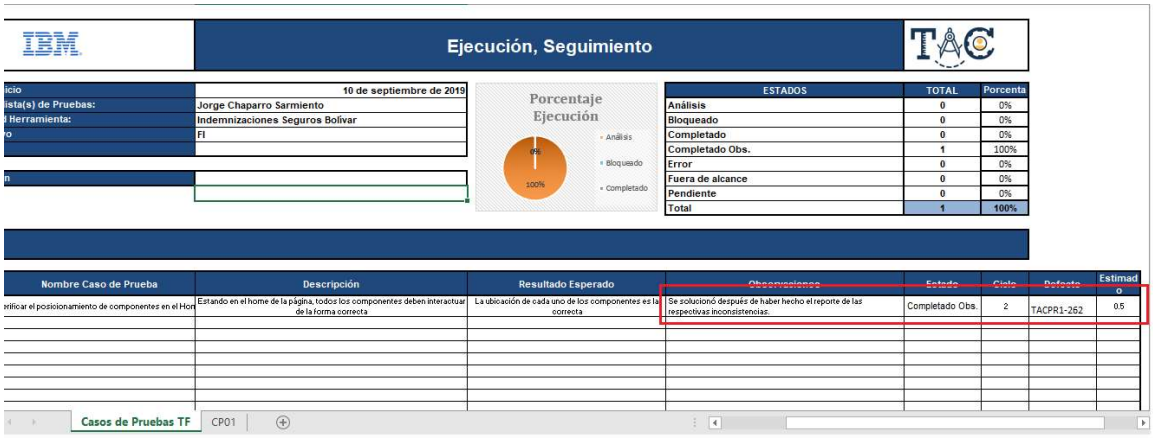


Figura 23. Documentación de ejecución de pruebas de QA.

La casilla de “observaciones” se destina a adjuntar comentarios muy breves sobre el estado de la prueba (cuál es el defecto, o si se solucionó después de reportar las fallas). En la casilla de “estado”, como su nombre lo indica, se fija la situación actual del caso de prueba (completado, completado con observaciones, fallido, bloqueado, cancelado, entre

otros). La casilla “ciclo” indica cuántas veces se ha ejecutado el caso de prueba. La casilla “defecto” indica el ticket de Jira (herramienta de seguimiento y gestión de proyectos utilizada por IBM) asociado al defecto (en caso de que exista) del caso de prueba. Por último, la casilla “Estimado” indicará el tiempo en horas que tomó realizar la ejecución y documentación del caso de prueba.

Claramente, si un caso de prueba se ejecuta satisfactoriamente en el primer ciclo de pruebas, no será necesario realizar alguna observación, el estado será “completado”, el ciclo será “1”, la casilla “defecto” estará vacía y el estimado será el tiempo de ejecución y documentación. Por el contrario, los casos de prueba que presenten defectos se deberán ejecutar cada vez que el desarrollador reporte novedades con respecto a éste (a menos de que se indique que ya no es necesario realizar más pruebas), bajo estas condiciones, sí será necesario dar observaciones (“se solucionó el defecto pero ahora falla ____”, “el desarrollador indica que presentó soluciones ante el defecto pero sigue fallando”, “se soluciona el defecto”, etc), el estado no podrá ser “completado”, evidentemente será más de un ciclo de pruebas, y habrá un ticket de Jira en “defecto”. El estimado será el tiempo total que se haya gastado en la ejecución de todos los ciclos de prueba para ese caso de prueba, y no solo en el último de ellos.

Además, deben presentarse las evidencias de la ejecución de cada caso de prueba como se muestra en la figura 24, para esto se destinan pestañas con el nombre “CPXX” en donde “XX” será el número asociado a cada caso de prueba:

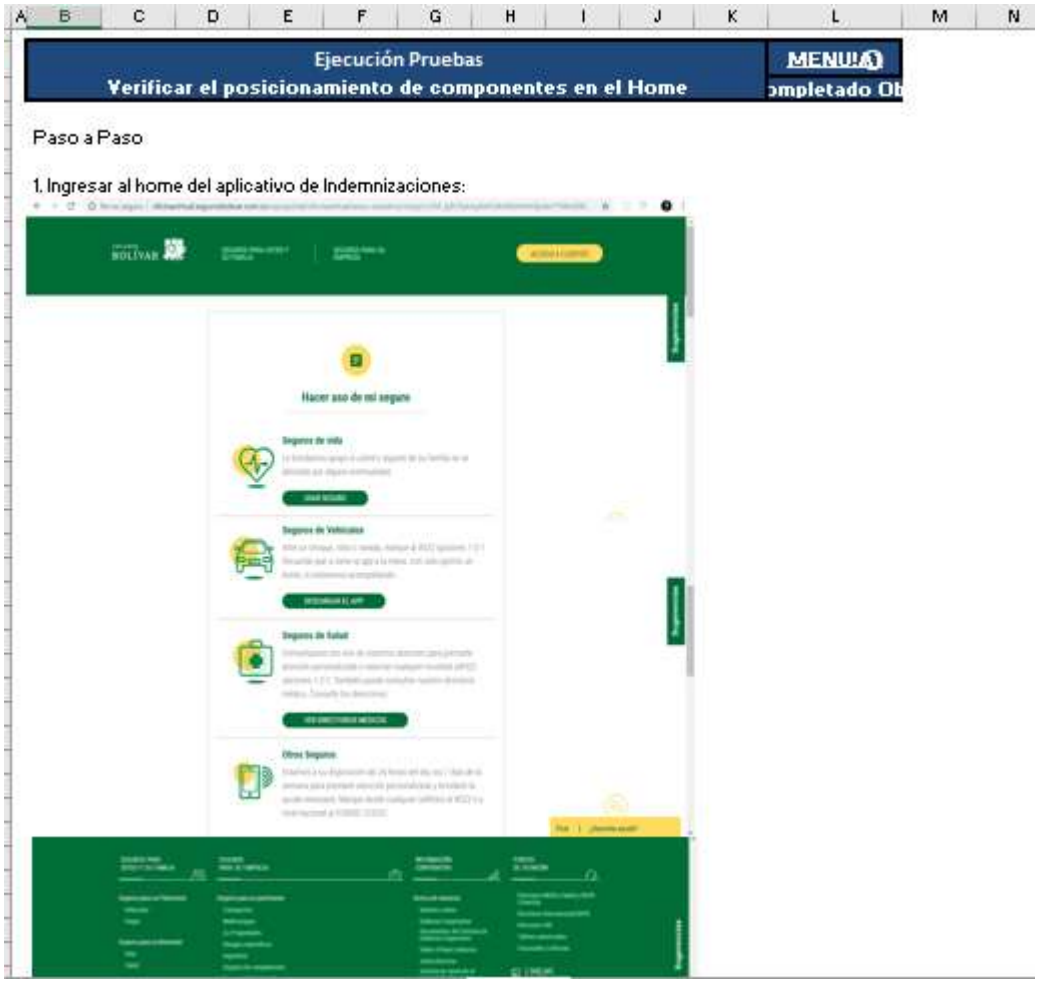


Figura 24. Evidencias de ejecución de casos de prueba.

Los defectos existentes, como ya fue mencionado anteriormente, deben reportarse en Jira, en donde es posible clasificarlos como bugs, tareas, subtareas, entre otros; se asignan a la persona encargada (desarrollador de la aplicación) para que genere una solución; se estima su gravedad y se describen de forma detallada las fallas; se fija su estado actual (pendiente, completado, cancelado, entre otros). En la figura 25, se muestra un ejemplo de un defecto reportado en Jira con las características mencionadas:

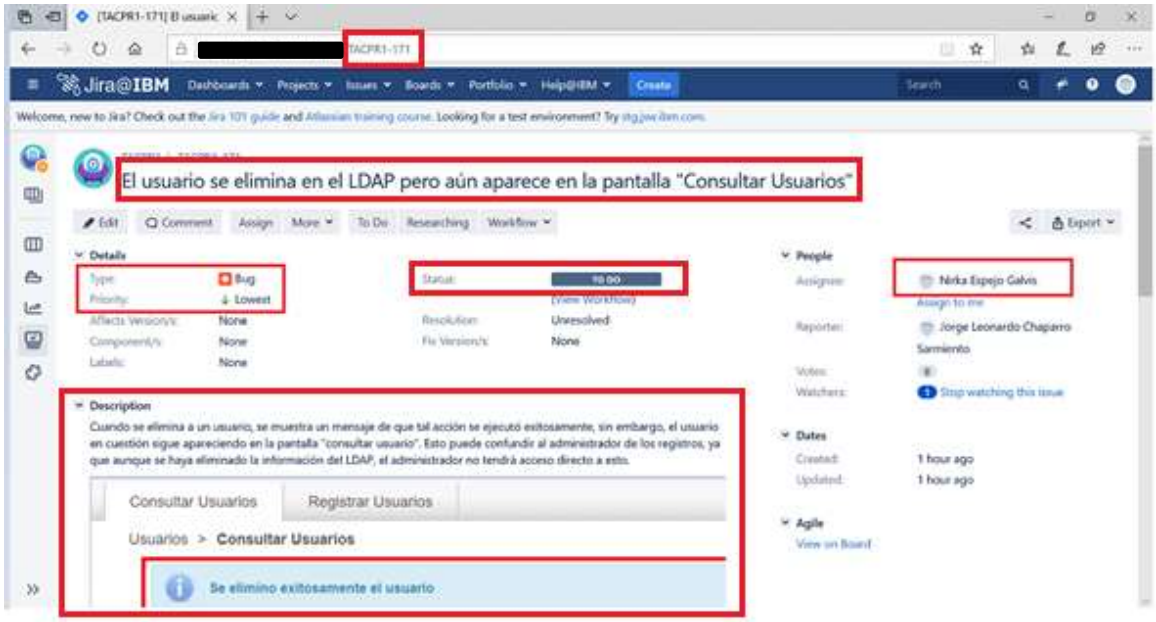


Figura 25. Reporte de un defecto en Jira.

Las características de estos tickets son modificadas bien sea por el desarrollador al cual fue asignado el defecto, o por la persona que creó el ticket.

Cada ciclo de pruebas de QA finaliza cuando todos los casos de prueba que lo componen se han ejecutado. Entonces, se reportan los defectos existentes y se queda a la espera de su solución para la ejecución de un segundo ciclo. Así, hasta que ya no existan defectos en la aplicación o hasta que se dé la orden de detener las pruebas.

Además, los documentos con los casos de prueba y las evidencias de su ejecución también deben cargarse a Jira, en el ticket que normalmente crea el PM del proyecto y es asignado a uno de los integrantes del equipo de TAC-CM-QA.

8.4 CAPÍTULO 4: ¿QUÉ CASOS DE PRUEBA SON AUTOMATIZABLES?

Los casos de prueba para el back-end no son automatizables, ya que como se menciona en la sección correspondiente, para acceder a la base de datos se requiere utilizar una VPN que bloquea las demás páginas, incluyendo herramientas de control de navegadores.

Los casos de prueba de tipo responsive tampoco son automatizables, ya que no hay herramientas de automatización que permitan controlar los navegadores de dispositivos móviles ni la consola del navegador en un computador (para simular un dispositivo móvil).

Por lo que los casos de prueba que se automatizarán serán los relacionados con el front-end para computadores.

Al no existir validaciones de tipo captcha en la aplicación, ni envío de OTPs (one time password), es posible automatizar los flujos de ésta. Estos flujos incluirán el llenado de información en campos de entrada de texto, selección de elementos en listas desplegables, carga de documentos, comprobación de existencia (no posicionamiento) de elementos y almacenamiento de información relevante (para verificar si el proceso de solicitud fue exitoso o fallido).

8.5 CAPÍTULO 5: DEFINICIÓN DE LA HERRAMIENTA DE AUTOMATIZACIÓN DE PRUEBAS

En la sección de marco teórico se presentaron distintas herramientas con las cuales es posible automatizar la ejecución de casos de prueba. Fue Selenium la que se

seleccionó entre ellas debido a sus características de ser Open Source, compatible con la mayoría de sistemas operativos y navegadores, pero más importante, debido a que su herramienta Selenium IDE permite la automatización de casos de prueba de una forma simple, sencilla y breve, ya que no requiere conocimiento técnico sobre lenguajes de programación; y su herramienta Selenium WebDriver, con la cual, la automatización se puede llevar a un nivel más complejo, pero a su vez, más eficiente, sin mencionar la posible vinculación de los códigos generados con aplicaciones y funcionalidades ajenas a Selenium.

8.6 CAPÍTULO 6: AUTOMATIZACIÓN DE CASOS DE PRUEBA CON SELENIUM IDE

Como se mencionó en secciones anteriores, lo primero que se debe hacer al ingresar a Selenium IDE es crear un nuevo proyecto, en este caso, con el nombre de "SB_Indemnizaciones". Entonces, se debe ingresar la URL en donde los casos de prueba se originan, para lo cual se utiliza el cuadro de entrada de texto mostrado en la figura 26:

Figura 26. Cuadro de entrada de texto para ingresar la URL.

Una vez hecho esto, será posible comenzar a agregar casos de prueba, dando clic al símbolo "+" ubicado a la derecha de la sección "Tests" mostrada en la figura 26. Tal y como se muestra en la figura 27, s debe asignar un nombre al caso de prueba:

Figura 27. Pantalla de asignación de nombre para el caso de prueba. Estructura del nombre de los casos de prueba.

El caso de prueba se habrá creado satisfactoriamente, y lo que procede, es comenzar a grabar (literalmente) los pasos que lo componen. Para esto, se debe dar clic al botón "Record" mostrado en la figura 28:

Figura 28. Botón para comenzar a agregar los pasos del caso de pruebas.

Al momento de hacer clic en el botón mencionado, el navegador Google Chrome se abrirá, redirigiendo a la página cuyo URL ingresado (figura 26) corresponde. Desde entonces, toda acción (clic, movimiento de ratón, carga de archivos, llenado de campos con información) que realice el usuario de forma manual será guardada por Selenium IDE. Cuando se hayan ejecutado manualmente todas las acciones correspondientes al caso de pruebas, se debe volver a Selenium IDE y detener la grabación de los pasos mediante el botón mostrado en la figura 29:

Figura 29. Botón para detener la grabación de los casos de prueba.

Entonces, en la sección del caso de prueba mostrada en la figura 30, se muestran todos los pasos que se realizaron y se grabaron:

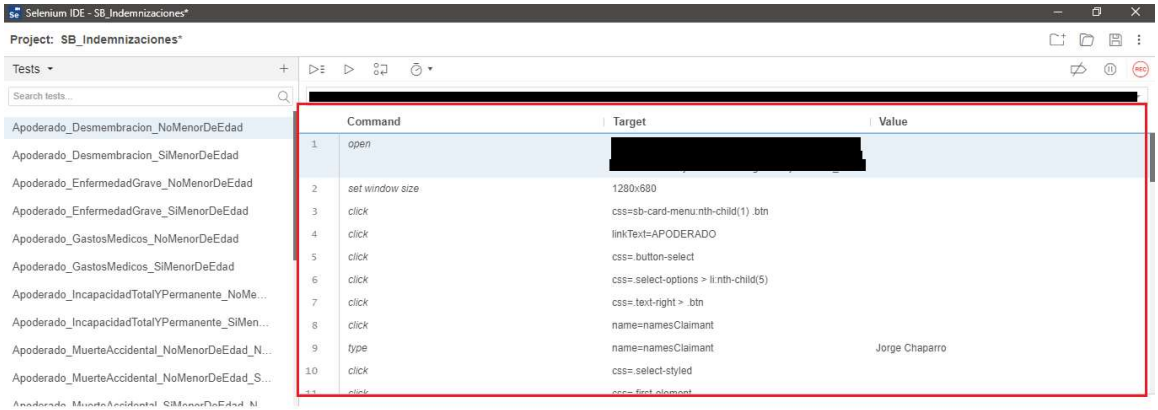


Figura 30. Pasos de casos de prueba.

Los pasos que se graban poseen cierta estructura, y esta depende del tipo de comando que se utilice. A continuación, se muestran algunos de ellos:

Figura 31. Estructura para los pasos de comando “Open”.

El comando “Open”, mostrado en la figura 31, se encarga de redirigir a la página cuya URL se ingresa en el campo “target”.

Figura 32. Estructura para los pasos de comando “Set window size”.

El comando “set window size”, mostrado en la figura 32, se encarga de configurar la pantalla de Google Chrome a la resolución indicada en el campo “target”.

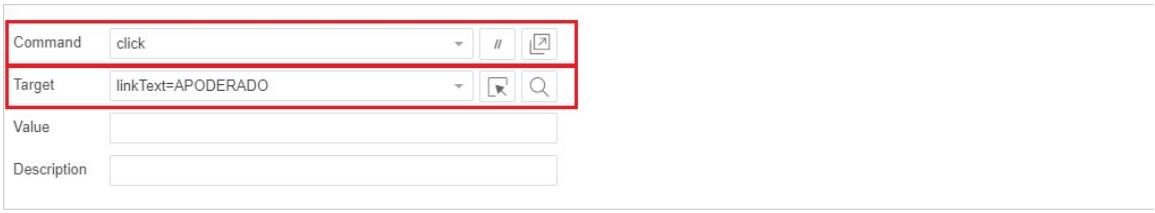


Figura 33. Estructura para los pasos de comando “Click”.

El comando “click”, mostrado en la figura 33, como su nombre lo indica, da un clic al elemento de la página web correspondiente al identificador que se ingresa en el campo “target”.

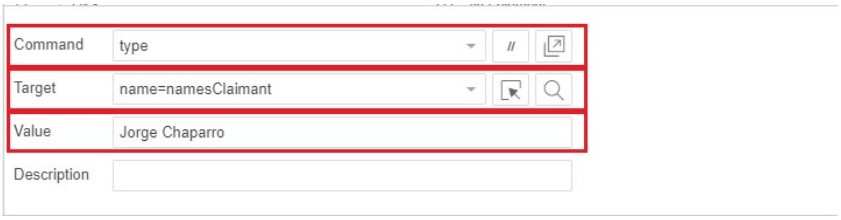


Figura 34. Estructura para los pasos de comando “Type”.

El comando “Type”, mostrado en la figura 34, se encarga de poner en el elemento (debe ser un campo de entrada) cuyo identificador se ingresa en el campo “target”, la información indicada en el campo “value”.

Los pasos que se grabaron también pueden modificarse de forma manual, esto es especialmente útil cuando se requiere eliminar acciones que no son de importancia para la ejecución automatizada del caso de prueba. Estas acciones son:

- Movimientos del ratón: Eliminar estos pasos no afecta a la funcionalidad de la automatización del caso de pruebas.
- Clics sobre campos de texto: Cuando se llena información en un campo de texto de forma manual, el usuario primero da clic sobre el campo, y luego sí completa la información correspondiente. Sin embargo, de forma automática, el clic sobre

el campo es irrelevante, ya que directamente puede ingresarse la información. Por lo que este paso se vuelve innecesario.

- Clics sobre campos para cargar documentos: Es similar a lo que ocurre con los campos de texto, pero en este caso, el archivo puede cargarse sin necesidad de hacer clic en el campo, por lo que el paso se vuelve innecesario.

Además de eliminar pasos, también es necesario agregar algunos otros de forma manual. Los pasos que deben agregarse son:

- Verificación de texto: Comprueban la existencia de los campos para cargar documentos, y que estos coincidan con los documentos que se solicitan:

Figura 35. Acciones de verificación.

Para esto se utiliza el comando “assert text”, el cual, según la figura 35, compara el texto que existe en el elemento de la página web cuyo identificador se indica en el campo “target” con el texto que se indica en el campo “value”. Si son iguales, el proceso continúa, pero si existen diferencias, se muestra un mensaje en la sección de resultados que indica el error.

- Carga de documentos: Se encargan de adjuntar los archivos en los campos correspondientes: Aunque una parte de esta acción se agregue de forma automática por Selenium IDE cuando almacena los pasos grabados, se debe

indicar la ruta y el nombre del archivo que se cargará, ya que estas variables, al ser pertenecientes al equipo y no a la página web, no son accesibles mediante Selenium.

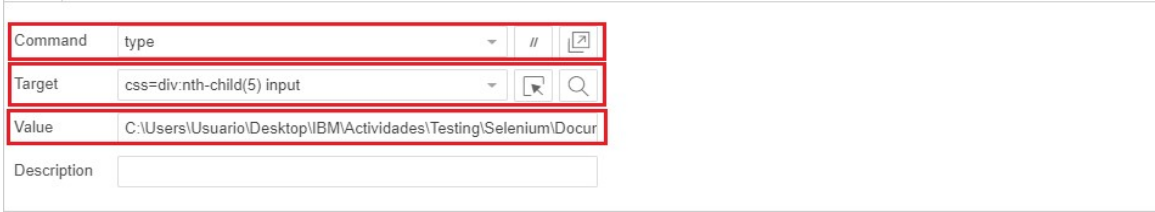


Figura 36. Acciones de carga de documentos.

Al igual que para llenar campos de texto con información, se utiliza el comando "type", pero ahora, en el campo "value" se debe indicar la ruta y el nombre del archivo que se desea cargar, tal como se muestra en la figura 36.

Además, en el navegador Google Chrome (tomando la ruta: configuración - extensiones - Selenium IDE – Detalles) debe habilitarse (como se muestra en la figura 37) el permiso para que Selenium IDE acceda a los archivos guardados en el ordenador:

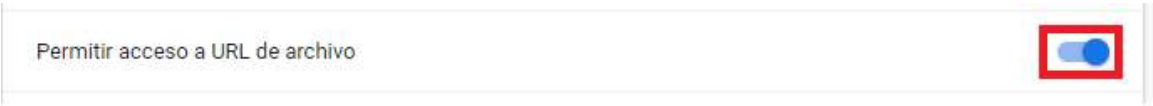


Figura 37. Habilitación de acceso a archivos locales.

Esta opción aparece desactivada por defecto. En tal estado, durante la ejecución del caso de prueba, cuando se llegue al comando que busca el archivo local correspondiente, el proceso se detendrá e indicará el error asociado.

- Almacenamiento de texto: Se encarga de guardar el texto que se muestra en determinado elemento de la página (útil para almacenar la información que comunica si la solicitud fue exitosa o fallida).

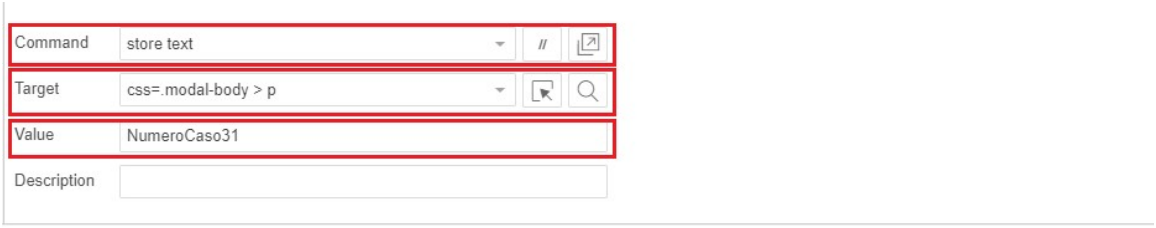


Figura 38. Acciones de almacenamiento de texto.

Se utiliza el comando “store text” mostrado en la figura 38, el cual almacena en la variable cuyo nombre se indica en el campo “Value”, el texto que aparece en el elemento con identificador indicado en “target”.

- Mostrar información almacenada: Se encarga de mostrar por pantalla la información almacenada en alguna variable.

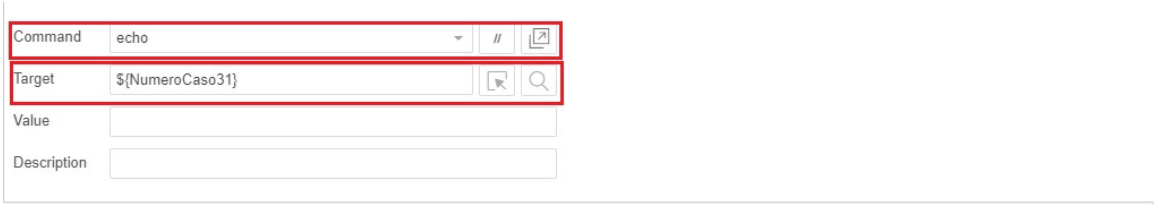


Figura 39. Acción de visualización de información.

Se utiliza el comando “echo” mostrado en la figura 39, el cual muestra en la sección de resultados la información almacenada en la variable indicada en el campo “target” (Por convención y estructura, el campo “target” debe llenarse como: *${NombreVariable}*).

Cuando los pasos del caso de prueba estén listos, podrán reproducirse las veces que se quiera mediante el botón resaltado en la figura 40:

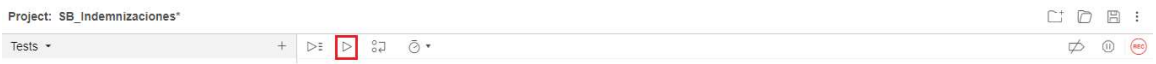


Figura 40. Botón para reproducir el caso de prueba de forma automática.

El proceso mencionado anteriormente debe repetirse para los casos de prueba en cuestión, en este caso, los cuarenta y cuatro flujos para la aplicación. Los casos de prueba se muestran en la parte izquierda de la pantalla de Selenium IDE, como se muestra en la figura 41:

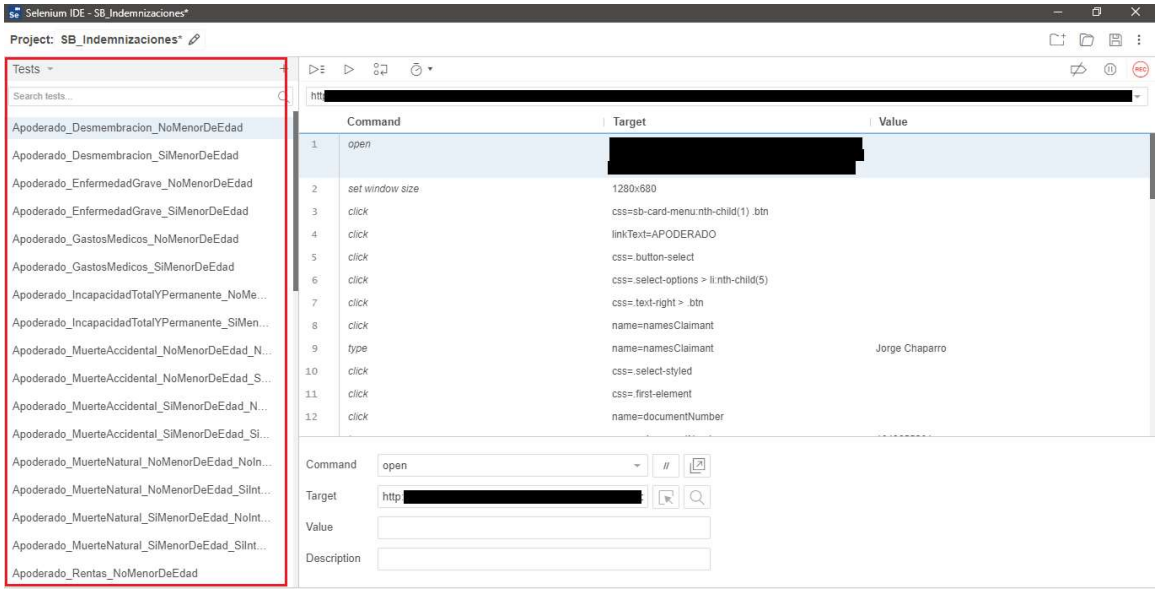


Figura 41. Casos de prueba automatizados con Selenium IDE.

Cuando existe más de un caso de pruebas automatizado, es posible ejecutarlos todos secuencialmente mediante el botón resaltado en la figura 42:

Figura 42. Botón para ejecutar la totalidad de casos de prueba de forma automática.

De forma que, con un solo clic, se podrán testear los cuarenta y cuatro distintos flujos de la aplicación “Indemnizaciones”.

8.7 CAPÍTULO 7: AUTOMATIZACIÓN DE CASOS DE PRUEBA CON SELENIUM WEB-DRIVER

WebDriver es una herramienta de Selenium mediante la cual se automatizarán los casos de pruebas que se han venido tratando hasta el momento, pero esta vez, mediante código de programación. Estos códigos se desarrollarán en el IDE (entorno de desarrollo integrado) de Eclipse.

Inicialmente, como se muestra en la figura 43, desde Eclipse se creará un proyecto de tipo Maven:

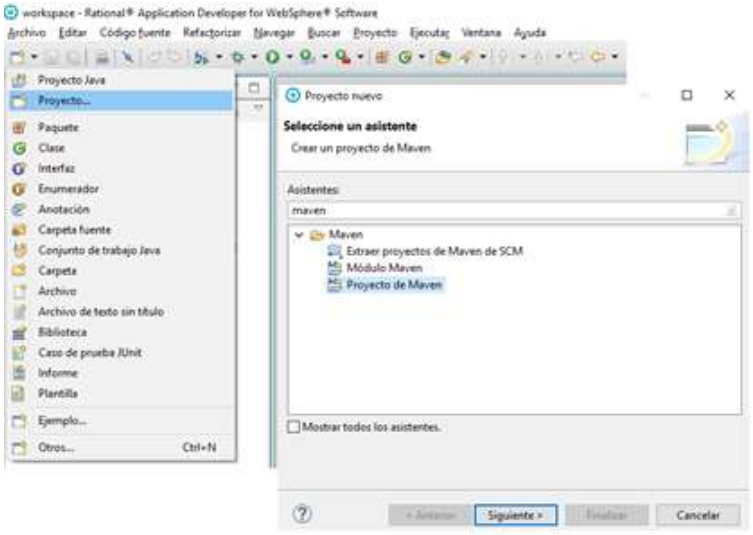


Figura 43. Creación de proyecto Maven en Eclipse.

Se debe asignar un nombre al proyecto (“PruebasSegurosBolivar” en este caso), y una vez creado, se obtendrá la estructura mostrada en la figura 44:

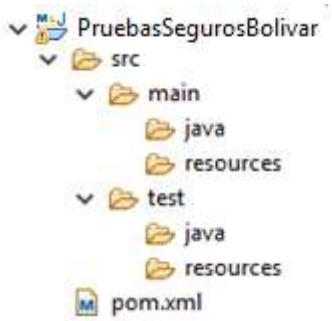


Figura 44. Estructura del proyecto Maven.

Para que los scripts (códigos) de automatización se ejecuten, se deben agregar al archivo pom.xml las dependencias de Selenium Web Driver [19] y Junit [20].

Además, es necesario descargar Chrome Driver [21] con el fin de que el navegador Google Chrome se pueda controlar desde Eclipse. Para que el proyecto pueda acceder a Chrome Driver, se da clic derecho en la carpeta "src/test/resources", y allí se crea una carpeta llamada "Chromedriver", sobre esta carpeta se hace clic derecho y se selecciona la opción importar>General>File System y allí se busca el ejecutable descargado. Debería aparecer como se muestra en la figura 45:

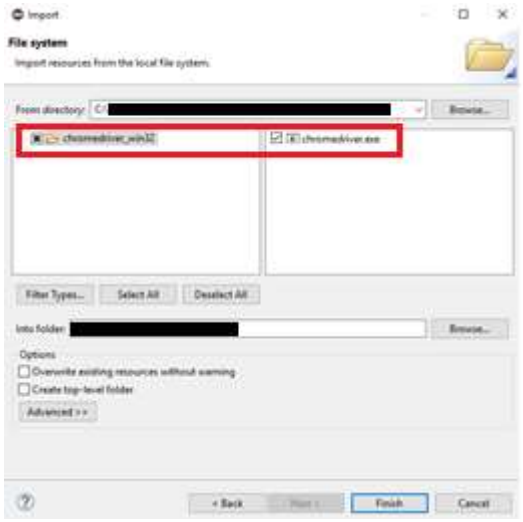


Figura 45. Configuración del archivo "Chrome Driver" en Eclipse.

En este punto, ya se cuenta con las dependencias y configuraciones necesarias para ejecutar un script de automatización con Selenium Web Driver.

Entonces, deben comenzar a generarse los códigos para cada uno de los cuarenta y cuatro casos de prueba. Selenium IDE contribuye también en este aspecto, ya que permite exportar en forma de código de programación los comandos ya almacenados.

Al hacer clic derecho sobre alguno de los casos de prueba realizados, se muestran las opciones de la figura 46, allí se selecciona “export”:

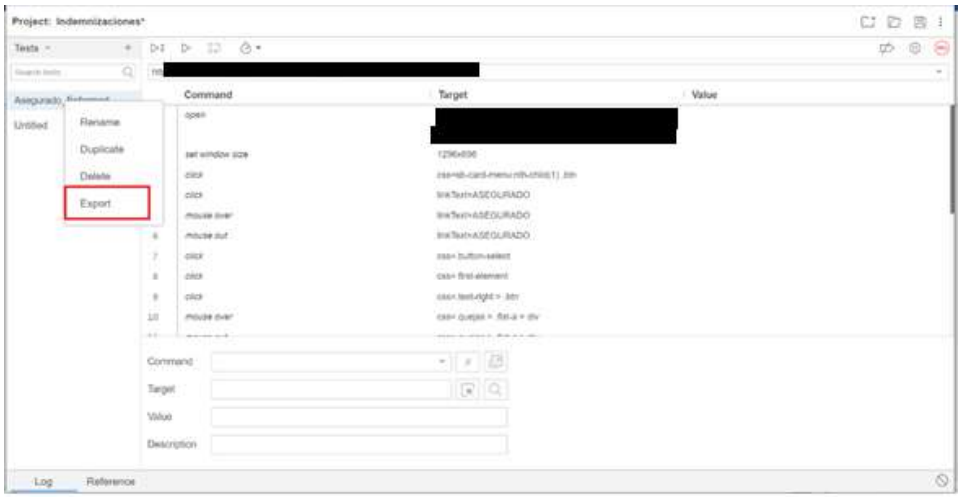


Figura 46. Generación de scripts de automatización con Selenium IDE (paso 1).

Posteriormente, como se muestra en la figura 46 se abrirá una ventana, mediante la cual se selecciona el lenguaje de programación en el que se generará el código (para este caso, se utilizará Java JUnit):

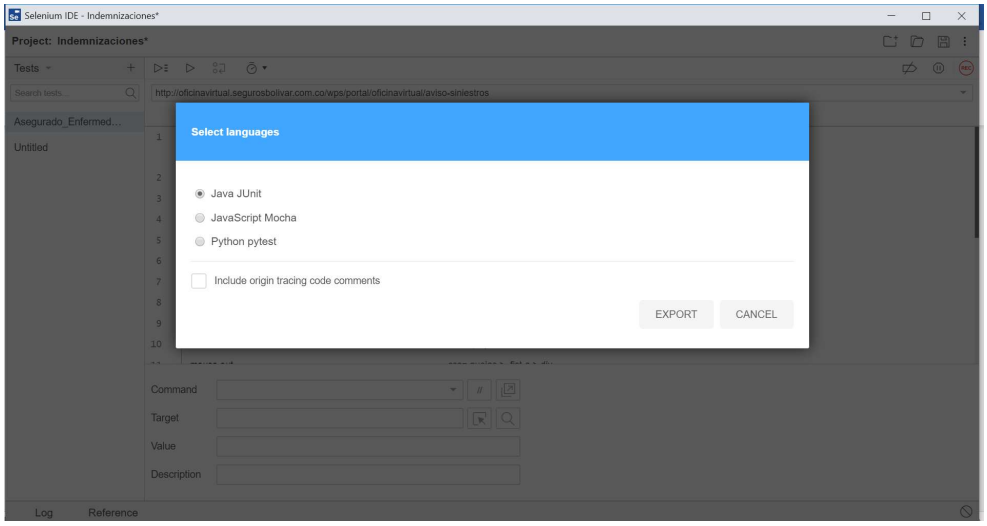


Figura 47. Generación de scripts de automatización con Selenium IDE (paso 2).

Una vez seleccionado el lenguaje de programación deseado, y dado clic en "export", se creará un archivo .java (en caso de que esta haya sido la elección) con el nombre del caso de prueba en cuestión (como se muestra en la figura 48), que puede abrirse desde cualquier editor de texto:

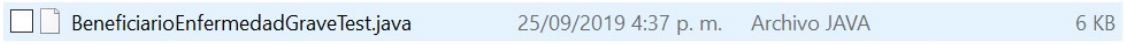
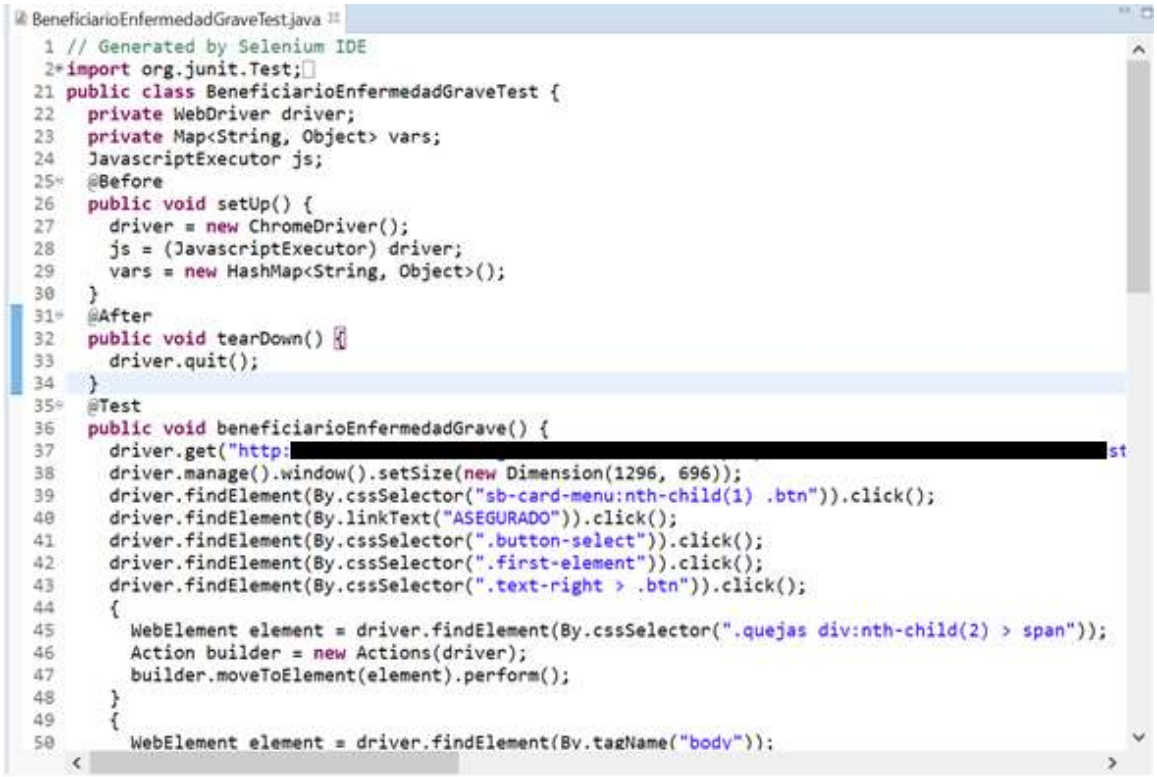


Figura 48. Archivo del script exportado desde Selenium IDE.

Al abrirlo desde eclipse, se mostrará algo como lo siguiente:



```
BeneficiarioEnfermedadGraveTest.java
 1 // Generated by Selenium IDE
 2 import org.junit.Test;
21 public class BeneficiarioEnfermedadGraveTest {
22   private WebDriver driver;
23   private Map<String, Object> vars;
24   JavascriptExecutor js;
25   @Before
26   public void setUp() {
27     driver = new ChromeDriver();
28     js = (JavascriptExecutor) driver;
29     vars = new HashMap<String, Object>();
30   }
31   @After
32   public void tearDown() {
33     driver.quit();
34   }
35   @Test
36   public void beneficiarioEnfermedadGrave() {
37     driver.get("http:[redacted]st
38     driver.manage().window().setSize(new Dimension(1296, 696));
39     driver.findElement(By.cssSelector("sb-card-menu:nth-child(1) .btn")).click();
40     driver.findElement(By.linkText("ASEGURADO")).click();
41     driver.findElement(By.cssSelector(".button-select")).click();
42     driver.findElement(By.cssSelector(".first-element")).click();
43     driver.findElement(By.cssSelector(".text-right > .btn")).click();
44     {
45       WebElement element = driver.findElement(By.cssSelector(".quejas div:nth-child(2) > span"));
46       Action builder = new Actions(driver);
47       builder.moveToElement(element).perform();
48     }
49     {
50       WebElement element = driver.findElement(By.tagName("body"));
```

Figura 49. Script de automatización exportado desde Selenium IDE (parte 1).

```
BeneficiarioEnfermedadGraveTest.java
50       WebElement element = driver.findElement(By.tagName("body"));
51       Action builder = new Actions(driver);
52       builder.moveToElement(element, 0, 0).perform();
53     }
54     driver.findElement(By.name("namesClaimant")).click();
55     driver.findElement(By.name("namesClaimant")).sendKeys("Jorge Chaparro");
56     driver.findElement(By.cssSelector(".select-styled")).click();
57     driver.findElement(By.cssSelector(".first-element")).click();
58     driver.findElement(By.name("documentNumber")).click();
59     driver.findElement(By.name("documentNumber")).sendKeys("1049655201");
60     driver.findElement(By.name("age")).click();
61     driver.findElement(By.name("age")).sendKeys("20");
62     driver.findElement(By.name("ocupation")).click();
63     driver.findElement(By.name("ocupation")).sendKeys("Pasante");
64     driver.findElement(By.cssSelector(".text-right > .btn")).click();
65     {
66       WebElement element = driver.findElement(By.linkText("Modelo de Sostenibilidad"));
67       Action builder = new Actions(driver);
68       builder.moveToElement(element).perform();
69     }
70     {
71       WebElement element = driver.findElement(By.tagName("body"));
72       Action builder = new Actions(driver);
73       builder.moveToElement(element, 0, 0).perform();
74     }
75     driver.findElement(By.cssSelector("div:nth-child(2) > sb-file .file")).click();
76     driver.findElement(By.name("file")).sendKeys("C:\\Users\\JorgeChaparroSarmien\\Desktop\\Activic
77     driver.findElement(By.cssSelector("div:nth-child(3) .file")).click();
78     driver.findElement(By.cssSelector("div:nth-child(3) input")).sendKeys("C:\\Users\\JorgeChaparrc
79     driver.findElement(By.cssSelector("div:nth-child(4) > sb-file .file")).click();
80     driver.findElement(By.cssSelector("div:nth-child(4) > sb-file input")).sendKeys("C:\\Users\\Jor
81     driver.findElement(By.cssSelector("div:nth-child(5) .file")).click();
```

Figura 50. Script de automatización exportado desde Selenium IDE (parte 2).

Aunque el código que se muestra en las figuras 49 y 50 funciona, se le realizarán modificaciones para hacerlo más corto, más entendible y más eficiente, en pocas palabras, es un código que debe optimizarse.

Inicialmente, para hacer que estos scripts sean más "óptimos", se crearán ciertas clases en el entorno de Eclipse, las cuales llevarán a cabo funciones específicas y estarán ubicadas en un "paquete" dentro de la ruta *src/test/java,* como se muestra en la figura 51:

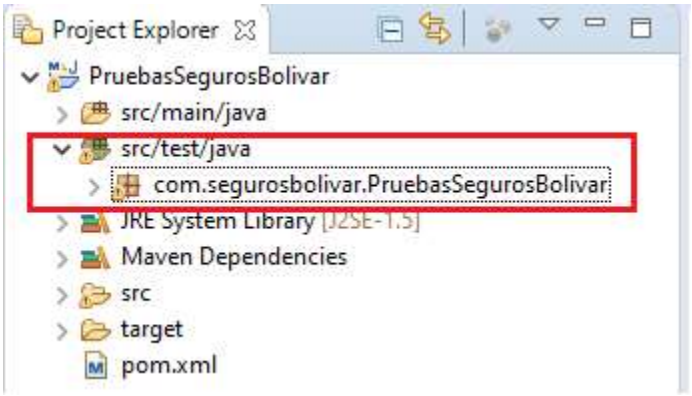


Figura 51. Paquete en el cual se crearán las clases del proyecto.

Se creará un total de 46 clases. Estas se clasificarán en tres tipos:

- Clase padre: En Java, existe una propiedad llamada "herencia", que consiste en que clases hijas de un proyecto, utilicen métodos que se definen una única vez en la clase padre. Esto es útil, ya que evita definir los métodos en cada una de las clases hijas. Por ende, la clase padre del proyecto en cuestión almacenará los métodos que serán requeridos para los procesos de automatización de los cuarenta y cuatro casos de prueba.
- Clases hijas: Son clases que utilizarán métodos que se definieron en la clase padre. El proyecto tendrá un total de cuarenta y cuatro clases hijas, cada una

destinada a uno de los cuarenta y cuatro distintos flujos automatizables de la aplicación “Indemnizaciones”.

- Clase de ejecución: Es la clase en la cual se vincularán la clase padre con las clases hijas, y ejecutará los casos de prueba de forma automatizada.

Al haber comprendido los tipos de clase que se tratarán en el proyecto, se procederá con la creación de las mismas. Inicialmente, con la clase padre, a la cual se le dará el nombre de “Base”. A continuación, se muestra un pequeño fragmento de su código:

*Base.java

```
1  package com.segurosbolivar.PruebasSegurosBolivar;
2  import java.io.File;
19
20 public class Base {
21     private static WebDriver driver;
22     private JavascriptExecutor js;
23     private Wait<WebDriver> wait;
24
25     public Base(WebDriver driver, JavascriptExecutor js) {
26         this.driver = driver;
27         this.js = js;
28     }
29
30     public WebDriver chromeDriverconnection() {
31         System.setProperty("webdriver.chrome.driver","./src/test/resources/Chromedriver/chromedriver.exe");
32         driver = new ChromeDriver();
33         driver.manage().timeouts().implicitlyWait(60, TimeUnit.SECONDS );
34         js = (JavascriptExecutor) driver;
35         wait = new FluentWait<WebDriver>(driver)
36                 .withTimeout(Duration.ofSeconds(60))
37                 .pollingEvery(Duration.ofSeconds(3));
38         return driver;
39     }
40
41     public WebElement findElement(By locator) {
42         return driver.findElement(locator);
43     }
44
45     public String getText(By locator) {
46         return driver.findElement(locator).getText();
47     }
48
49     public void sendkeys(String inputText, By locator) {
50         driver.findElement(locator).sendKeys(inputText);
51     }
```

Figura 52. Estructura de la clase padre “Base”.

Como puede apreciarse en la figura 52, el código se dividió en 4 secciones, la primera de ellas, que comprende la línea de código 1, simplemente se encarga de definir el paquete al cual pertenece la clase (el paquete fue definido en la figura 51). La segunda

sección, que comprende las líneas de código de la 2 a la 18 (se encuentran minimizadas), se encarga de llamar a las librerías de WebDriver y JUnit que serán utilizadas posteriormente. La tercera sección, que va de la línea de programación 20 a la 39, establece los estándares que se utilizarán en el proyecto (como la ruta de chromedriver y tiempos de espera implícitos), además de instanciar objetos (*driver* y *js*) de distintos tipos (*WebDriver* y *JavascriptExecutor*). La cuarta y última sección del código, que comprende desde la línea 41, está destinada a la creación de todos los métodos que se utilizarán en las clases hijas. Los métodos que se crearon, se hicieron con el fin de programar de una forma más simple en las clases hijas. Para sustentar esto, supongamos que se quiere escribir el texto "hola mundo" en un elemento de la página web que esté ligado al identificador "campo_de_texto_entrada_1". Como se pudo ver en la figura 50, esto se tendría que realizar mediante el siguiente código:

driver.findElement(By.cssSelector("campo_de_texto_entrada_1")).sendKeys("Hola mundo");

Sin embargo, con la creación del método asociado a la escritura de texto en un campo (línea 49, 50 y 51 de la figura 52), el código que se requiere es entonces el siguiente:

sendkeys("Hola mundo", campo_de_texto_entrada_1);

Se crean métodos que realicen los funciones de hacer clic, encontrar cierto elemento, almacenar texto, verificar si se muestra cierto elemento, redirigirse a una URL, desplazar la pantalla hasta determinado pixel, maximizar o cerrar la ventana, entre otros.

Esto, además de hacer mucho más sencilla la programación de las clases hijas, también es una herramienta preventiva, ya que si por algún motivo, en alguna versión futura de Selenium, la función que escribe determinado texto en un campo de la página web se ve modificada, en la programación solamente tendrá que cambiarse el método creado en la clase "Base", así, el código desarrollado en las clases hijas no se verá afectado (si no existiera el método en la clase "Base", se tendrán que aplicar modificaciones en cada sección del código en donde se haya utilizado la función en cuestión).

Al haber creado la clase padre, se desarrollarán las clases hijas, es decir, las clases destinadas a los flujos de la aplicación. Estas serán un poco más extensas, ya que como se ha mencionado antes, los flujos constan de llenar formularios, cargar documentos, y verificar elementos de la página web. A continuación, se muestran las secciones más importantes del código asociado a uno de los flujos (el flujo es el de Tipo de solicitud "Asegurado", tipo de cobertura "Enfermedad Grave" y el beneficiario no es menor de edad):

Figura 53. Estructura de las clases hija, parte 1.

En la figura 53 se muestra el código dividido en sus primeras cuatro secciones, al igual que en casos anteriores, en la primera de ellas se menciona el paquete en el cual se encuentra la clase, la segunda llama a las librerías que serán necesarias en el resto del programa. La tercera sección ya es distinta, es en ésta en donde se utiliza la propiedad de la herencia, ya que mediante la función *extends*, la clase en cuestión (*AseguradoEnfermedadGraveNoMenorDeEdadTest*) hace uso de los métodos y estándares que fueron definidos en la clase *Base*. En la cuarta sección, comprendida desde la línea 18, se crearán y asignarán valor a las variables que se usarán posteriormente; estas variables se ligan a los identificadores asociados a los elementos de la página web.

AseguradoEnfermedadGraveNoMenorDeEdadTest.java

```
51
52  public void procesoSolicitud() {
53      get("[redacted]");
54      click(botonSegurosVida);
55      click(tabAsegurado);
56      scroll(listaCovertura);
57      click(listaCovertura);
58      click(enfermedadGrave);
59      scroll(botonContinuar1);
60      click(botonContinuar1);
61      sendkeys("Jorge Chaparro", nombreAsegurado);
62      click(listaTipoDocumento);
63      click(cedula);
64      sendkeys("1049654213", numeroDocumento);
65      sendkeys("23", edad);
66      sendkeys("Pasante", ocupacion);
67      click(botonNoMenorEdad);
68      click(botonContinuar2);
69
70      String textDoc1 = getText(findElement(By.cssSelector("div:nth-child(2) > sb-file label:nth-child(1)")));
71      String textoDoc1 = "Formato Único para reclamación de seguros de vida y sus anexos(B-012) (Un solo archivo)";
72      if(textDoc1.equals(textoDoc1))
73      {
74          System.out.println("El documento 1 coincide");
75      }
76      else
77      {
78          System.out.println("El documento 1 no coincide");
79      }
80      assertEquals(textoDoc1, textDoc1);
81      sendkeys("C:\\\\Users\\\\Usuario\\\\Desktop\\\\IBM\\\\Actividades\\\\Testing\\\\Selenium\\\\Documentos de Prueba\\\\1.docx", document
```

Figura 54. Estructura de las clases hija, parte 2.

AseguradoEnfermedadGraveNoMenorDeEdadTest.java

```
134
135  sendkeys("jeiutra 54y5 _:_ASGWR", describirSituacion);
136  click(botonContinuar3);
137  click(listaCiudad);
138  click(primeraCiudad);
139  sendkeys("1235432134", numeroTelefono);
140  sendkeys("sdfreg@gmail.com", email);
141  click(aceptarTerminos);
142  click(botonContinuar4);
143  String NumeroCaso1 = getText(findElement(By.cssSelector(".modal-body > p")));
144  System.out.println(NumeroCaso1);
145  click(botonFinalizar);
146  close();
147 }
148 }
149
```

Figura 55. Estructura de las clases hija, parte 3

Las figuras 54 y 55 tienen la quinta y última sección del código. Se crea una función *procesoSolicitud()* que contendrá cada uno de los pasos que componen el caso de prueba para un flujo de la aplicación. Es acá donde se utilizan los métodos provenientes de la clase padre para redirigir, hacer clic, desplazarse por la pantalla, escribir, almacenar y verificar texto, y cargar documentos.

Acá se ve la importancia de la asignación de variables para los identificadores de los elementos de la página web, ya que en caso de que se quiera dar clic a cierto elemento, es mucho más sencillo utilizar el código

click(listaCobertura);

que, en caso de no haber asignado variables, utilizar el código

click(*By.cssSelector("sb-card-menu:nth-child(1) .btn"));*

Al igual que para los métodos de la clase padre, la asignación de variables es una estrategia preventiva, ya que si en alguna modificación de la aplicación, se llegara a cambiar el identificador de un elemento, solo se tendrá que realizar la modificación en la línea de código donde se asigne la variable y no en cada línea de código en donde se utilice el elemento.

De forma similar a las figuras 53, 54 y 55, se deben desarrollar las otras cuarenta y tres clases hijas que cumplan con el resto de los flujos de la aplicación.

Por último, se desarrollará una clase llamada *testrunner*, que será la que ejecute las funciones del programa como tal, tanto de las clases hijas como de la clase padre. Esta clase tiene la estructura mostrada en la figura 56:

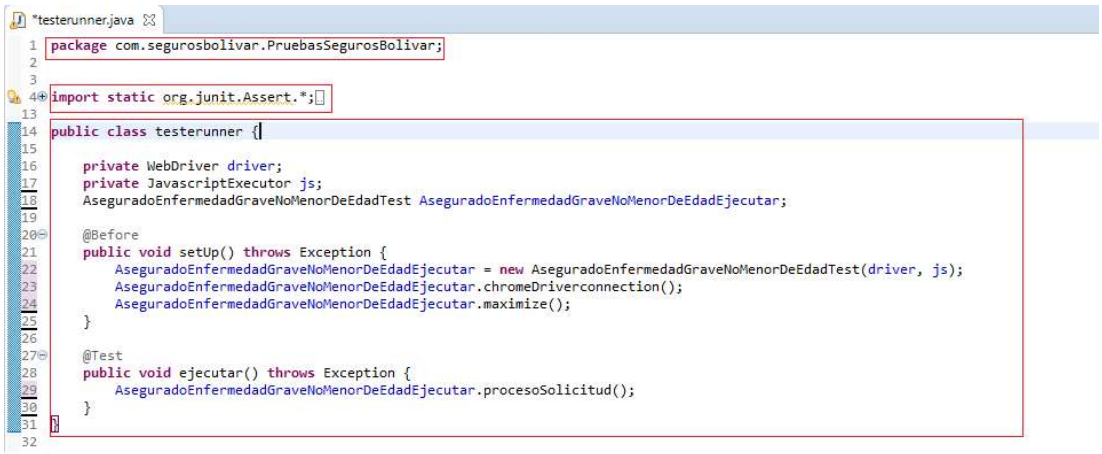


Figura 56. Estructura de la clase de ejecución.

Nuevamente, el programa de la clase se divide en secciones, tres en este caso, las dos primeras, como en los casos anteriores, se destinan a llamar al paquete en el que se encuentra la clase y a las librerías necesarias. La tercera sección, se encarga de instanciar un objeto asociado a cada una de las clases hijas (línea 18), de abrir y maximizar el navegador de Chrome para el flujo correspondiente (línea 21 a 25), y de ejecutar la función *procesoSolicitud()* que es diferente para cada uno de los cuarenta y cuatro flujos, por lo que la función debe asociarse al objeto que se instanció a cada clase.

A medida que se vayan generando las clases hijas, se deben ir agregando líneas de código a la clase *testrunner,* para que así, cada clase hija tenga asociado el proceso de

abrir y maximizar el navegador de Chrome, y el proceso de ejecución de su función *procesoSolicitud()*.

Al llegar a este punto, se da por concluido el desarrollo del proyecto.

9. RESULTADOS

A continuación, se muestran las consolas de Selenium IDE y Eclipse cuando se ha ejecutado con y sin errores la automatización de un caso de prueba:

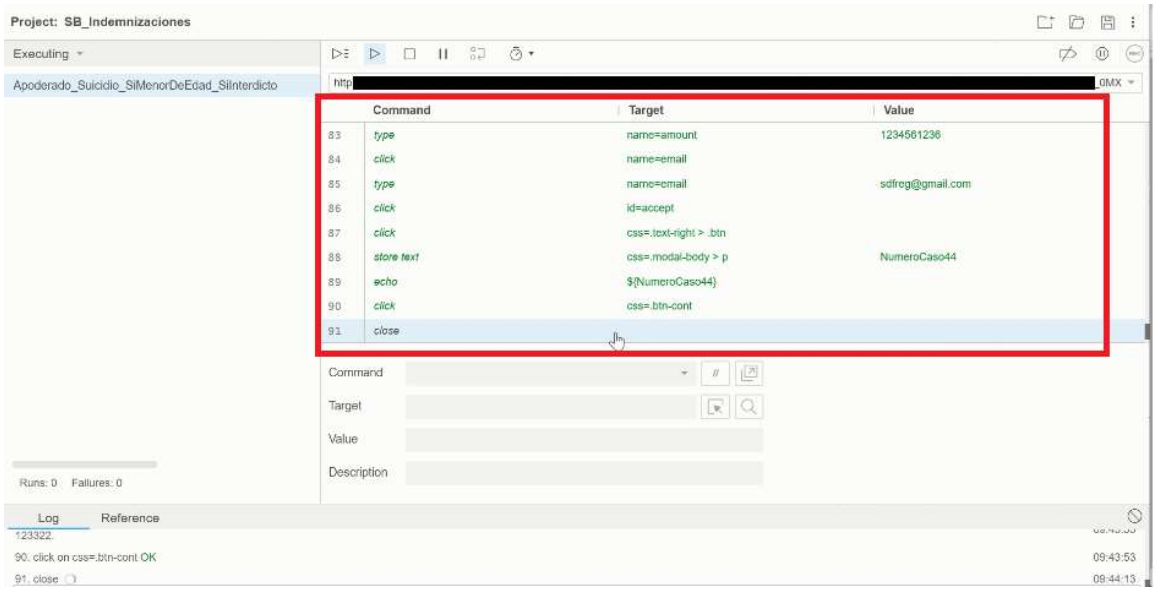


Figura 57. Resultados al ejecutar exitosamente la automatización desde Selenium IDE. Parte 1.

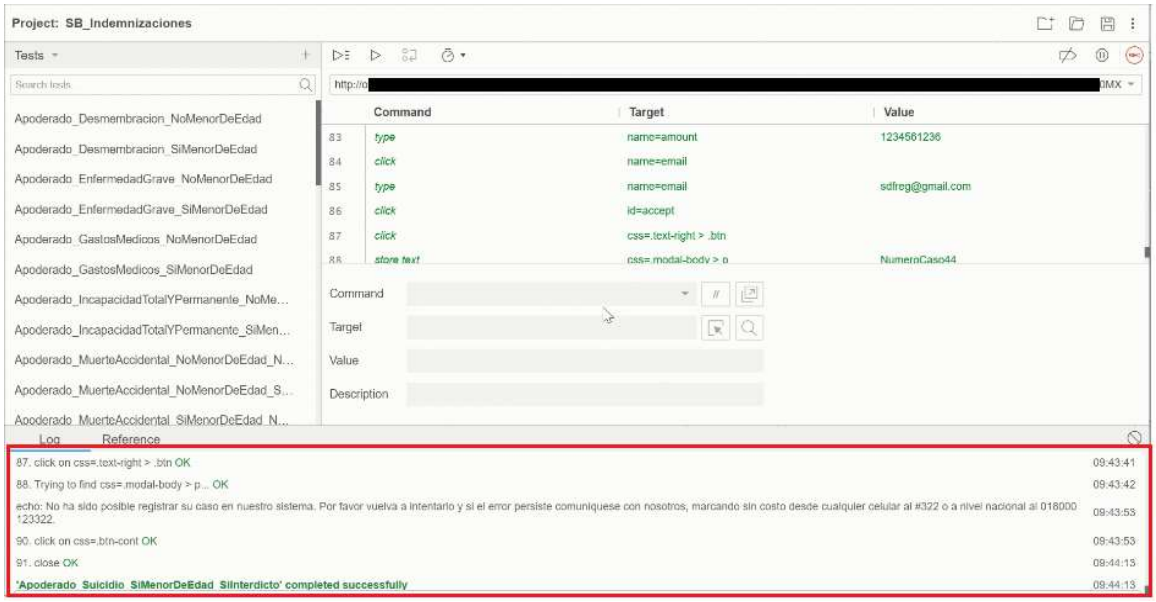


Figura 58. Resultados al ejecutar exitosamente la automatización desde Selenium IDE. Parte 2.

De las figuras mostradas anteriormente se pueden apreciar dos cosas: la primera, y como se ve en la figura 57, es que a medida que se van ejecutando los pasos de que

componen la automatización del caso de prueba, estos se van resaltando de un color verde, lo que indica que su ejecución se está llevando a cabo exitosamente. Además, y como se muestra en la figura 58, en la consola de Selenium IDE se muestra también el estado de cada uno de estos pasos que componen el caso de prueba, cuyo valor es "OK" resaltado del mismo color verde. En esta consola también se muestra resaltado en color verde un mensaje informativo que comunica que el caso de prueba (*Apoderado_Suicidio_SiMenorDeEdad_SiInterdicto* en este caso) se completó satisfactoriamente cuando todos y cada uno de los pasos que lo componen se hayan ejecutado sin problema.

Como caso contrario, a continuación, se muestra el comportamiento de Selenium IDE cuando se presentan fallas en la ejecución de los procesos de automatización:

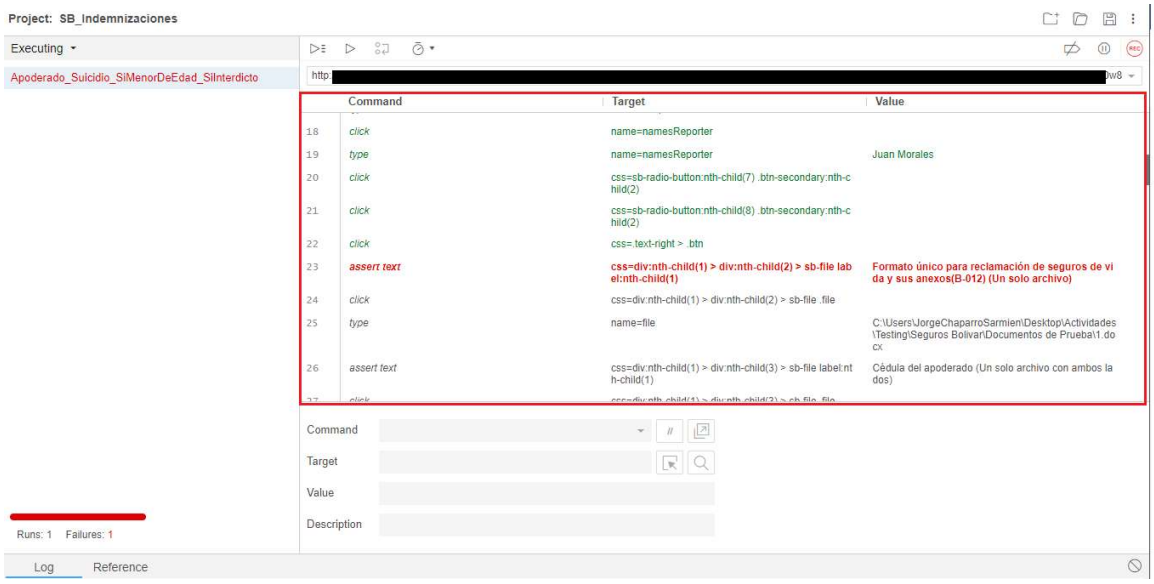


Figura 59. Resultados en Selenium IDE cuando se presentan fallas en la automatización. Parte 1.

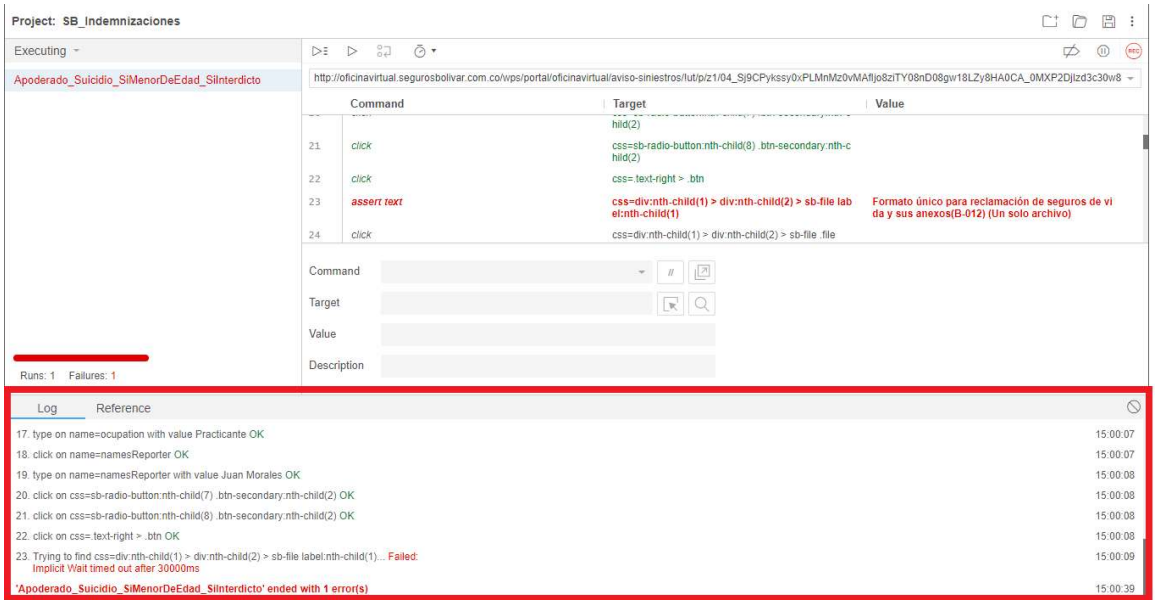


Figura 60. Resultados en Selenium IDE cuando se presentan fallas en la automatización. Parte 2.

Cuando existe algún error en el proceso de automatización, en la sección donde se encuentran los pasos del caso de prueba, se resalta el color rojo aquél en el que se haya presentado la falla (tal como se muestra en la figura 59), y, de forma detallada, se describe el error en la consola de Selenium IDE (figura 60), que para este caso, el error se generó porque no fue posible encontrar el elemento asociado al identificador en cuestión antes de que la espera implícita de la herramienta llegara a su tiempo máximo (30 segundos). Además, se informa que el caso de prueba terminó su ejecución después de que se presentara el error.

Como la inconsistencia anterior, es posible que se presenten distintos tipos de errores en la ejecución automatizada del caso de prueba. Sin importar de qué se trate, el proceso se detendrá y mostrará el error de forma detallada en la consola.

Ahora, cuando la automatización de los casos de prueba se ejecuta desde Eclipse, es posible obtener los siguientes resultados:

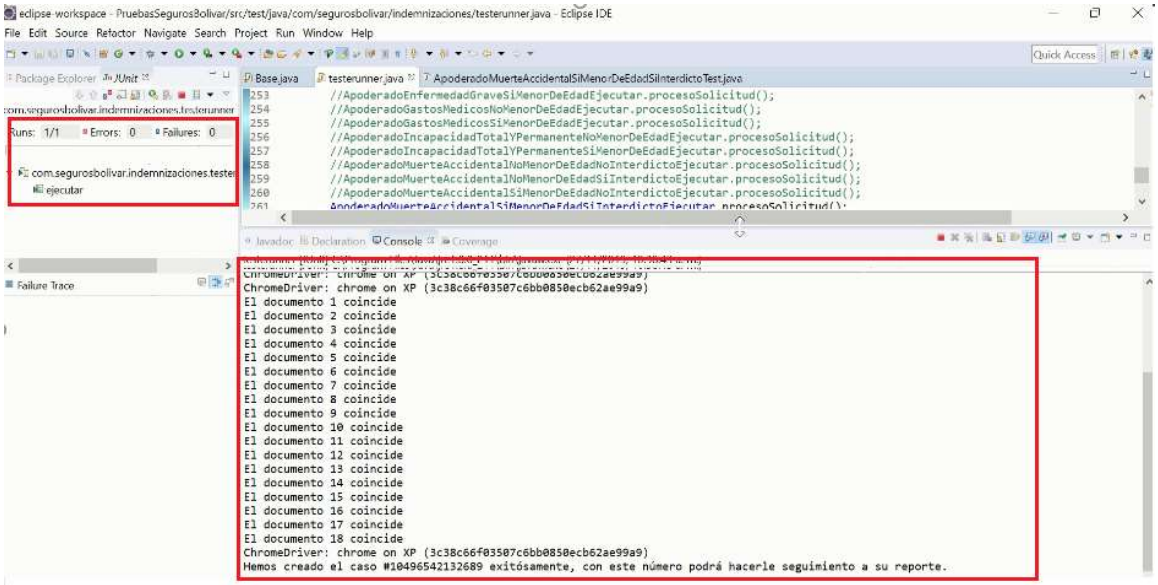


Figura 61. Resultados obtenidos en Eclipse al ejecutar exitosamente la automatización.

Como puede verse en la figura 61, el programa se ejecutó una vez (Runs 1/1) sin presentar errores ni fallas. En la consola se muestran únicamente mensajes correspondientes a la programación (informativos sobre la coincidencia entre los documentos que se solicitan y de verificación sobre el estado de la solicitud) y de la conexión creada por WebDriver; lo cual indica que la automatización se ejecutó satisfactoriamente.

Cuando existen fallas en la automatización, la consola se presentará de forma similar a como se muestra en la figura 61, solo que, ahora estará vacía o mostrará los mensajes correspondientes hasta antes de que se presente el error. Además, en la sección *Errors* o *Failures* ya no se mostrará el número 0, sino el correspondiente a las fallas que existan, que estarán explicadas de forma detallada en esta misma sección.

10. CONCLUSIONES

- En un proyecto que está continuamente en desarrollo y mejora, las pruebas de regresión son sumamente importantes, ya que estas comprueban la funcionalidad que la aplicación/servicio/página tenía en versiones anteriores, de forma que al equipo de desarrollo y a los clientes se les asegura calidad con cada actualización que se le hace al proyecto.
- El proyecto desarrollado puede ejecutarse únicamente en el navegador de Google Chrome (seleccionado, al ser el navegador utilizado en las fuentes de información que se tomaron como referencia para el desarrollo de este proyecto [15] y [22]), ya que se hace uso de la dependencia Chrome Driver. Si se quisiera ejecutar desde otro navegador, sería necesario borrar de la ruta *src/test/resources* (Figura 44) la dependencia en cuestión, e instalar el controlador correspondiente al nuevo navegador. Únicamente se debe trabajar desde un navegador a la vez, ya que el usar más de una dependencia implicará que los scripts se ejecuten sobre todos los navegadores que estén asociadas a estas, consumiendo un exceso de recursos computacionales en la automatización de testing, además de que se tendrán que agregar/modificar líneas de código a la configuración de la clase de ejecución mostrada en la figura 56 (en donde se evidencia la utilización de la dependencia ChromeDriver en la sección comprendida de la línea 21 a 24).
- Ejecutar las pruebas de QA para la aplicación “Indemnizaciones” de forma manual, inicialmente requería de aproximadamente 10 horas de trabajo. Al implementar la automatización, los recursos de tiempo que inicialmente se

invertían en esta labor se redujeron a menos de 45 minutos, además de que también se redujeron los recursos humanos (ya no se necesita de un tester para la ejecución de casos de prueba relacionados a *Indemnizaciones*) y económicos (ya no se le debe pagar a un tester para que ejecute los casos de prueba, sino solo para que documente y reporte las incidencias que surjan), por lo que el desarrollo del proyecto tuvo un gran impacto en CAI, IBM.

- Los proyectos desarrollados, tanto en Selenium IDE como en SeleniumWebDriver, solo serán funcionales inicialmente en el equipo en que fueron creados. Esto se debe a que en ellos se utilizan rutas referentes a documentos (con un nombre específico) y a archivos, que permitirán bien sea el cumplimiento de los flujos de casos de prueba (documentos que se cargarán a la página) o la correcta ejecución de los programas (librerías, dependencias y demás). En un nuevo equipo, lo más probable es que tales rutas sean diferentes o incluso los archivos sean inexistentes. Sin embargo, debería ser posible (se parte de una suposición, ya que no fue implementado) crear un directorio raíz dentro del mismo proyecto de Selenium (tanto IDE como WebDriver) que contenga los archivos en cuestión, para que puedan ser llamados desde los scripts (entonces, desde una ubicación fija).
- En el proyecto desarrollado en Eclipse, la implementación de la clase “Base” es sumamente importante, ya que además de simplificar la programación que se aplicará a las clases hijas, asegura modificaciones mínimas en caso de que los comandos de Selenium cambien en las futuras actualizaciones de su software.

- Almacenar los identificadores de los elementos de la página web en variables puede parecer innecesario, pero debido a su compleja estructura, es recomendable hacerlo, con el fin de que el código sea más entendible a por simple inspección para los programadores.
- Entre las herramientas que se consideraron para generar la automatización de casos de prueba, Selenium fue la que más beneficios aportaba, ya que además de ser open source, también podía ser utilizada por personas con pocos conocimientos en el área de la programación (lo que la hacía superior a Cucumber); esto gracias a su herramienta Selenium IDE (la cual también simplificaba el trabajo al momento de generar la automatización en forma de código de programación). Esto puede evidenciarse gracias a la Tabla 1.
- Aunque la automatización se haya implementado correctamente, existe la posibilidad de que ésta falle si la respuesta ante el llamado de los servicios externos de la aplicación (los que pertenecen a Seguros Bolívar, como lo son las listas desplegables de tipo de documento y ciudad de ocurrencia) sea mayor a 30 segundos (tiempos implícitos definidos por Selenium IDE y Selenium WebDriver). Esto podría solucionarse eliminando los tiempos de espera implícitos, sin embargo, surgiría un nuevo error, y es que, si los servicios externos realmente están fallando y los elementos provenientes de tales servicios nunca aparecen en la aplicación web, el caso de prueba jamás terminaría su ejecución, ya que estaría en espera constante por los elementos en cuestión, y un tester

tendría que finalizar la labor de forma manual (justo lo que no debería hacerse si el proyecto está implementado).

11. RECOMENDACIONES

Del presente documento, existen dos tipos de recomendaciones: las enfocadas a la página web (sobre cómo se ejecutan los flujos de la aplicación) y las enfocadas a la automatización de los casos de prueba (cómo es posible mejorar el rendimiento de los proyectos de Selenium IDE y de Eclipse).

11.1 RECOMENDACIONES PARA LA PÁGINA WEB

- En el formulario “Información del asegurado” mostrado en la figura 11, no se están realizando validaciones ante la mayoría de edad, es decir, en caso de que como tipo de documento se seleccione “cédula de ciudadanía”, en el campo de texto “edad” se debería permitir el ingreso únicamente de números mayores o iguales a 18.
- Siempre que se da clic en un botón “atrás” para retornar a algún formulario, toda la información que ya se había llenado se borra, por lo que deben proporcionarse nuevamente todos los datos. Los formularios deberían conservarse tal y como el usuario los dejó.
- En la pantalla para cargar los documentos, no existen restricciones de extensiones para estos, es decir, que todo archivo, sin importar su formato, puede ser cargado a la aplicación y enviado a base de datos. Debería limitarse a archivos .jpg, .jpeg, .png, .pdf, .doc y .docx.

11.2 RECOMENDACIONES PARA LA AUTOMATIZACIÓN DE CASOS DE PRUEBA DESARROLLADA

- Actualmente, en la pantalla donde se muestra el listado de documentos que se solicitarán posteriormente (figura 10), la automatización solamente busca al botón “continuar” para darle clic y continuar con el flujo. Sería bueno utilizar los comandos “Assert” de Selenium para verificar que efectivamente se están mostrando los documentos asociados a cada solicitud.
- La clase “testrunner” debería simplificarse, buscando una forma de que el navegador se abra y maximice para todas las clases hijas de forma general. Actualmente, este procedimiento se tiene que realizar para cada una de ellas.
- Para que el proyecto de Eclipse pueda entenderse incluso por sujetos que no cuenten con conocimiento técnico sobre programación, Cucumber es una buena herramienta, ya que será la interfaz que muestre en lenguaje natural los comandos que se están ejecutando en tiempo real.
- El proyecto de Eclipse puede exportarse como un archivo .jar. Este archivo, mediante un único clic ejecutará la automatización de los cuarenta y cuatro casos de prueba desarrollados, lo cual permitirá que personas que desconozcan el entorno de trabajo de Eclipse realicen el testing de la aplicación (al menos de los casos de prueba que fueron automatizados).
- Agregar los flujos erróneos a la aplicación. Es decir, que de forma automática se llenen incorrecta o incompletamente los formularios, con el fin de verificar el comportamiento de la aplicación bajo estas condiciones.

- Existen fases del proceso de solicitud que son comunes para todos los flujos (formulario de datos de contacto), por lo que la automatización de estas fases comunes podría implementarse una vez de forma general, y no una vez para cada una de las clases hijas, como se hace actualmente.
- Para que el proyecto se pueda ejecutar desde cualquier equipo sin tener que hacerle modificaciones, podría crearse una carpeta raíz (con una ubicación constante) que contenga los archivos necesarios (documentos que se cargarán y dependencias) para su correcto funcionamiento.

12. BIBLIOGRAFÍA

[1] Nectec (2019). *¿Qué es IBM en informática?* [Página Web]. URL: https://www.netec.com/que-es-ibm-en-informatica

[2] IBM (2016). *Conozca IBM, Productos y Soluciones.* [Página Web]. URL: https://www.ibm.com/expressadvantage/mx/pdf/Folleto_Conozca_IBM.pdf

[3] Das, D. Roy, P. Shubhashish, D. (2017). *What are GTS and GBS in IBM? Which is a better grade?* [Página Web]. URL: https://www.quora.com/What-are-GTS-and-GBS-in-IBM-Which-is-a-better-grade

[4] Verisign. *What is a URL?* [Página Web]. URL: https://www.verisign.com/en_US/website-presence/online/what-is-a-url/index.xhtml

[5] OBS Business School. *¿Qué es un Project Manager y cuáles son sus funciones?* [Página Web]. URL: https://obsbusiness.school/int/blog-project-management/salidas-profesionales-pm/que-es-un-project-manager-y-cuales-son-sus-tareas

[6] Linuxito. (2013). *El modelo de desarrollo, testing y producción.* [Página Web]. URL: https://www.linuxito.com/programacion/237-el-modelo-de-desarrollo-testing-y-produccion

[7] Código Facilito. (2015). *Commits. Administrar tu repositorio.* [Página Web]. URL: https://codigofacilito.com/articulos/commits-administrar-tu-repositorio

[8] Choucair. (2018). *¿Sabes qué son los ambientes de prueba de desarrollo de software?* [Página Web]. URL: https://www.choucairtesting.com/a-day-at-the-office/

[9] Utting, M. Legeard, B. (2006). *Practical Model-Based Testing: A Tools Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[10] Toledo, F. Gurcio, A. Scuoteguazza, G. (2014). *Introducción a las pruebas de sistemas de información.* Abstracta, Montevideo, Uruguay.

[11] Cristiá, M. (2015). *Introducción al Testing de Software.* Facultad de Ciencias Exactas, Ingeniería y Agrimensura. Universidad Nacional del Rosario. Argentina.

[12] Ghezzi, C. Jazayeri, M. Mandrioli, D. (1991). *Fundamentals of software engineering.* Prentice Hall, Upper Saddle River, New Jersey, USA.

[13] Myers, G. (2004). *The Art of Software Testing.* Segunda Edición. John Wiley & Sons.

[14] Sánchez, J. (2015). *Pruebas de Software, Fundamentos y Técnicas.* Universidad Politécnica de Madrid. España.

[15] Selenium. [Página Web]. URL: http://www.seleniumhq.org/

[16] HP. (2010). *HP QuickTest Professional para Business Process Testing.* [Página Web].

URL:https://softwaresupport.softwaregrp.com/doc/KM1201023?fileName=hp_man_QTP11.00_BPT_User_sp_pdf.pdf

[17] SoapUI. [Página Web]. URL: http://www.soapui.org/about-soapui/what-is-soapui-.html

[18] Cucumber. [Página Web]. URL: https://cucumber.io/

[19] Dependencia de Selenium WebDriver. [Página Web]. URL: https://www.seleniumhq.org/download/maven.jsp

[20] Dependencia de JUnit. [Página Web]. URL: https://junit.org/junit4/dependency-info

[21] Chrome Driver. [Página Web]. URL: https://chromedriver.chromium.org/downloads

[22] Chaparro, J. Riascos, H. (2019). *Manual de Selenium.* IBM, Colombia.

[23] Martínez, J. *Fundamentos de programación en Java.* Universidad Complutense de Madrid. Facultad de Informática. Editorial EME.

[24] García, J. Rodríguez, J. Mingo, I. Imaz, A. Brazález, A. Larzabal, A. Calleja, Jesús. García, J. (2000). *Aprenda Java como si estuviera en primero.* Universidad de Navarra. Escuela Superior de Ingenieros Industriales.

[25] García, H. *El lenguaje de programación Java.*

[26] Rouse, M. (2018). *Integrated development environment.* [Página Web]. URL: https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

[27] Castillero, O. (2017). *Los 15 tipos de investigación (y características).* [Página Web]. URL: https://psicologiaymente.com/miscelanea/tipos-de-investigacion

13. ANEXOS

Jorge Leonardo Chaparro Sarmiento, nacido el 26 de octubre de 1998 en la ciudad de Tunja. Es egresado con profundización en música del Colegio de Boyacá. Estudiante de decimo semestre de ingeniería electrónica en la Universidad Santo Tomás, Seccional Tunja, Colombia. Sus intereses académicos y laborales se orientan hacia la automatización, instrumentación y control digital, con el fin de optimizar procesos que se realizan en el hoy, o incluso, innovar en estos ámbitos. Ha participado en distintas actividades académicas a nivel nacional e internacional, dentro de las cuales se destacan el Curso de Robótica en Ambientes de Comunicación No Verbal, llevado a cabo en diciembre de 2018 en la Universidad de Saitama, Japón; y el intercambio académico en la Universidad Autónoma de Baja California, en la ciudad de Ensenada, México, durante el primer semestre de 2017. Durante el segundo semestre de 2019, realizó prácticas profesionales en la empresa IBM, en la ciudad de Bogotá, Colombia; sus labores se orientaban a los procesos de control de calidad de desarrollo de software (ejecución de pruebas de calidad de forma manual y automatización de las mismas).