Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería de Software

Implementación de un framework de automatización de pruebas en un marco de trabajo ágil para mejorar el proceso de calidad de software en una compañía de seguros

TRABAJO DE SUFICIENCIA PROFESIONAL

Para optar el Título Profesional de Ingeniero de Software

AUTOR

Jesús Manuel LUCERO AVILA

ASESOR

Norberto Antonio OSORIO BELTRÁN

Lima, Perú

2022

Referencia bibliográfica

Lucero, J. (2022). *Implementación de un framework de automatización de pruebas en un marco de trabajo ágil para mejorar el proceso de calidad de software en una compañía de seguros*. [Trabajo de suficiencia profesional de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ingeniería de Sistemas e Informática, Escuela Profesional de Ingeniería de Software]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Jesús Manuel Lucero Avila
Tipo de documento de identidad	DNI
Número de documento de identidad	45479561
URL de ORCID	https://orcid.org/0000-0001-6180-1379
Datos de asesor	
Nombres y apellidos	Norberto Antonio Osorio Beltrán
Tipo de documento de identidad	DNI
Número de documento de identidad	08799230
URL de ORCID	https://orcid.org/0000-0001-5921-1118
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	César Augusto Alcántara Loayza
Tipo de documento	DNI
Número de documento de identidad	09132297
Miembro del jurado 1	
Nombres y apellidos	Augusto Parcemon Cortez Vásquez
Tipo de documento	DNI
Número de documento de identidad	08634618
Miembro del jurado 2 (Asesor)	
Nombres y apellidos	Norberto Antonio Osorio Beltrán
Tipo de documento	DNI
Número de documento de identidad	08799230
Datos de investigación	
Línea de investigación	No aplica

Grupo de investigación	No aplica
Agencia de financiamiento	Sin financiamiento
Ubicación geográfica de la investigación	País: Perú Departamento: Lima Provincia: Lima Distrito: Cercado de Lima Jr. Carlos Amezaga No. 375 Universidad Nacional Mayor de San Marcos Latitud: -12.0564232 Longitud: -77.0843327
Año o rango de años en que se realizó la investigación	2022
URL de disciplinas OCDE	2.02.04 -- Ingeniería de sistemas y comunicaciones https://purl.org/pe-repo/ocde/ford#2.02.04

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
Escuela Profesional de Ingeniería de Software

Acta Virtual de Sustentación
del Trabajo de Suficiencia Profesional

Siendo las 19:50 horas del día 19 de agosto del año 2022, se reunieron virtualmente los docentes designados como Miembros de Jurado del Trabajo de Suficiencia Profesional, presidido por el Mg. Alcántara Loayza César Augusto (Presidente), Mg. Cortez Vásquez Augusto Parcemon (Miembro) y el Ing. Osorio Beltrán Norberto Antonio (Miembro Asesor), usando la plataforma Meet (https://meet.google.com/ikv-yywj-rqb), para la sustentación virtual del Trabajo de Suficiencia Profesional intitulado: ***"IMPLEMENTACIÓN DE UN FRAMEWORK DE AUTOMATIZACIÓN DE PRUEBAS EN UN MARCO DE TRABAJO ÁGIL PARA MEJORAR EL PROCESO DE CALIDAD DE SOFTWARE EN UNA COMPAÑIA DE SEGUROS",*** *por el Bachiller* ***Lucero Avila Jesús Manuel;*** *para obtener el Título Profesional de Ingeniero de Software.*

Acto seguido de la exposición del Trabajo de Suficiencia Profesional, el Presidente invitó al Bachiller a dar las respuestas a las preguntas establecidas por los miembros del Jurado.

El Bachiller en el curso de sus intervenciones demostró pleno dominio del tema, al responder con acierto y fluidez a las observaciones y preguntas formuladas por los señores miembros del Jurado.

Finalmente habiéndose efectuado la calificación correspondiente por los miembros del Jurado, el Bachiller obtuvo la nota de ***17 (DIECISIETE).***

A continuación el Presidente de Jurados el Mg. Alcántara Loayza César Augusto, declara al Bachiller ***Ingeniero de Software****.*

Siendo las 20:58 horas, se levantó la sesión.

Presidente
Mg. Alcántara Loayza César Augusto

Miembro
Mg. Cortez Vásquez Augusto Parcemon

Miembro Asesor
Ing. Osorio Beltrán Norberto Antonio

Universidad Nacional Mayor de San Marcos

Universidad del Perú. Decana de América

Facultad de Ingeniería de Sistemas e Informática

Escuela Profesional de Ingeniería Software

INFORME DE EVALUACIÓN DE ORIGINALIDAD
Nº 021-EPISW-FISI-2022

1. *Autoridad Académica que emite el **I**nforme de Originalidad:*	*Directora de la Escuela Profesional de Ingeniería de Software*
2. *Apellidos y Nombres de la autoridad académica:*	*Dra. Nora Bertha La Serna Palomino*
3. *Operador del programa informático de similitudes:*	*Dra. Nora Bertha La Serna Palomino*
3. *Documento evaluado:*	*Tesis para Pregrado* *Título: "**Implementación de un framework de automatización de pruebas en un marco de trabajo ágil para mejorar el proceso de calidad de software en una compañía de seguros**"*
5. *Autores del documento:*	*Lucero Avila, Jesús Manuel*
6. Fecha de recepción de documento	*Recepción: 10/09/2022*
7. *Fecha de aplicación del programa detector de similitudes:*	*Revisión: 10/09/2022*
8. Software utilizado:	*Turnitin*
9. Configuración del programa detector de similitudes:	*Excluye textos entrecomillados: Sí* *Excluye biografías: Sí* *Excluye cadenas menores a 40 palabras: Sí* *Otro criterio (especificar): No*
10. Porcentaje de similitudes según programa detector de similitudes	*Ocho por ciento (8%)*
11. Fuentes originales de las similitudes encontradas	*Se adjuntan en 01 (una) foja al presente informe*
12. Observaciones:	*Ninguna*
13. Calificación de originalidad *i. Documento cumple criterios de originalidad, sin observaciones* *ii. Documento cumple criterios de originalidad, con observaciones* *iii. Documento no cumple criterios de originalidad.*	*Documento cumple criterio de originalidad, sin observación*
14. *Fecha del Informe:*	*14/11/2022*

Dra. Nora Bertha La Serna Palomino

Directora (e) de la EPISW

DEDICATORIA

A mi familia quienes siempre me brindaron todo su amor y apoyo incondicional para hacer de mí una mejor persona y lograr todas mis metas.

AGRADECIMIENTO

Al Final de este trabajo me gustaría agradecer a las siguientes personas que han apoyado que este trabajo llegue a su fin.

- A mis padres que, con esfuerzo, dedicación, perseverancia lograron formarme en cada etapa de mi vida.
- Al profesor asesor Norberto Antonio Osorio Beltrán, quien me oriento con profesionalismo con sus consejos y recomendaciones a lo largo de la elaboración de este informe.
- A mis compañeros y amigos quieres me brindaron su apoyo en todo momento.

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE

Implementación de un framework de automatización de pruebas en un marco de trabajo ágil para mejorar el proceso de calidad de software en una compañía de seguros

Autor: **Lucero Avila, Jesús Manuel**

Asesor: **Osorio Beltrán, Norberto Antonio**

Título: **Trabajo de Suficiencia Profesional para optar el Título Profesional de Ingeniero de Software**

Fecha: **Junio 2022**

RESUMEN

El presente trabajo de suficiencia profesional describe la definición de los lineamientos y técnicas para llevar a cabo la automatización de las pruebas de regresión de aplicaciones Web y APIs en la compañía de seguros. El objetivo principal es mejorar y optimizar el proceso de calidad de software, y para ello se implementó un framework para la construcción de los scripts. El proyecto se realizó bajo el marco de trabajo ágil Scrum, que permitió dividir el desarrollo del framework en tareas y sprints. En el plano técnico o tecnológico se usó el enfoque de Desarrollo Dirigido por Comportamiento (BDD de las siglas en inglés), también se utilizó los patrones de diseño Modelo de Objeto de Página (POM de las siglas en ingles), Singleton y las herramientas Selenium, Rest Assured, Java, Cucumber, Serenity, Jenkins y aplicando dos de las buenas prácticas de los principios SOLID (S.O.L.I.D de las siglas en ingles). Al implementar el framework de automatización de pruebas y tenerla integrada con Jenkins, permitió que cada vez que se desplieguen historias de usuarios en el ambiente de pruebas se ejecuten las pruebas manuales propias de los cambios realizados y a su vez se ejecute el Job de Jenkins para las pruebas de regresión dando como resultado una reducción en el tiempo de ejecución, lograr una mayor cobertura de pruebas y así garantizar la calidad necesaria y aumentar la productividad para mayor beneficio de la compañía.

Palabras Clave: Automatización de pruebas, Desarrollo dirigido por el comportamiento, Selenium, Serenity, Modelo de Objeto de Página, Jenkins.

MAJOR NATIONAL UNIVERSITY OF SAN MARCOS

FACULTY OF SYSTEMS AND INFORMATICS ENGINEERING

PROFESSIONAL SCHOOL OF SOFTWARE ENGINEERING

Implementation of a test automation framework in an agile framework to improve the software quality process in an insurance company

Author: **Lucero Avila, Jesús Manuel**

Advisor: **Osorio Beltrán, Norberto Antonio**

Title: **Professional Sufficiency Work to opt for the Professional Title of Software Engineer**

Date: **June 2022**

ABSTRACT

This work of professional sufficiency describes the definition of the guidelines and techniques to carry out the automation of regression testing of Web applications and APIs in the insurance company. The main objective is to improve and optimize the software quality process, and for this purpose a framework for the construction of the scripts was implemented. The project was carried out under the agile framework Scrum, which allowed to divide the framework development in tasks and sprints. At the technical or technological level, the Behavior Driven Development (BDD) approach was used, as well as the design patterns Page Object Model (POM), Singleton and the tools Selenium, Rest Assured, Java, Cucumber, Serenity, Jenkins and applying two of the best practices of the SOLID principles (S.O.L.I.D.). By implementing the test automation framework and having it integrated with Jenkins, it allowed that every time user stories are deployed in the test environment, the manual tests of the changes made are executed and at the same time the Jenkins Job is executed for regression testing, resulting in a reduction in execution time, achieving greater test coverage and thus ensuring the necessary quality and increasing productivity for the greater benefit of the company.

Keywords: Test Automation, Behavior Driven Development, Selenium, Serenity, Page Object Model, Jenkins.

Tabla de contenido

INDICE DE TABLAS

INDICE DE FIGURAS

INTRODUCCIÓN

En el presente informe se detalla el proyecto de la implementación de un framework de automatización de pruebas para aplicaciones Web y APIs en una compañía de seguros con el objetivo de mejorar y optimizar el proceso de calidad de software al momento de realizar las pruebas de regresión.

Esta iniciativa surgió debido a un problema importante que tienen las pruebas de regresión para el equipo de analistas de pruebas funcionales, que es el consumo de tiempo en la ejecución ya que son realizadas de forma manual y esto lleva a una tardía identificación de errores generando un costo para la compañía ya que este esfuerzo crece a medida que se incorporan nuevas funcionalidades en las aplicaciones.

El presente trabajo detalla y describe los siguientes capítulos:

- En el capítulo I, se especifica el desarrollo de la trayectoria profesional del autor, los roles, las funciones y herramientas tecnológicas que se usaron en los proyectos que participe en orden cronológico la que evidencia la experiencia laboral obtenida.
- En el capítulo II, se describe la información de la compañía de seguros, tales como su organigrama, visión y misión. Además, se describe en detalle la experiencia laboral del autor trabajando en dicha organización.
- En el capítulo III, detalla la situación problemática, la solución, metodología, implementación, los enfoques y patrones empleados, así como el fundamento y consideraciones importantes para el proceso de la implementación del framework de automatización de pruebas para las aplicaciones Web y APIs, así como la evaluación económica y los beneficios obtenidos de la solución.
- En el capítulo IV, se mencionan los aportes y conocimientos empleados del autor, las necesidades que se atendieron y la experiencia adquirida dentro del proyecto.
- En el capítulo V, se presenta las conclusiones y recomendaciones finales del informe.

CAPÍTULO I - TRAYECTORIA PROFESIONAL

PRESENTACIÓN PROFESIONAL

El autor del presente trabajo es bachiller en Ingeniería de Software de la Universidad Nacional Mayor de San Marcos, con más de 5 años de experiencia laboral en calidad y automatización de pruebas de software para diferentes proyectos en los sectores de banca, telecomunicaciones y seguros bajo el marco de trabajo ágil (Scrum). El autor a lo largo de su experiencia se ha desempeñado en los perfiles de Analista de pruebas, QA Automatizador e Ingeniero de automatización de pruebas.

Además, el autor cuenta con conocimientos en pruebas de caja blanca, caja negra y con sólida experiencia en técnicas de automatización de pruebas Web, APIs, Móvil, Escritorio, Rendimientos e Integración continua.

Adicionalmente, el autor cuenta con habilidades analíticas, de resolución de problemas, liderazgo, proactivo, siempre orientado a resultados y con mucha disposición para el trabajo en equipo.

Tabla 1: Experiencia profesional

Compañía de Seguros	**Febrero 2021 - Actualidad**
Cargo	Ingeniero de Automatización de pruebas
Funciones	- Definir los lineamientos de gobierno de automatización web, APIs, móvil, rendimiento. - Definir el framework de automatización de pruebas web, APIs, móvil, rendimiento. - Definir y configurar los proyectos automatizados en Jenkins. - Realizar pilotos de herramientas de pruebas que ayuden en el proceso de pruebas. - Encargado de dar seguimiento y asesoría a los automatizadores de los equipos donde realizan automatizaciones web y APIs. - Automatizar aplicaciones web, APIs, móviles.
Globant	**Diciembre 2020 – Enero 2021**
Cargo	QA Automatizador
Funciones	- Definir las estrategias de pruebas. - Realizar el análisis y diseño de casos de pruebas web, APIs. - Usar la herramienta Jira para el registro y seguimiento de historias de usuario, tareas y defectos. - Realizar ejecuciones de escenarios y casos de pruebas funcionales manuales. - Verificar los resultados y evaluación de los criterios de aceptación.

	- Realizar consultas en Base de Datos Oracle. - Realizar informes sobre el proceso de pruebas y avances. - Realizar pruebas de regresión. - Evaluar y seleccionar escenarios para las automatizaciones. - Generar reportes de las ejecuciones de los escenarios automatizados. - Usar la metodología de trabajo Scrum.
Choucair Testing	**Marzo 2020 – Diciembre 2020**
Cargo	QA Automatizador
Funciones	- Análisis y diseño de casos de prueba web / APIs, móviles (Android, IOS). - Usar la herramienta Trello para el registro y seguimiento de tareas, defectos. - Realizar ejecuciones de escenarios y casos de pruebas funcionales manuales. - Realizar ejecuciones de casos de pruebas de APIs con las herramientas Postman, ReadyAPI. - Verificar los resultados y evaluación de los criterios de aceptación. - Realizar informes sobre el proceso de pruebas y avances. - Evaluar y seleccionar escenarios de pruebas para realizar las automatizaciones. - Realizar el diseño de escenarios en lenguaje Gherkin. - Crear y configurar los proyectos en Java. - Implementación de escenarios con Selenium, Cucumber, SerenityBDD – Web. - Implementación de escenarios con Cucumber, SerenityBDD, Rest Assured, SerenityRest, Postman, ReadyAPI - APIs. - Verificar los resultados obtenidos y evaluar los criterios de aceptación. - Generar reportes de las ejecuciones de los escenarios automatizados. - Usar la herramienta de versionamiento GitLab. - Configurar y ejecutar los escenarios automatizados en la herramienta Jenkins. - Metodología de trabajo Scrum. - Realizar consultas en Dynamobd.
Choucair Testing	**Julio 2019 – Febrero 2020**
Cargo	QA Automatizador
Funciones	- Selección de casos de prueba para automatizar. - Diseño de escenarios en lenguaje Gherkin. - Creación y configuración de proyecto en Java. - Implementación de escenarios con Selenium, Cucumber, SerenityBDD.

	- Verificar los resultados obtenidos y evaluar los criterios de aceptación. - Generar reportes de las ejecuciones de los escenarios automatizados. - Usar la herramienta de versionamiento Gitlab. - Usar la herramienta Trello para registro y seguimiento de tareas. - Configuración y ejecución de escenarios automatizados en Jenkins. - Metodología de trabajo Scrum.
Choucair Testing	**Enero 2019 – Junio 2019**
Cargo	Analista de Pruebas
Funciones	- Análisis del negocio. - Elaboración del plan y estrategia de pruebas. - Análisis, diseño de casos basado en riesgos y estimación. - Realizar la ejecución de escenarios y casos de pruebas. - Verificar los resultados obtenidos y realizar la evaluación de los criterios de aceptación. - Realizar informes sobre el proceso de pruebas y avances. - Usar la herramienta RTC / RQM (Registro de casos de prueba, registro de bugs). - Validación de información y consultas - PL/SQL.
Choucair Testing	**Agosto 2018 – Diciembre 2018**
Cargo	Analista de Pruebas Móviles
Funciones	- Análisis del negocio. - Elaboración del plan y la estrategia de pruebas. - Análisis, diseño de casos basado en riesgos y estimación. - Realizar la ejecución de escenarios y casos de pruebas. - Verificar los resultados obtenidos y realizar la evaluación de los criterios de aceptación. - Realizar informes sobre el proceso de pruebas y avances. - Metodología Agile - Scrum. - Usar la herramienta Jira (Registro de casos de prueba, registro de bugs, Kanban). - Desarrollo de pruebas funcionales y no funcionales. - Mobile Automation Testing - Appium – Java. - REST API Testing – Postman. - Usar la herramienta BitBucked. - Validación de información y consultas - SQL Server.
Choucair Testing	**Mayo 2018 – Julio 2018**
Cargo	Analista de Pruebas Colombia
Funciones	- Análisis de negocio y requisitos funcionales. - Selección de las condiciones de las pruebas.

	- Realizar la ejecución de los escenarios y casos de pruebas definidos en el plan de pruebas. - Realizar la verificación de los resultados obtenidos y realizar la evaluación de criterios de aceptación. - Mantener registros de pruebas, gestión de defectos e informes de estado. - Elaboración de informes sobre el proceso de pruebas y sobre la aplicación probada. - Desarrollo de pruebas funcionales. - Usar la herramienta de colaboración Jira, para la creación de escenarios y casos de pruebas, gestión de defectos e informes de estado. - Pruebas de servicios REST y SOAP (Herramienta SOAPUI). - Validación de información - SQL Server.
Equifax	**Junio 2016 – Abril 2018**
Cargo	Analista de Calidad
Funciones	- Análisis de negocio y requisitos funcionales. - Diseño, creación y ejecución de los escenarios y casos de pruebas definidos en el plan de pruebas. - Verificación de los resultados obtenidos y realizar la evaluación de criterios de aceptación. - Mantener registros de pruebas, gestión de defectos e informes de estado. - Interactuar con los desarrolladores, gestores de proyectos y analistas de negocios. - Elaboración de informes sobre el proceso de pruebas y sobre la aplicación probada. - Analizar los resultados de las pruebas y proponer oportunidades de mejora. - Metodología Agile (Scrum). - Desarrollo de pruebas funcionales y automatizadas. - Usar la herramienta de colaboración JIRA. - Usar las herramientas de automatización: Junit, TestNG, Selenium, JMeter, Jenkins, Sonar, Go Server, Docker.
Quipucamayoc UNMSM	**Octubre 2013 – Octubre 2015**
Cargo	Analista Programador
Funciones	- Análisis, Diseño, Implementación de nuevas funcionalidades en el módulo Recursos Humanos usando las herramientas HTML5, JavaScript, JQuery, Boostrap, Oracle, myBatys, Spring, Git. - Análisis, Diseño e Implementación de nuevas funcionalidades, elaboración de reportes y mantenimiento en el módulo de planificación usando las herramientas JSF, Primefaces, XHTML, JS, Oracle, Ireport, SVN.

Fuente: Elaboración propia

Tabla 2: Formación académica profesional

Formación	Institución	Periodo
Bachiller en Ingeniería de Software	Universidad Nacional Mayor de san Marcos - Facultad de Ingeniería de Sistemas e Informática – Escuela Académico Profesional de Ingeniería de Software	2009 - 2016

Fuente: Elaboración propia

Tabla 3: Idiomas

Idioma	Institución	Periodo
Inglés – Nivel Básico	Universidad Nacional Mayor de San Marcos	2009 -2011

Fuente: Elaboración propia

Tabla 4: Cursos

Curso	Institución	Periodo
Automatización de Pruebas funcionales de Software con Selenium y Java en Aplicaciones Web	JB ENTERPRISE GROUP	2016
Seguridad de la Información	ISEC	2017
Curso Metodologías Agiles	SCRUMstudy	2017
Fundamentos de JavaScript	Udemy	2018
Pruebas de APIs - Postman	PLATZI	2019
Integración Continua, proceso de Implantación, Herramientas	JB ENTERPRISE GROUP	2019
AiU Certified Tester in Artificial Intelligence (CTAI)	Choucair Testing	2019

Fuente: Elaboración propia

Tabla 5: Certificaciones

Curso	Periodo
ISTQB® Certified Tester, Foundation Level	2017
ISTQB® Certified Tester – Foundation Level Extensión, Agile Tester	2018
Scrum Fundamentals Certified	2018
DevOps Master	2018
DevOps Essentials Professional	2019

Fuente: Elaboración propia

Tabla 6: Otros conocimientos

Conocimiento	Herramientas
Lenguaje de programación	Java, JavaScript, Python, Groovy
Base de datos	Oracle, MySQL, SQL Server, DynamoDB
Pruebas Unitarias	Junit, Mockito, TestNG
Pruebas Automatizadas	Selenium, Cucumber, Serenity, Cypress, Appium
Pruebas de Performance	SoapUI, JMeter, ReadyAPI
Versionamiento	SVN, Git
Repositorios en la nube	GitHub, GitLab
Integración continua	Jenkins, Teamcity
Issue trackers	Jira, Trello
Metodologías	Scrum

Fuente: Elaboración propia

CAPÍTULO II - CONTEXTO EN EL QUE SE DESARROLLÓ LA EXPERIENCIA

2.1. Empresa - Actividad que realiza

La experiencia profesional adquirida se ha desarrollado en una importante entidad aseguradora, es una organización líder y con amplia trayectoria en el mercado peruano que opera principalmente en el sector seguros, se enfocan en brindar una cartera amplia en todos sus servicios y productos que permiten a las personas naturales y clientes corporativos gestionar eficazmente sus riegos.

Su promesa refleja su deseo de ayudar a los clientes a proteger lo que más valoran garantizando que logren sus objetivos, ya que les permite realizar sus actividades con confianza y tiene presencia en prácticamente todo el país ya que tiene un equipo de ventas más amplio a nivel nacional, contando con cientos de canales de venta incluidos por comercios, bancos, corredores, etc.

2.2. Visión

Ser una compañía responsable con la sociedad, centrada en los clientes y liderando la transformación digital del sector. Para cumplir con la visión, debemos estar al nivel de las grandes aseguradoras de referencia internacional, desarrollando proyectos innovadores y consolidando nuestro liderazgo en ventas, rentabilidad, eficiencia, satisfacción al cliente y gestión del capital humano.

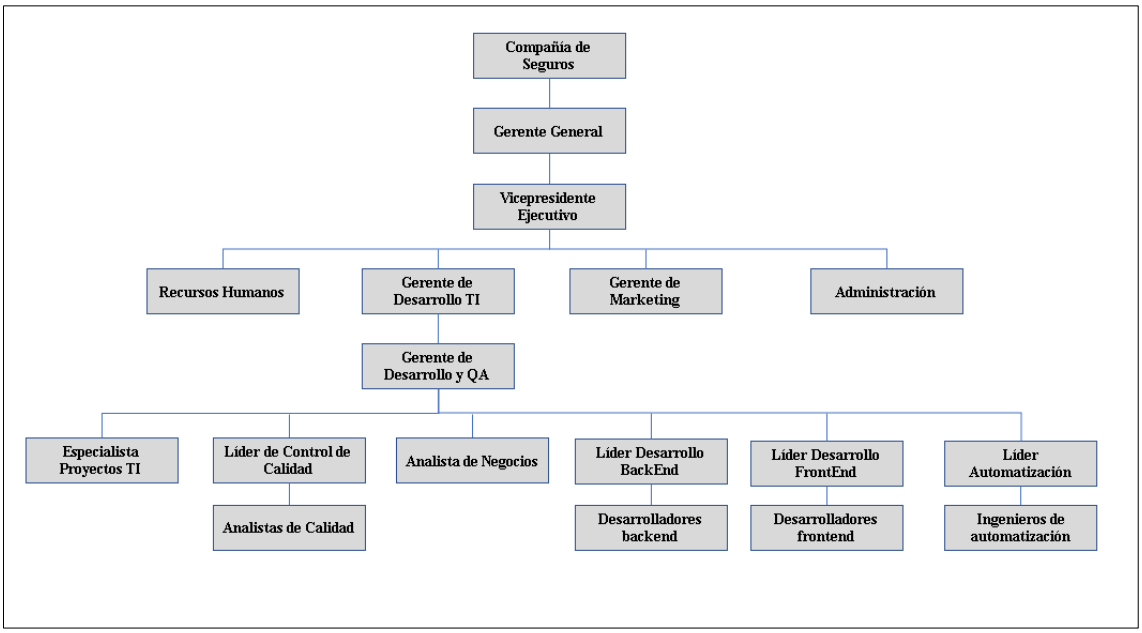
2.3. Misión

Ser una compañía que trabaja día a día por un mundo con menos preocupaciones brindando a nuestros clientes nuevos productos y cada vez mejores servicios que se alineen con los valores organizacionales que nos guían, que son: Dedicación, integridad, compromiso y excelencia.

2.4. Organización de la empresa

A continuación, en la figura 1 se detalla el organigrama de la empresa.

Figura 1: Organigrama de la compañía de Seguros



Fuente: Elaboración propia

2.5. Área, cargo y funciones desempeñadas

El autor del presente trabajo se ha desempeñado como Ingeniero de Automatización de pruebas para una compañía de seguros en el área de automatización de pruebas de software.

Las funciones del autor esta alineada al cumplimiento del proceso, lineamientos del gobierno de automatización en un entorno de trabajo ágil.

Las funciones que el autor desempeña son:

- Definición de lineamientos de gobierno de automatización Web, APIs, Móvil, Rendimiento.
- Definir el framework de automatización de pruebas Web, APIs, Móvil, Rendimiento.
- Definición, configuración de proyectos automatizados en la herramienta de integración continua Jenkins en entornos Windows y Linux.
- Realizar pilotos de herramientas de pruebas que ayuden en el proceso de pruebas.
- Encargado de dar seguimiento y asesoría a los automatizadores de los equipos donde realizan automatizaciones Web y APIs.

- Realizar la estrategia de automatización de pruebas Web, APIs, Móviles.
- Analizar los escenarios de pruebas de proyectos para realizar las automatizaciones Web, APIs, Móviles.
- Diseñar los escenarios de pruebas en lenguaje Gherkin.
- Automatizar los escenarios de pruebas.
- Participar continuamente en las ceremonias de Scrum.
- Proponer y recomendar mejoras en el proceso de gestión de la calidad.
- Dar capacitaciones sobre automatización a los analistas de pruebas funcionales.

2.6. Experiencia profesional realizada en la organización

Durante el tiempo laboral del autor en la organización tuvo su experiencia como QA Automatizador donde realizó las actividades de ejecución de pruebas funcionales en aplicaciones (Web, APIs, Móviles), el diseño en lenguaje Gherkin de escenarios y casos de pruebas para la implementación de los scripts.

El autor tuvo la oportunidad de cambiar al perfil de Ingeniero de automatización de pruebas siendo su principal función en la compañía la de evaluar proyectos para realizar las automatizaciones de las pruebas en aplicaciones (Web, APIs), realizar la construcción de los scripts, realizar las integraciones de los proyectos en la herramienta Jenkins, brindar asesorías a los automatizadores de los diferentes proyectos de la compañía donde se aplica la automatización y realizar constante seguimiento para que se cumplan los lineamientos del gobierno de automatización.

Toda la experiencia del autor en la organización fue en un marco de trabajo ágil participando en las ceremonias de Planificación, Reuniones diarias, Refinamiento, Revisión de los objetivos del sprint y Retrospectiva.

CAPÍTULO III – ACTIVIDADES DESARROLLADAS

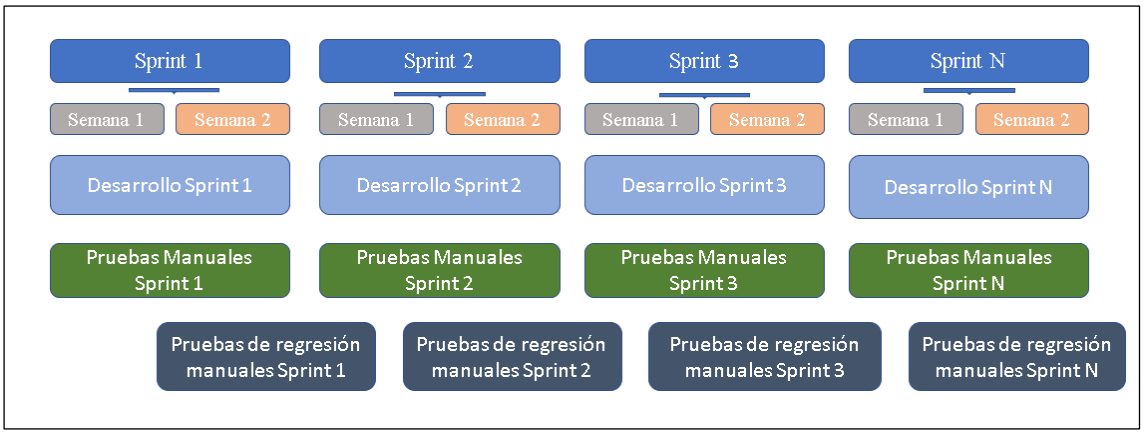
3.1. Situación problemática

La compañía de seguros cuenta con diferentes aplicaciones web donde se realizan las ventas digitales, las cotizaciones de los diferentes servicios y productos que ofrecen; dado que continuamente se deben realizar pruebas de regresión en las aplicaciones Web y APIs para asegurar el correcto funcionamiento de los flujos y no se vean impactados por los desarrollos que se realizan debido al mantenimiento y a las nuevas funcionalidades.

Dado que los analistas de calidad realizan estas pruebas de regresión y la toma de las evidencias de los paso a paso de cada ejecución de forma manual les toma considerable tiempo y esfuerzo, en consecuencia, no llegaban a terminar las pruebas en un Sprint, razón por la cual muchas veces se reducía la cobertura de pruebas de regresión para llegar a las fechas planificadas y esto aumenta el riesgo de no detectar defectos.

Esto se puede ver representado en la figura 2, donde podemos ver la planificación de las pruebas de regresión de forma manual que se ejecuta cada vez que se realiza cambios en las aplicaciones y estas pruebas de regresión van aumentando debido a que las aplicaciones van evolucionando, por lo tanto, el tiempo de realizar las ejecuciones de estas pruebas aumenta en cada Sprint.

Figura 2: Planificación de las pruebas de regresión manuales



Fuente: Elaboración propia

3.1.1. Definición del problema

En base a la situación descrita se identificaron los siguientes problemas:

- Se reducía la cobertura de las pruebas de regresión para llegar a las fechas planificadas debido a que la ejecución de las pruebas les tomaba mucho tiempo y con esto aumenta el riesgo de no detectar defectos.
- Se identificaron casos de pruebas complejas y repetitivas donde los analistas de pruebas funcionales realizan los mismos pasos con diferentes datos de entrada para un mismo proceso de inicio a fin.
- Se identificó que la toma de las evidencias de los paso a paso y el reporte de las pruebas les consumía mucho tiempo ya que esto era realizado de forma manual.
- La identificación de los errores en las pruebas de regresión se detectaba de forma tardía debido a que los analistas de pruebas funcionales realizan dichas pruebas de forma manual.
- Se identificó que no se cuenta con Scripts de automatización de pruebas para las aplicaciones Web y APIs.

3.2. Solución

3.2.1. Objetivos

Objetivo general

Implementar un framework de automatización de pruebas en un marco de trabajo ágil para mejorar el proceso de calidad de software en una compañía de seguros.

Objetivos específicos

- Definir los procesos que se desarrollan para la implementación de pruebas automatizadas aplicables a plataformas Web y APIs.
- Definir la arquitectura para las aplicaciones Web y APIs del framewrok de automatización.
- Definir la estructura de carpetas de los proyectos.
- Aplicar el enfoque de Desarrollo guiado por el comportamiento (BDD).
- Aplicar los patrones de diseño Modelo de Objetos de Página (POM) y Singleton.
- Aplicar dos de las buenas prácticas de los principios SOLID.
- Realizar el diseño de los escenarios de pruebas en lenguaje Gherkin.
- Realizar la implementación de los Scripts.
- Utilizar Jenkins, herramienta de integración continua para las ejecuciones de las pruebas automatizadas.
- Acortar el periodo que se tarda en ejecutar las pruebas de regresión.
- Proporcionar una cobertura de pruebas más amplia.
- Generar reportes de las ejecuciones de las pruebas automatizadas.

3.2.2. Alcance

El alcance de este trabajo comprende la definición de los lineamientos y técnicas del gobierno de automatización, los procesos, arquitecturas, mejores prácticas para implementar el framework de automatización de pruebas que nos permitirá construir los Scripts basándonos en el enfoque de Desarrollo guiado por el comportamiento (BDD) que tiene como propósito definir un lenguaje común entre el negocio y técnicos, para eso usamos Cucumber que nos permitirá definir nuestras funcionalidades, escenarios y casos de pruebas usando el lenguaje Gherkin y transformar estos escenarios a especificaciones ejecutables que nos permitirá integrarnos con Selenium para la interacción con los objetos de las aplicaciones web y para las APIs se usara Rest Assured para la interacción con el request, response y en ambos casos usaremos Serenity para generar documentación viva.

También se usará el patrón de diseño Modelo de Objeto de Página (POM) que nos permitirá estructurar nuestros Scripts para que sea más fácil el mantenimiento, para evitar la duplicidad de código y finalmente se integrará los proyectos con la herramienta de integración continua Jenkins creando los Pipelines con ejecución parametrizada.

3.2.3. Etapas y metodología

Este trabajo se realizó siguiendo los lineamientos de la metodología ágil Scrum; donde para cada Sprint se contó con las ceremonias de planificación, las reuniones diarias, el refinamiento, las reuniones de revisión y la retrospectiva.

A continuación, se detalla cada ceremonia realizada durante cada Sprint.

Reunión de planificación:

La reunión de planificación cuenta con una duración de 2 horas para un Sprint de 2 semanas, en esta reunión se definen las historias de usuario que entraran en el sprint backlog priorizado. Los desarrolladores, analistas de pruebas funcionales, ingeniero de automatización son los encargados de estimar y desglosar en una lista de tareas cada historia de usuario, de esta manera se sabe cuál es el total de historias que se pueden trabajar en cada Sprint por la cantidad de puntos de historias totales, para la puntuación se hace uso del método de la serie de Fibonacci. Una vez finalizada la priorización y estimación de las historias de usuario se inicia la ejecución del Sprint.

Reuniones diarias:

Durante el Sprint se lleva a cabo reuniones diarias de 15 minutos donde cada integrante del equipo comenta sobre lo que realizo el día anterior, lo que realizara el día actual y si tuviera algún impedimento, en caso de que hubiera impedimentos se cuenta con el apoyo del Scrum Master para poder continuar con las labores.

Para esta reunión diaria se cuenta con una pizarra en jira con todas las historias de usuario del sprint y con sus respectivos estados (sprint backlog, en progreso, listo para QA, en QA, terminado), en el cual cada integrante del equipo debe cambiar el estado de sus tareas según sus avances diarios y así mantener la pizarra actualizada para las reuniones.

Reunión de refinamiento:

Durante el Sprint, en la segunda semana se realiza el refinamiento que tiene una duración de 2 horas donde se revisa las historias del backlog y se definen los criterios de aceptación así asegurar que siempre estén preparados las historias que se verán en el siguiente Sprint.

Reunión de revisión:

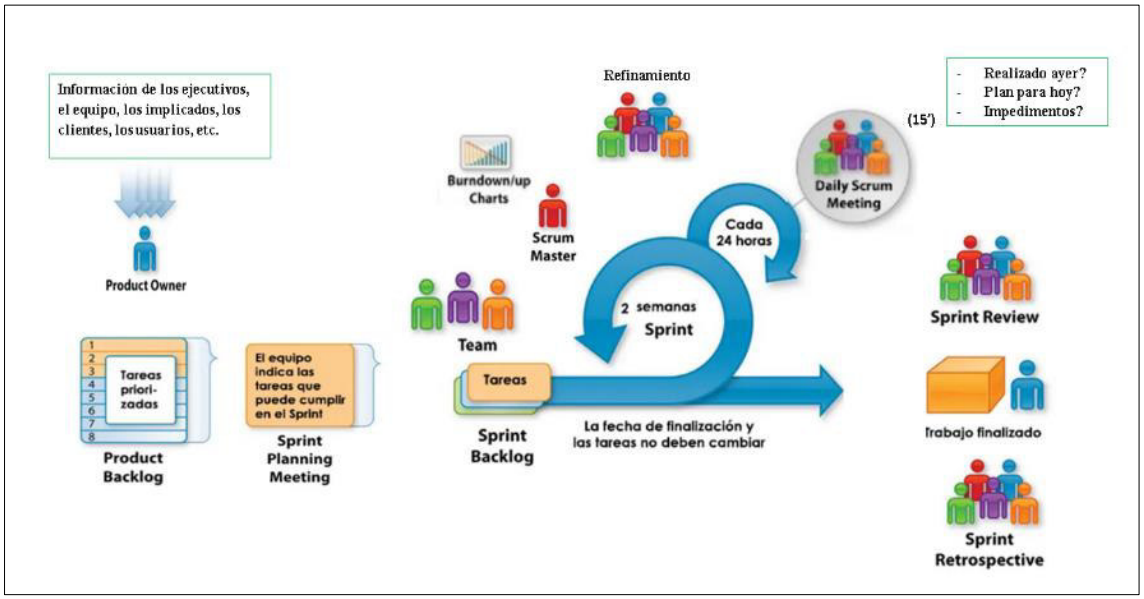
En el último día de cada Sprint se realiza la reunión de revisión, donde se muestra las historias de usuario culminadas al Product Owner, el cual debe indicar si el desarrollo de las historias cumple con los criterios de aceptación establecidos, la reunión tiene una duración de 1 hora.

Reunión de retrospectiva:

Se realiza el último día del Sprint, en esta reunión se revisa como nos ha ido en el sprint y se crea un plan de trabajo donde se plantean mejoras que serán abordadas durante el siguiente Sprint y así conseguir una mejora continua.

En la figura 3, se puede ver representado la metodología de trabajo Scrum.

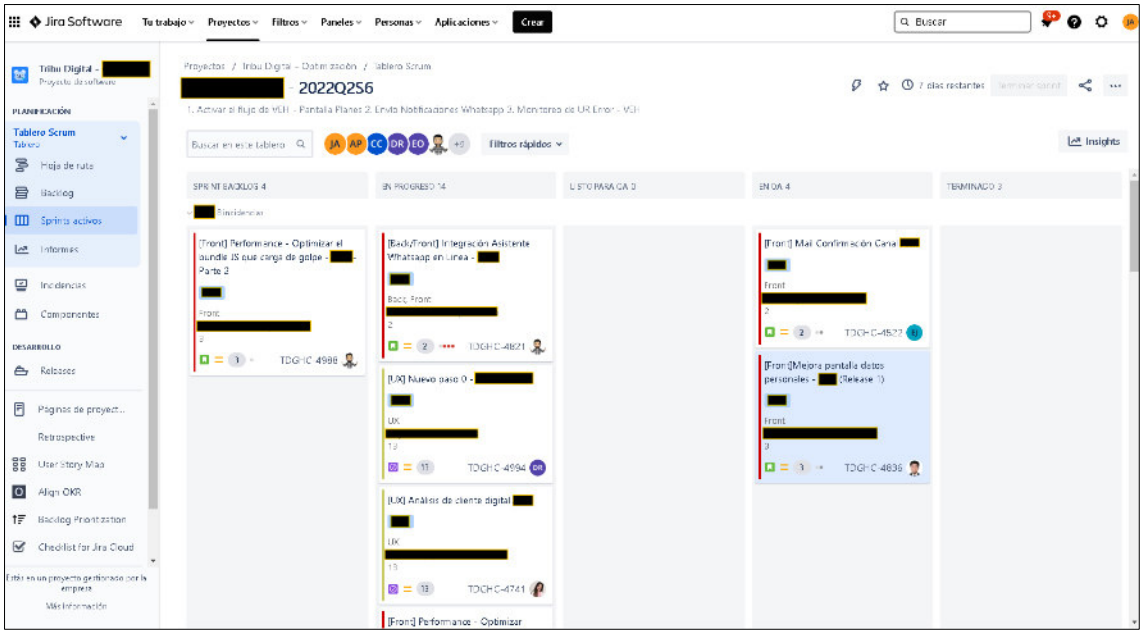
Figura 3: Flujo Scrum durante cada *S*print



Fuente: Elaboración propia

En la figura 4, podemos ver representado la creación de las historias de usuario que se trabajan en cada Sprint y los distintos estados para el seguimiento.

Figura 4: Tablero de seguimiento de las tareas en Jira



Fuente: Jira interno del proyecto

Se definió las mejores prácticas para escribir los escenarios de pruebas en los features:

- Los feature solo deben probar funcionalidades de una aplicación y no a la aplicación completa.
- Usar el mismo idioma que los clientes (Gherkin nos permite especificar el idioma ya que cuenta con más de 60 idiomas aproximadamente, si no especificamos el idioma toma por defecto el idioma inglés) ya que al usar el lenguaje que hable el cliente será más fácil la comunicación con ellos.
- Usar etiquetas, ya que nos va a permitir agrupar y organizar los escenarios y el feature.
- Evitar el uso de conjunciones en un mismo paso, ya que cada paso debe hacer una cosa; si hay un mismo paso que contiene dos acciones se debe separar mediante la palabra reservada And. Esto aumentara la capacidad de reutilización de la acción.
- Usar el Background de manera coherente.
- Escribir los escenarios de forma declarativa, ya que es el más alineado con las historias de usuario.
- Se recomienda que cada escenario no tenga When / Then anidados.

- Cada escenario debe tener un propósito o flujo único ya sea el fujo básico o alterno.
- No debe existir varios flujos descritos en un mismo escenario.

En la figura 5, se muestra la estructura del diseño de un escenario de pruebas en lenguaje Gherkin.

Figura 5: Diseño de un escenario con Gherkin

```
Feature: Emisión Seguro de Salud - AMI - TYP

  @EmisionTerceroPreexistencia
  Scenario Outline: 7- Emisión Seguro de Salud con tercero que tiene preexistencia
    Given ingresamos a la web seguro de salud
    And registramos datos del usuario "<tipoDocumento>", "<nroDocumento>", "<celular>"
    When seleccionamos un plan "<plan>"
    Then validar mensaje typ preexistencia

    Examples:
      |0|tipoDocumento|nroDocumento|celular|plan|
      |1|DNI|48792506|999999999|Regular|
```

Fuente: Código fuente del framework de automatización

Se hizo uso del xpath relativo para la localización y manipulación de los elementos de la aplicación Web:

Se hace uso de este tipo de localizador ya que nos permite ubicar elementos complejos y dinámicos en una aplicación web cuyos atributos cambian dinámicamente durante la actualización o cualquier operación en cualquier lugar de una aplicación web y nos brinda una mayor precisión al hacer uso de las condicionales (OR, AND) y los ejes al escribir el script.

Sintaxis del xpath relativo:

- **//:** Se encarga de seleccionar el nodo actual.
- **Nombre etiqueta:** Es el nombre de la etiqueta del nodo en particular.
- **@:** Selecciona el atributo.
- **Atributo:** Es el nombre del atributo del nodo.
- **Valor:** Es el valor de los atributos.

Ejemplo xapth relativo:

//div[@type=’email’ and @name=’username’]

Tabla 7: Ejes de los xpath para su construcción

TIPO DE EJE	DESCRIPCIÓN
ancestor	Permite seleccionar todos los antepasados del nodo actual.
ancestor-or-self	Permite seleccionar todos los antepasados incluyendo el nodo actual.
attribute	Permite seleccionar todos los atributos del nodo actual.
child	Permite seleccionar todos los hijos del nodo actual.
descendant	Permite seleccionar todos los descendientes del nodo actual.
descendant-or-self	Permite seleccionar todos los descendientes incluyendo el nodo actual.
following	Permite seleccionar todo en el documento HTML después de la etiqueta de cierre del nodo actual.
namespace	Permite seleccionar todos los nodos del namespace del nodo actual.
parent	Permite seleccionar el padre del nodo actual.
self	Selecciona el nodo actual.

Fuente: Elaboración propia

Para las validaciones de los resultados esperados tanto para las aplicaciones Web y APIS se realiza mediante los siguientes Assert:

- **AssertEquals:** Permite comparar los valores esperados y reales.
- **AssertNotEquals:** Es lo opuesto al AssertEquals.
- **AssertTrue:** Permite comprobar si la condición es verdadera.
- **AssertFalse:** Permite para comprobar si el valor devuelto es falso.
- **AssertNull:** Permite verificar si el objeto es nulo.

Para implementar el framework de automatización se realizaron un conjunto de actividades las cuales han sido agrupadas en las siguientes etapas:

1. **Definición de los lineamientos y arquitectura del framework:**

   Se define los lineamientos y técnicas del gobierno de automatización, estructura de carpetas y diseño de la arquitectura del framework para la automatización de pruebas de aplicaciones Web y APIs.

2. **Relevamiento de información:**

   Se realiza dos reuniones teniendo como premisa la necesidad de automatizar casos de pruebas de regresión, para lo cual se invita a los miembros del equipo de trabajo (Analista de pruebas funcionales, Product Owner y el Scrum Master) donde nos realizan una breve presentación de sus procesos macro, la cual permitirá evaluar que se puede automatizar y definir los tipos de pruebas que se automatizará.

3. **Gestión de información:**

   El ingeniero de automatización evalúa la viabilidad de la iniciativa bajo ciertos supuestos de factibilidad técnica y habilitadores, las cuales son las siguientes:

   - **Habilitador de ambiente:** Es imprescindible contar con un ambiente lo suficientemente estable para que se garantice el continuo desarrollo de la automatización de pruebas, para ello se sugiere que este ambiente de pruebas sea parecido al ambiente de producción.
   - **Habilitador de gestión:** Es el Analista de pruebas funcionales quien nos brindará los elementos necesarios para poder llevar acabo esta iniciativa, así mismo conjuntamente se definirá el alcance y la estrategia de automatización de pruebas, priorizando el backlog y escenarios de pruebas.
   - **Habilitador de conocimiento:** Son las personas con las que se puede recopilar más información sobre las historias de usuario, y estos son: Product Owner, Desarrolladores y Analistas de Pruebas Funcionales.
   - **Habilitador resolutor incidentes:** Es importante contar con este soporte durante la automatización de pruebas, ya que será nuestro apoyo a nivel

de configuración y en caso identifiquemos algún bloqueante o defecto en la aplicación, este rol lo tienen los desarrolladores.

4. Estimación de esfuerzos:

Una vez realizado el análisis de información y con el alcance definido, se procede a realizar la estimación de la iniciativa de automatización de pruebas.

Los métodos de estimaciones incluyen los siguientes:

- Estimación basada en analogías.
- Estimación por estructura.
- Estimación basada en tamaño de la historia.
- Estimaciones grupales usando Fibonacci.

5. Diseño de casos de pruebas:

El ingeniero de automatización y/o analista de pruebas funcionales realizan el diseño de los casos de pruebas en Jira o Excel usando el lenguaje Gherkin.

6. Implementación:

En este proceso se encuentran enfoques diferenciales a la hora de realizar el desarrollo de las pruebas automatizadas.

Tabla 8: Descripción de los tipos de pruebas que se automatizaran

Tipo de prueba	Descripción
Interfaz grafica	En este enfoque se utilizan herramientas que simulan las acciones que realizaría un usuario (clic en botones, escribir en una caja de texto, seleccionar un combo, etc.).
APIs	En este enfoque habitualmente las pruebas que se realizan directamente sobre servicios, a los que se les pasa una serie de argumentos para comprobar su funcionamiento.

Fuente: Elaboración propia

7. Versionamiento:

Se realiza el versionamiento de todos los proyectos en el repositorio GitHub siguiendo los lineamientos definidos de tal forma que el equipo de automatización pueda colaborar simultáneamente con diferentes automatizadores

dentro del mismo proyecto y realizar las integraciones con las herramientas Nexus y Jenkins.

8. Integración continua:

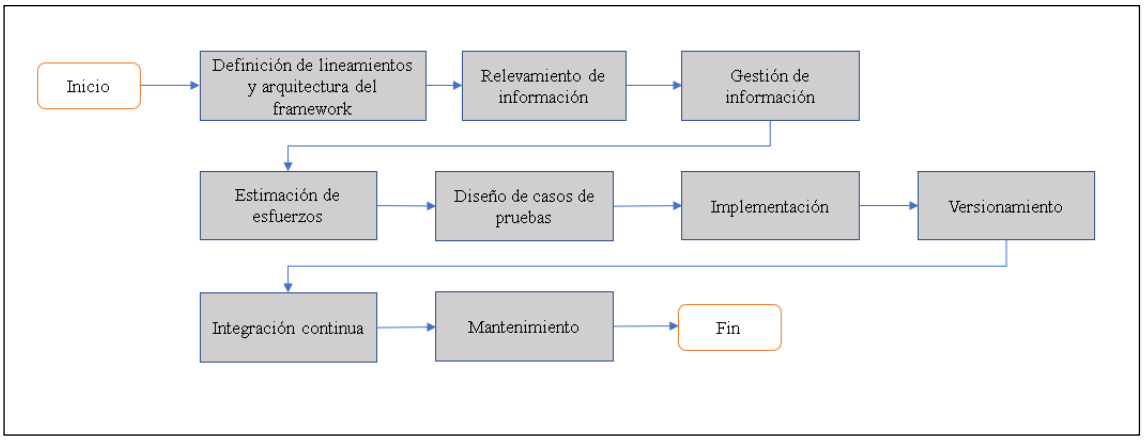
Para realizar la integración continua de los proyectos auto-def, auto-app y auto-utilitarios deberán ser creados en el repositorio de GITHUB de la empresa, estos proyectos deberán contar con las ramas master y reléase. Las ramas features solo estarán presentes en caso el proyecto aun tenga desarrollo en curso, una vez finalizado las pruebas de integración en la rama reléase, las ramas features deberán ser eliminadas.

9. Mantenimiento:

Se evalúa y soluciona los incidentes durante la ejecución de las pruebas, si el incidente es por reconocimiento de objetos y configuraciones, el ingeniero de automatización es el responsable de realizar el mantenimiento siempre y cuando no supere el medio día en esfuerzo, en caso contrario se deberá planificar su mantenimiento evaluando el impacto.

En la figura 6, se muestra el diagrama del proceso que se sigue para implementar el framework de automatización de pruebas.

Figura 6: Proceso de cómo se desarrolla la automatización de pruebas



Fuente: Elaboración del equipo de gobierno de automatización

Las actividades y entregables se realizó en 13 sprints que fueron realizados para este proyecto cuya fecha de inicio fue el 02 de marzo del 2020 y la fecha fin el 31 de agosto del 2020.

- **Sprint 1**

  Periodo: 02/03/2020 – 13/03/2020

  Se realizó la definición de los procesos y lineamientos que se desarrollaran para la implementación del framework de automatización.
- **Sprint 2**

  Periodo: 16/03/2020 – 27/03/2020

  Se realizó la definición de los procesos y lineamientos que se desarrollaran para la implementación del framework de automatización.
- **Sprint 3**

  Periodo: 30/03/2020 – 10/04/2020

  Se realizó el diseño de la arquitectura de las aplicaciones Web y APIs del framework de automatización.
- **Sprint 4**

  Periodo: 13/04/2020 – 24/04/2020

  Se realizó el relevamiento de la información de las aplicaciones a automatizar.
- **Sprint 5**

  Periodo: 27/04/2020 – 08/05/2020

  Se realizó la gestión de la información y la estimación de esfuerzos.
- **Sprint 6**

  Periodo: 11/05/2020 – 22/05/2020

  Se realizó el diseño de los escenarios de pruebas de la aplicación Web y APIs.
- **Sprint 7**

  Periodo: 25/05/2020 – 05/06/2020

  Se realizó la gestión de los ambientes de pruebas y la implementación de los Script de la aplicación Web y su versionamiento.

- **Sprint 8**

  Periodo: 08/06/2020 – 19/06/2020

  Se realizó la implementación de los Script de la aplicación Web y su versionamiento.
- **Sprint 9**

  Periodo: 22/06/2020 – 03/07/2020

  Se realizó la implementación de los Script de la aplicación Web y versionamiento.
- **Sprint 10**

  Periodo: 06/07/2020 – 17/07/2020

  Se realizó la implementación de los Script de las APIs y versionamiento.
- **Sprint 11**

  Periodo: 20/07/2020 – 31/07/2020

  Se realizó la implementación de los Script de las APIs y versionamiento.
- **Sprint 12**

  Periodo: 03/08/2020 – 14/08/2020

  Se realizó la creación y configuración de los Pipelines de los proyectos de automatización en Jenkins.
- **Sprint 13**

  Periodo: 17/08/2020 – 31/08/2020

  Se realizó el mantenimiento y ejecución de los scripts automatizados.

Para dar seguimiento al proyecto se utilizó la herramienta Jira, donde se registran las historias de usuarios, actividades, defectos, impedimentos, escenarios de pruebas; también realizando las ceremonias de la metodología Scrum que permiten tener una visión global del avance del proyecto como también una visión detallada del avance del Sprint.

3.2.4. Fundamentos utilizados

Automatización de pruebas

La automatización de pruebas utiliza herramientas para controlar automáticamente la ejecución de productos de software mediante la creación e implementación de Scripts en un lenguaje de programación en específico (Java, JavaScript, Ruby, Python, etc.), lo que le permite comparar los resultados que obtiene con los resultados esperados.(Crespo, 2018).

Cucumber

Es una de las herramientas que podemos utilizar para automatizar nuestras pruebas BDD (Desarrollo guiado por el comportamiento), ya que permite ejecutar descripciones funcionales en texto plano como pruebas de software automatizadas. Cucumber fue creado en 2018 por Aslak Hellesoy y está escrito en Ruby, aunque tiene implementaciones para casi cualquier lenguaje de programación (Ruby, Java, Groovy, JavaScript, Python, etc.). Para la definición de los requisitos Cucumber usa el lenguaje Gherkin, el cual permite traducir las especificaciones en un lenguaje que personas de negocio y técnicas las puedan entender. (SmartBear, Cucumber Guide, 2019)

Gherkin es un lenguaje DSL legible para personas de negocio y técnicas, que permite definir el comportamiento del software sin detallar como se está implementando, además permite realizar la documentación de las funcionalidades mientras se escribe los escenarios de pruebas. (SmartBear, Gherkin Syntax, 2019). Principales palabras clases:

- **Feature:** Indica las funcionalidades de la aplicación que se van a probar, debe ser un título claro que haga referencia a la funcionalidad.
- **Scenario:** Describe cada escenario de la funcionalidad que vamos a probar.
- **Given:** Provee las precondiciones, el contexto para el escenario.
- **When:** Son las acciones o eventos que se van a realizar.
- **Then:** Especifica las validaciones que se van a realizar con los resultados esperados.

- **And:** Nos permite aumentar la legibilidad y pueden ser usadas en vez de definir repetidas veces los pasos anteriores.
- **Background:** Steps ejecutados antes de cada escenario.
- **Scenario Outline:** Indica que el escenario se ejecutara con ejemplos.
- **Examples:** Indica que se usaran parámetros para la ejecución del escenario.

Como funciona:

- Para realizar las automatizaciones de las pruebas se utiliza la herramienta Cucumber con el lenguaje Gherkin.
- Lee los archivos .features de la ruta indicadas.
- Cada escenario de pruebas puede tener uno más escenarios.
- Por cada uno de los escenarios se ejecuta los Steps.

Beneficios:

- Es útil para involucrar a las partes interesadas de negocios que no pueden leer fácilmente el código.
- Cucumber se centra en la experiencia del usuario final.
- La forma de realizar estas pruebas permite una reutilización más fácil del código en las pruebas.
- Permite realizar las configuraciones y ejecuciones de forma rápida y sencilla.
- En lugar de escribir la prueba puramente en código con Cucumber se comienza escribiendo una historia de usuario legible por humanos.

En la figura 7, se muestra el diseño del escenario de prueba usando las palabras reservadas de Gherkin y los tags.

Figura 7: Diseño de un escenario usando Gherkin

```
@Regresion
Feature: Emisión Seguro de Salud

  @EmisionTerceroRegistrado
  Scenario Outline: 1- Emisión Seguro de Salud con tercero registrado sin familiares
    Given ingresamos a la web seguro de salud
    And registramos datos del usuario "<tipoDocumento>", "<nroDocumento>", "<celular>"
    When seleccionamos un plan "<plan>"
    And registramos datos personales "<correo>"
    And respondemos con no todas las preguntas
    And registramos la direccion "<dpto>", "<provincia>", "<distrito>", "<direccion>"
    And realizamos el pago "<nroTarjeta>", "<fVen>", "<ccv>", "<nombre>", "<apellido>", "<correo>"
    Then validar emision del seguro de forma satisfactoria

    ###DATOS###@DataPrueba|1@01-TerceroRegistrado
    Examples:
      |0|tipoDocumento|nroDocumento|celular  |plan   |correo       |dpto|provincia|distrito  |direccion |
      |1|DNI          |47474747    |999999999|Regular|abc@gmail.com|LIMA|LIMA     |LIMA      |San miguel|
```

Fuente: Elaboración propia

Serenity

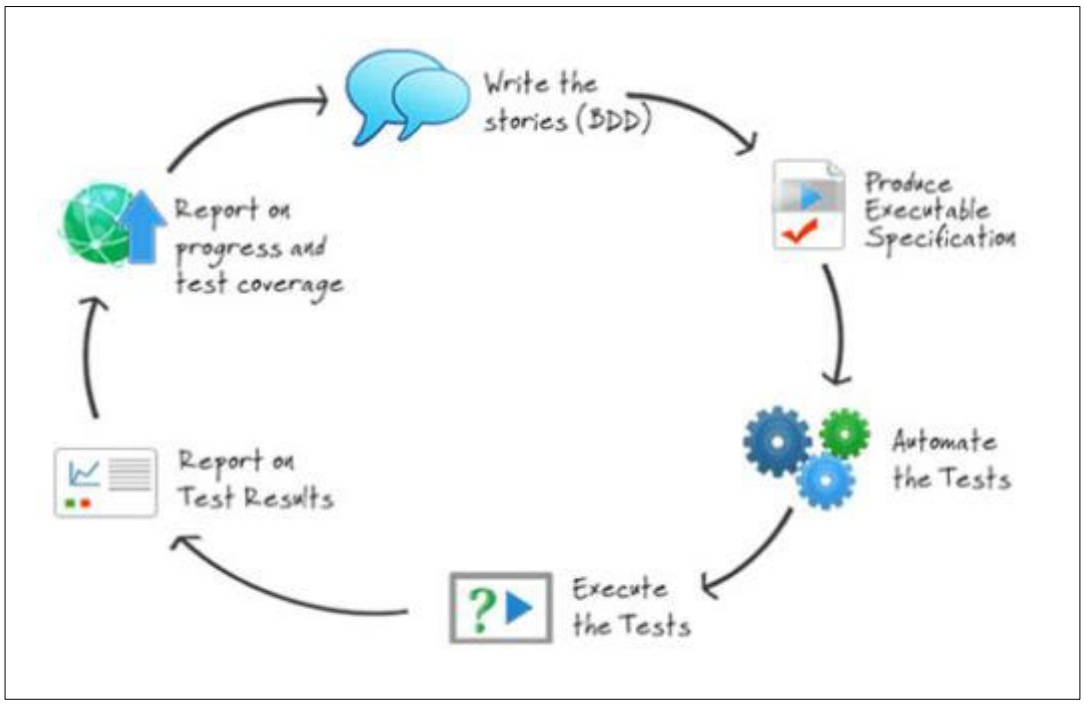
Es una biblioteca de código abierto que nos ayuda a escribir pruebas de aceptación automatizadas de alta calidad más rápidamente y generar informes ilustrativos y narrativos sobre sus pruebas. (Furguson, 2014)

Sus características principales son:

- Permite redactar pruebas flexibles y de fácil de mantenimiento.
- Permite crear informes descriptivos e ilustrativos sobre las pruebas.
- Permite personalizar las pruebas automatizadas según las necesidades del proyecto.
- Permite dar seguimiento de cuánto de la aplicación se están probando.
- Muestra el resultado de la prueba, incluidos los tiempos de ejecución y los mensajes de errores si falla una prueba.

En la figura 8, se muestra el flujo de trabajo de la herramienta Serenity.

Figura 8: Flujo de trabajo con la librería Serenity



Fuente: https://serenity-bdd.info/

BDD (Desarrollo guiado por el comportamiento)

Consiste básicamente en una estrategia de desarrollo y lo que plantea es la definición de los requisitos desde el punto de vista del comportamiento de la aplicación, desde el negocio, creado en un lenguaje común lo que mejora la comunicación entre los equipos para el negocio y los técnicos. El objetivo principal es que el equipo explique en detalle el funcionamiento de la aplicación a desarrollar, y sea comprensible por todos los miembros del equipo. (North, 2006)

Para definir y diseñar los casos usando BDD para una historia de usuario se realiza bajo las palabras reservadas Given, When, Then. Para empezar a hacer BDD, solo se necesita conocer las siguientes palabras reservadas:

Feature: Es la funcionalidad que se va a probar.

Scenario Outline: Describe cada escenario que se va a probar, cada escenario es una secuencia de pasos y esos pasos son reutilizables en otros escenarios que realice la misma acción, pero con otro resultado esperado.

Given: Son las precondiciones, describe el contexto de un escenario.

When: Son las diferentes acciones que se van a realizar.

Then: Se realiza las verificaciones de los resultados esperados.

Background: Se utiliza cuando se tiene un paso común en todos los escenarios

And: Este elemento se introduce para hacer más fluida la escritura

Tag(@): Se utiliza para etiquetar los escenarios dentro de una feature.

Esta estrategia encaja bien en el marco de trabajo ágil, ya que generalmente se especifican los requisitos como historias de usuario; estas historias deben tener especificado sus criterios de aceptación de las cuales parten las pruebas de aceptación y pueden ser redactadas en el lenguaje Gherkin.

Singleton

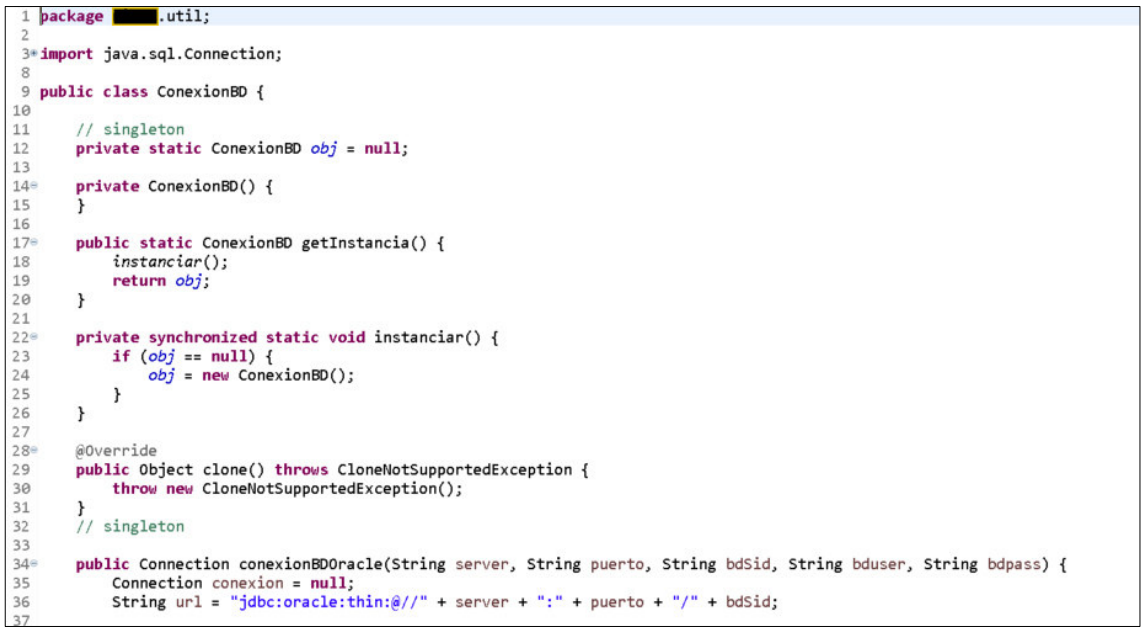
Este patrón se utiliza ya que nos permite restringir la creación de objetos pertenecientes a una clase. La idea es tener una clase que solo tenga una instancia a lo largo de toda nuestra aplicación y esto es importante que sea así porque si hubiese más de una instancia esto generaría problemas, usualmente este patrón se utiliza para optimizar recursos. (Fagua Barrera, 2015)

Para implementar el patrón Singleton se requiere al menos 3 elementos, las cuales son:

- **Un constructor privado:** Así evitamos que se creen por terceros ejecuciones.
- **Un campo estático que contiene su única instancia:** Hace referencia al objeto que vamos a crear a través del constructor.
- **Un método estático público:** Instancia el objeto la primera vez y lo almacena en una variable.

En a la figura 9, se muestra la forma de implementar el patrón Singleton.

Figura 9: Implementación del patrón Singleton



```
1  package [redacted].util;
2
3  import java.sql.Connection;
8
9  public class ConexionBD {
10
11     // singleton
12     private static ConexionBD obj = null;
13
14     private ConexionBD() {
15     }
16
17     public static ConexionBD getInstancia() {
18         instanciar();
19         return obj;
20     }
21
22     private synchronized static void instanciar() {
23         if (obj == null) {
24             obj = new ConexionBD();
25         }
26     }
27
28     @Override
29     public Object clone() throws CloneNotSupportedException {
30         throw new CloneNotSupportedException();
31     }
32     // singleton
33
34     public Connection conexionBDOracle(String server, String puerto, String bdSid, String bduser, String bdpass) {
35         Connection conexion = null;
36         String url = "jdbc:oracle:thin:@//" + server + ":" + puerto + "/" + bdSid;
37
```

Fuente: Código fuente del framework de automatización

Principios SOLID

Nos brinda las buenas prácticas de cómo debemos construir nuestro framework, como lo debemos desarrollar, son lineamientos generales muy básicos pero que nos van a permitir realizar un código más limpio. (Oloruntoba, 2021)

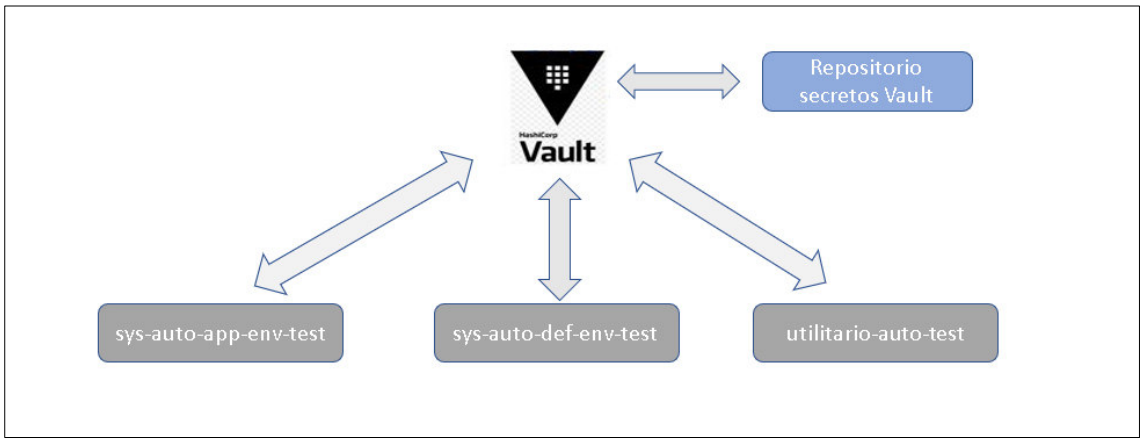
- **Principio de responsabilidad única:** Consiste en que siempre tenemos que otorgarle una única responsabilidad un único propósito porque en la medida que nosotros empezamos a generar código que trata de abarcar distintas necesidades en la misma porción del código lo estamos tornando inmanejable, muy difícil de mantener.
- **Principio abierto - cerrado**: Este principio consiste en que nuestros métodos deben quedarse abierto para su extensión, pero cerrada para su modificación, esto quiere decir que se debe adaptar a nuevos requerimientos sin tener que modificar su estructura interna.

Vault

Es una herramienta que se utiliza para acceder a información secreta de forma segura, en este caso accederemos a las contraseñas de las aplicaciones web y las credenciales para las APIs y así no tener informaciones confidenciales en el código fuente del proyecto. (Dadgar, 2000)

En la figura 10, se muestra la forma de interacción de la herramienta Vault con los proyectos.

Figura 10: Interacción entre los proyectos y Vault



Fuente: Elaboración propia

Selenium

Esta es una herramienta principalmente para ejecutar pruebas funcionales como pruebas de compatibilidad y pruebas de regresión en un navegador web; puede grabar, reproducir pruebas en varios lenguajes de programación como JavaScript, Ruby, Java, Python, C#, etc.

Las ejecuciones de los scripts de pruebas se pueden realizar en los diferentes navegadores web (Chrome, Mozilla, Explorer) y en diferentes sistemas operativos como Windows, Linux y Mac. (Selenium, s.f.)

Selenium está dividido en 3 principales componentes:

- **Selenium IDE:** Es una extensión del navegador Firefox que le permite grabar, depurar, editar y reproducir las pruebas en el navegador Firefox sin necesidad de codificar.

- **Selenium WebDriver:** Se realiza mediante código, se crea proyectos escalables para las pruebas en diferentes ambientes, esto se debe a que se puede realizar utilizando lenguajes de programación para la construcción de los scripts de pruebas.
- **Selenium Remote control:** Ayuda a realizar las ejecuciones de las pruebas automatizadas a través de diferentes navegadores y sistemas operativos.

Jenkins

Es una herramienta de integración continua de código abierto que nos permite ejecutar las pruebas de nuestros proyectos en un entorno independiente, archivando los resultados de cada Build y generando estadísticas. Las ejecuciones pueden programarse periódicamente, ya sea bajo demanda o mediante disparadores tras realizar un commit en el repositorio del proyecto. (Jenkins, s.f.)

Jenkins nos permite realizar las siguientes acciones:

- La construcción continua y pruebas automatizadas de proyectos de software.
- Orquestación de pruebas.
- Monitorización y seguimiento de la ejecución de servicios externos.
- Despliegue automático.
- Permite instalar una enorme cantidad de plugin para integrarse con otras herramientas.
- Mantiene un histórico de la ejecución y el resultado de estos.

Rest Assured

Es una biblioteca escrita en lenguaje Java y diseñado para simplificar las pruebas sobre servicios basados en REST ya que la sintaxis es muy natural y legible para los humanos por que sigue el enfoque de Desarrollo guiado por el comportamiento (BDD).

Proporciona un DSL descriptivo (lenguajes específicos de dominio) de acuerdo con el formato Gherkin (Given, When, Then) fácil de usar que muestra una unión a un punto de conexión HTTP y da los resultados esperados.

- **Given:** Permite establecer un fondo, aquí se pasa los encabezados de las solicitudes, la consulta y el parámetro de la ruta, el cuerpo y las cookies.
- **When:** Realiza la acción, envía el request para obtener el response.
- **Then:** Devuelve el resultado para poder acceder a la respuesta y comprobar su contenido.

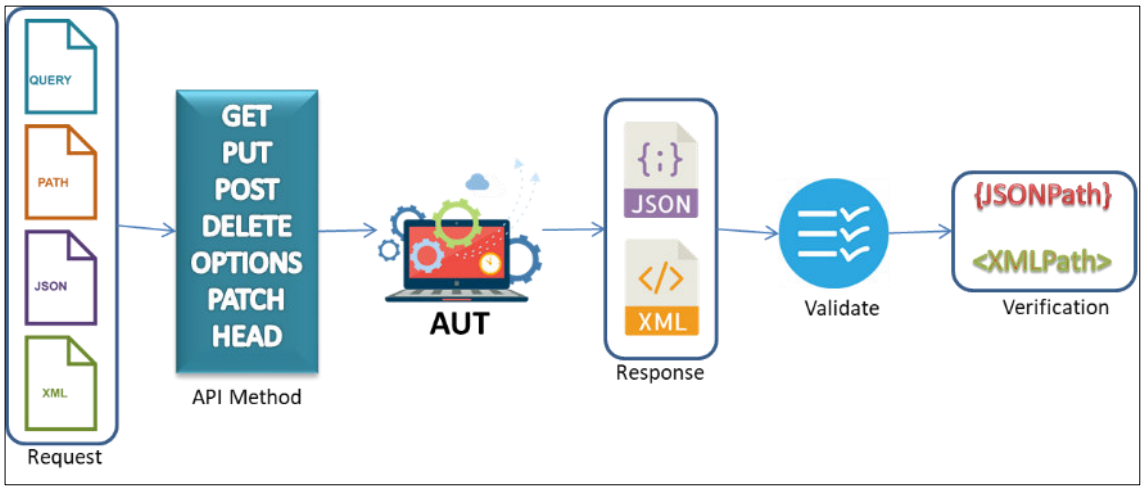
Este framework soporta peticiones POST, GET, PUT, DELETE, OPTIONS, PATCH, HEAD y además una vez realizada las peticiones permite realizar validaciones sobre la respuesta obtenida utilizando comparadores que soportan la legibilidad de las pruebas; también se puede extraer el cuerpo de la respuesta si se necesita realizar una validación más compleja. (Haleby, s.f.)

Actualmente, sus principales ventajas son:

- Validar el estado del código, del mensaje e incluso se puede ver el cuerpo de la respuesta, se tiene el control del código.
- Son fáciles de integrar con pruebas de todo tipo: funcionales, unitarias, integradas, etc.
- Permite integrar con Jenkins fácilmente
- Permite combinar con pruebas automatizadas de aplicaciones web y no requiere de herramientas externas para ejecutarse.

En la figura 11, se muestra el flujo de trabajo de una API

Figura 11: Flujo de trabajo con la librería Rest Assured



Fuente: https://rest-assured.io/

Modelo de Objeto de Página (POM)

Este patrón se utiliza para mapear las páginas de las aplicaciones web a clases page que permitan aislar las acciones de las diferentes páginas y a la vez agrupar todos los webElements de una página y las acciones que se pueden llevar a cabo en una misma clase y así mejorar el mantenimiento de estas, ya que un punto muy importante al automatizar aplicaciones web es que pueden sufrir cambios, y si la implementación manipula directamente los webElements de las páginas, estos serán muy frágiles y van a requerir el doble de mantenimiento. (POM, 2014)

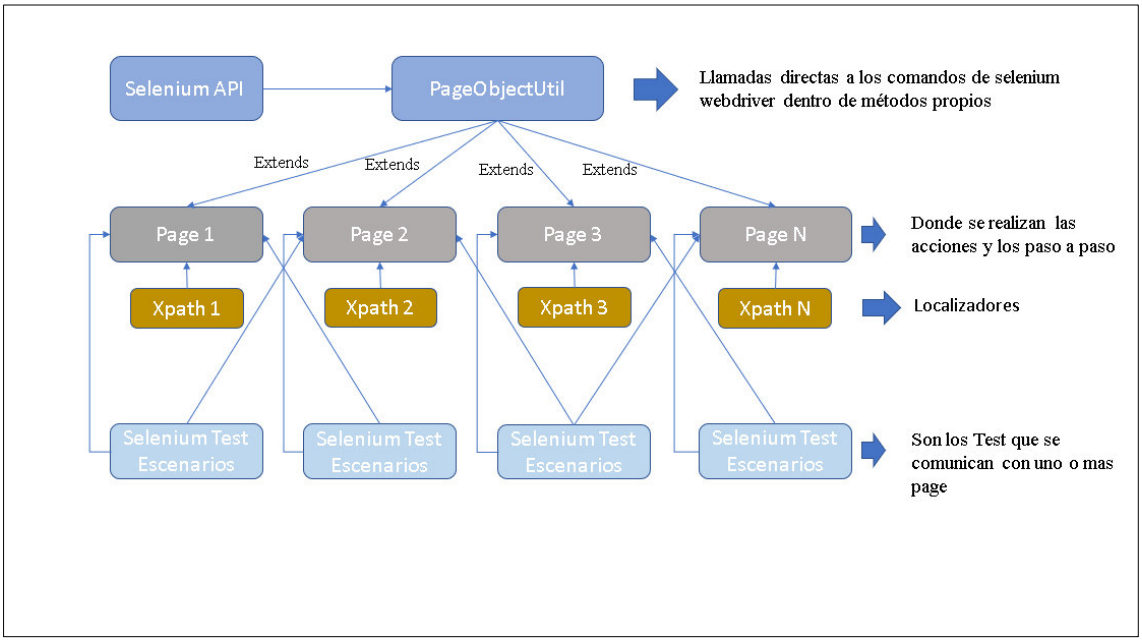
Cuando se realiza el diseño de una prueba automatizada para una aplicación web se debe localizar sus webElements para poder realizar una acción, como hacer clic en botones, escribir en cuadro de textos, seleccionar un valor en un combo, etc.

El patrón POM se encarga de encapsular el comportamiento de las páginas o una parte de esta, la cual permite escribir pruebas y manipular los elementos de una página sin tener que lidiar con el HTML. En resumen, el patrón POM se utiliza para hacer pruebas funcionales automatizadas de forma más limpia encapsulando información de la aplicación ya que un page puede ser reutilizado en todas las pruebas que se requiera, la idea es modelar las páginas de las aplicaciones web y sus comportamientos para así lograr pruebas claras de escribir, entender y mantener.

En la siguiente figura 13, se visualiza que se tiene diferentes pruebas y lo que va a hacer es comunicarse el test con las page directamente, estos page son las que van a interactuar con las páginas (HTML) a través de Selenium. Por lo tanto, si cambia la página (HTML) no se va a tener que cambiar en los test ya que el test va a ser el mismo, lo que se va a tener que modificar son los page.

En la figura 12, se muestra el diagrama del patrón de diseño Modelo de Objeto de Página.

Figura 12: Diagrama del patrón Modelo de Objeto de Página



Fuente: Elaboración propia

Metodología scrum

Scrum es un marco de trabajo más usada en diversos proyectos de desarrollo, es un marco bastante adaptable y rápido que ayuda a los equipos, las personas y las organizaciones a agregar valor al negocio en corto tiempo ya que es de 2 a 3 semanas el tiempo que dura un sprint y se puede entregar valor.

Scrum tiene cuatro eventos formales para la revisión y coordinación dentro de un evento en el sprint. Estos eventos funcionan ya que implementan los pilares de la metodología scrum las cuales son: inspección, adaptación y transparencia. (Sutherland, 2020)

- **Transparencia:** Donde los procesos y las actividades deben ser visibles tanto para aquellas personas que realizan el trabajo, así como para las personas que no realicen el trabajo.
- **Inspección:** Los artefactos de Scrum y el progreso hacia las metas y objetivos acordados deben verificarse de manera regular y diligente para detectar desviaciones o posibles problemas inesperados.
- **Adaptación:** Si algún aspecto del proceso se desvía fuera de los limites definidos o el resultado del producto es inaceptable, entonces se debe

ajustar los procesos aplicados, esto debe realizarse lo antes posible para minimizar la desviación adicional.

Los valores de Scrum son los siguientes.

- Compromiso
- Coraje
- Enfoque
- Apertura
- Respeto

Los roles de Scrum.

- **Product Owner:** Este rol es el responsable de impulsar el máximo valor comercial del proyecto definiendo, aclarando y priorizando los requisitos de los proyectos.
- **Scrum Master:** Este rol es el responsable de facilitar las ceremonias, asegurar que el equipo siga los lineamientos y prácticas de la metodología Scrum y tenga un ambiente favorable sin impedimentos para llevar a cabo el proyecto.
- **Equipo Scrum:** Este es el grupo de personas responsables de entender los requerimientos establecidos por el Product Owner y de implementar los entregables de los proyectos.

Eventos de Scrum.

Scrum tiene ceremonias predefinidas para crear regularidad y minimizar la necesidad de realizar reuniones que no están definidas en Scrum.

- **Sprint:** Son las iteraciones de 2 o 3 semanas dentro de un proyecto Scrum con un objetivo claro donde se implementan los incrementos de los productos.
- **Planificación:** Es el evento donde todo el equipo se reúne para definir y estimar las historias de usuario que se trabajaran durante los sprint.

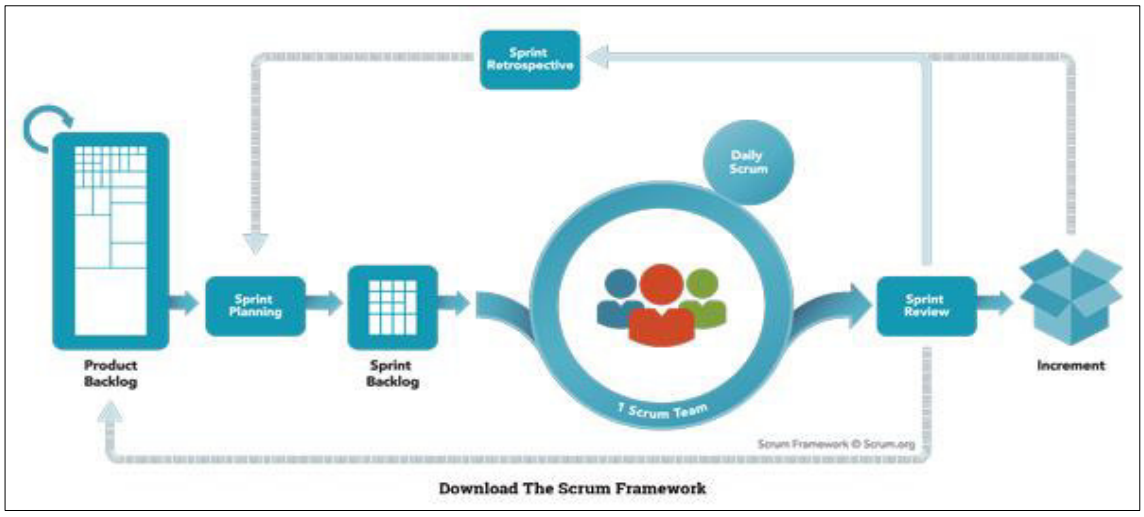
- **Daily:** Son las reuniones diarias donde el equipo de desarrollo se reúne y responden tres preguntas: Que hice ayer, que hare hoy y si tiene algún impedimento; esta reunión no debe durar más de 15 minutos.
- **Refinamiento de historias:** Es la ceremonia donde se reúne todo el equipo para revisar, detallar y definir los criterios de aceptación de las historias de usuario que se trabajaran en el siguiente sprint y así identificar posibles dudas, dependencias o impedimentos.
- **Revisión de Sprint:** Es la ceremonia donde se realiza la presentación de los resultados y objetivos logrados durante el sprint.
- **Retrospectiva:** Es la ceremonia que se realiza al final de cada sprint donde todo el equipo se reúne y se realiza 3 preguntas: que funciono, que no funciono y que se puede mejorar; a partir de las respuestas se identifican los factores más importantes y se buscan soluciones para aplicar las mejoras en cada sprint de forma gradual.

Los artefactos de Scrum

- **Product Backlog:** Es la lista de requerimientos ordenados y priorizados del proyecto.
- **Sprint Backlog:** Son los elementos de la lista del producto seleccionados para ser implementados en el Sprint.

En la figura 13, se muestra el flujo de trabajo con la metodología Scrum.

Figura 13: Flujo de trabajo Scrum



Fuente: https://www.scrum.org/resources/blog/que-es-scrum

Maven

Es una herramienta de código abierto para la gestión y construcción de proyectos de software, tiene un modelo de configuración y construcción basado en un formato XML. (Maven, 2000)

El ciclo de vida Maven es:

- **Validación:** Valida si la estructura del proyecto es correcta.
- **Compilación:** Genera los ficheros compilando el código fuente.
- **Prueba:** Ejecutar pruebas unitarias para el proyecto.
- **Paquete:** Se encarga de empaquetar el código compilado en un formato distribuible como .jar o .war.
- **Prueba de integración:** Realiza las pruebas de integración del proyecto.
- **Verificar:** Ejecuta comprobaciones para verificar que el proyecto sea válido y cumpla con los estándares de calidad.
- **Instalar:** Instala el código empaquetado en el repositorio local maven.
- **Implementar:** Realiza la copia del código empaquetado al repositorio remoto para compartirlo con otros desarrolladores.

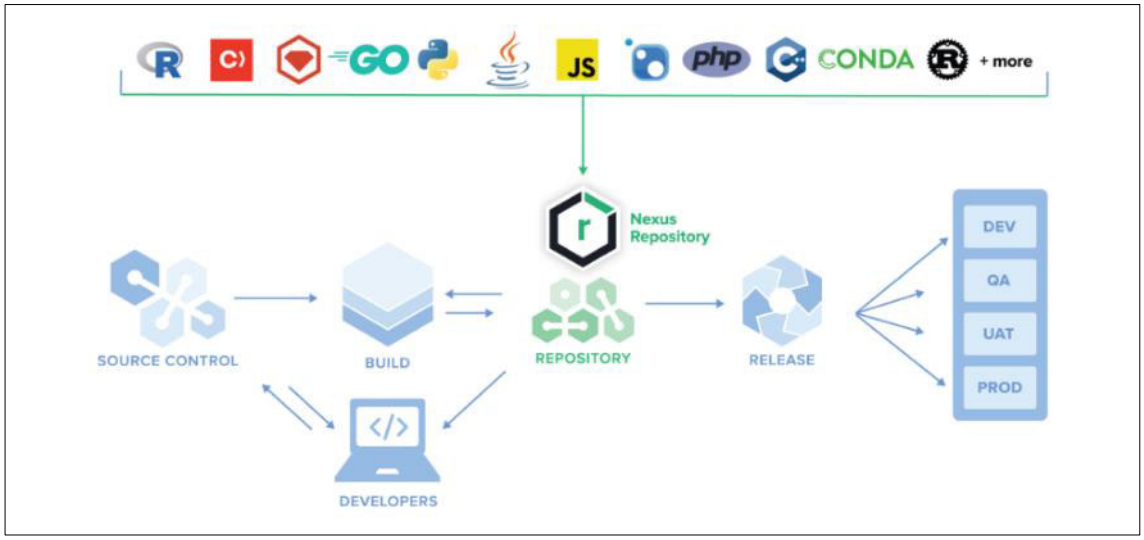
Además, Maven nos provee de un repositorio donde podemos alojar, mantener y distribuir los artefactos permitiendo una gestión correcta de las librerías, proyectos y dependencias.

Nexus

Es un sistema de control y almacenamiento de librerías, ya que permite centralizar en un único punto la gestión de las dependencias. Permite el versionado de artefactos gestionando los dos estados de desarrollo de un artefacto Snapshot y release. (Nexus, s.f.)

En la figura 14, se muestra el diagrama de organización de la herramienta Nexus.

Figura 14: Diagrama y organización del repositorio Nexus



Fuente: https://www.sonatype.com/products/nexus-repository

3.2.5. Implementación de las áreas, procesos, sistemas y buenas prácticas

Para la implementación del framework de automatización de pruebas de las aplicaciones Web y APIs se siguió la metodología y etapas definidos en la sección 3.2.3.

3.2.5.1 Definición de los lineamientos y arquitectura del framework:

Se define la estructura de las diferentes capas del framework

Definición de los proyectos:

- **sys-auto-def-env:** Este proyecto va a contener las clases y paquetes orientadas a la capa de negocio, se incluirán paquetes orientados a la creación de escenarios, definitions, drivers de los navegadores, archivos de configuración del proyecto y para la integración con Jenkins, manejo de la data de prueba y ejecutores de los escenarios. El nombre del proyecto deberá tener la siguiente nomenclatura.

  Web: ventadigital-auto-def-web-test

  APIs: cotizarseguro-auto-def-api-test

- **sys-auto-app-env:** Este proyecto va a contener las clases y paquetes orientados a la interacción con el sistema a automatizar (web y APIs), en este proyecto se incluirá los mapeos de los objetos, las acciones de estos y los paso a paso para ser utilizados en la ejecución de los escenarios. El nombre del proyecto deberá tener la siguiente nomenclatura.

  Web: ventadigital-auto-app-web-test

  APIs: cotizarseguro-auto-app-api-test

- **utilitarios-auto:** Este proyecto utilitario estará dedicado a dar soporte a todos los proyectos de automatización. El proyecto contiene clases con utilidades genéricas (conexiones a base de datos, conexiones a las APIs, métodos de escritura y lectura de los archivos Excel, métodos de interacción con las páginas web, tales como: devolver valores (devolver un booleano si el elemento de la pantalla es visible o no, devolver el texto del elemento) o para realizar acciones (hacer clic, escribir en un campo de texto, realizar scroll, etc.).

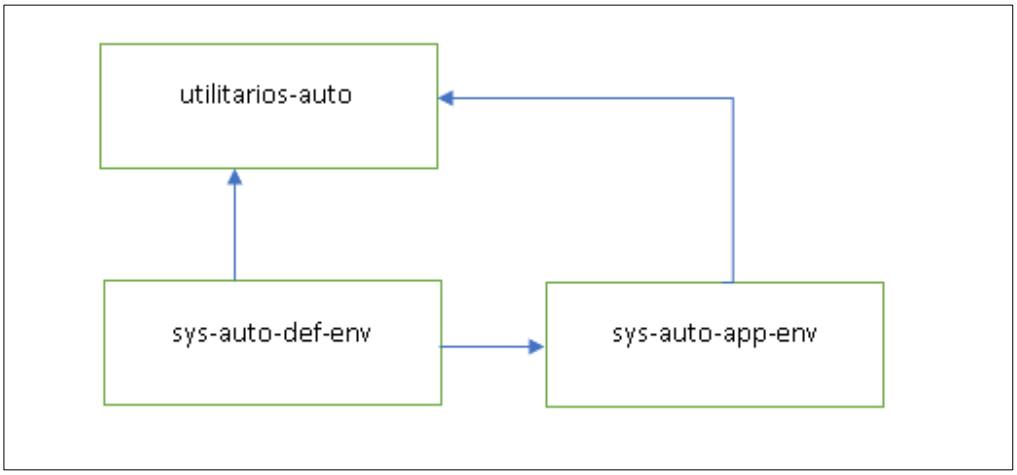
Dependencia entre proyectos:

Donde el proyecto:

- **sys-auto-def-env** incluirá como dependencias a los proyectos **utilitarios-app y sys-auto-app-env**.
- **sys-auto-app-env** incluirá como dependencia al proyecto **utilitarios-app.**

En la figura 15, se puede ver las dependencias entre los proyectos.

Figura 15: Dependencias entre proyectos



Fuente: Elaboración del equipo de gobierno de automatización

Contenido y estructura de carpetas de los proyectos:

El proyecto contiene una estructura base que servirá de modelo para la construcción de nuestras pruebas automatizadas.

Proyecto: sys-auto-def-env

Paquete: src/test/java

- **empresa.test.run**: Este paquete va a contener las clases Java para realizar la ejecución de los escenarios, esta clase invoca al paquete de definitions para su ejecución.
- **empresa.test.definition:** Son las clases Java intermediarias entre la capa de negocio (en Cucumber donde cada línea del escenario Gherkin de la feature mapea a un método (estos métodos son reutilizados en diferentes escenarios) con ayuda de las anotaciones @Given, @When, @Then) y la interacción con el sistema (invoca a las clases step del

proyecto sys-auto-app-env), esta clase contiene componentes reutilizables y también se encarga de realizar las validaciones de los escenarios y así mismo de la comunicación con las fuentes de datos para su ejecución.

- **empresa.test.inout**: Contiene las clases Java para el manejo de la data de prueba (data driven).
- **empresa.test.util:** Contiene las clases Java que contienen utilitarios propios del proyecto, esta clase incluye los métodos recursivos, variables propias como nombre de rutas, equivalencias parametrizadas propias del proyecto.
- **empresa.test.query:** Clases Java que incluyen queries con lógicas de negocio que interactúan con los steps del proyecto.

Paquete: src/test/resources

- **feature:** Esta carpeta contiene los archivos feature del proyecto que contienen los escenarios de pruebas redactados en lenguaje Gherkin, se requiere descargar Cucumber para usar la extensión en eclipse.
- **driver:** Contiene los ejecutables drivers de los exploradores web (Chrome, Mozilla, etc.) usado en las pruebas, esta carpeta también debe contener las versiones para la ejecución con Jenkins (Linux, Windows).
- **datadriven:** Se encuentra la carpeta plantilla la cual contiene el Excel base (sin resultados) para el manejo de la data de prueba. Hay que considerar que en la ejecución de proyecto desde Jenkins el Excel plantilla será copiado a una ruta externa y a partir de ahí ser consumido, el archivo Excel de data de prueba deberá alojarse en una ruta compartida.
- **query:** Contiene las sentencias o consultas de la base de datos necesarias para la ejecución de los escenarios.
- **Secrets.properties:** Este archivo de propiedades es donde se registran los accesos necesarios para la ejecución del proyecto. Este archivo servirá solo para trabajar localmente en el desarrollo del script, por

ende, deberá estar registrado en el archivo .gitignore para que al momento de subir el código al repositorio GitHub ser excluido.

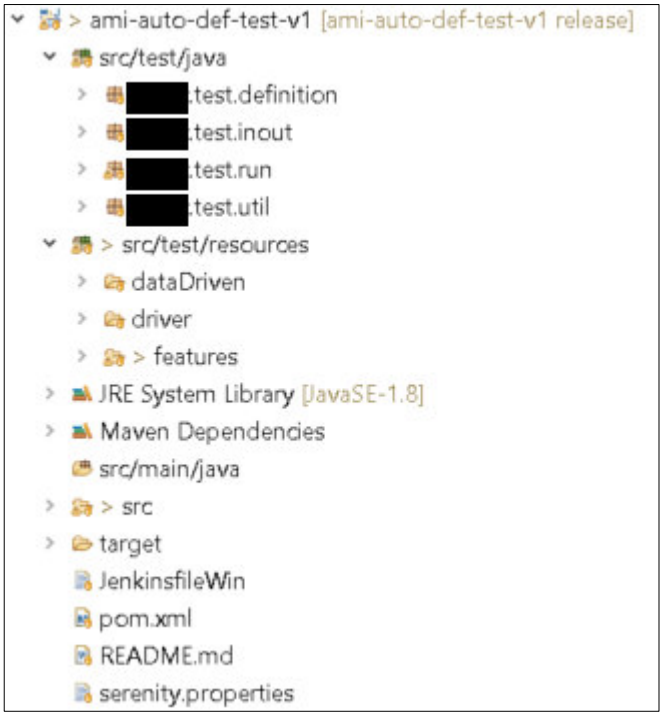
Cuando se realice la ejecución desde Jenkins los accesos serán obtenidos desde HashiCorpVault.

Raíz del proyecto:

- **Jenkinsfile:** Este archivo contiene configuraciones para la integración y ejecución de las pruebas mediante Jenkins. Se considera configuraciones para: Definición de variables de entorno, consideraciones para la construcción del proyecto, comando para ejecución parametrizadas con sentencias Cucumber, generación de reporte Serenity y su publicación.
- **Pom.xml:** Es un archivo XML de configuración donde se registran las dependencias (librerías, plugin), nombres y atributos propios del proyecto para su construcción.
- **serenity.properties:** Archivo de configuración que contiene los parámetros o condiciones para la ejecución bajo el framework. Ejemplo: Ruta del driver, tipo de codificación, ejecución en primer y segundo plano, capturas de pantalla, etc.

En la figura 16, se muestra la estructura de carpetas del proyecto ami-auto-def-test.

Figura 16: Estructura de carpetas del proyecto ami-auto-def-test



Fuente: Estructura de carpetas del proyecto de automatización

Proyecto: sys-auto-app-env

Paquete: src/main/java/

- **empresa.app.step:** Capa intermedia entre los definitions y las acciones de una aplicación web o los request y response de los servicios web. El objetivo de estas clases java es la generación de los diferentes pasos que pueden utilizarse en la ejecución de un escenario y las validaciones de los criterios de aceptación.
- **empresa.app.page:** Estas clases Java son las encargadas de definir los paso a paso requeridos y las acciones dentro de la aplicación, este paquete se usará si el proyecto a automatizar es web.
- **empresa.app.request:** Estas clases Java son las encargadas de establecer los diferentes request que contienen los servicios web, este paquete se usará si el proyecto a automatizar son APIs.
- **empresa.app.response:** Estas clases Java son las encargadas de almacenar los valores obtenidos de los reponses, este paquete se usará si el proyecto a automatizar son APIs.

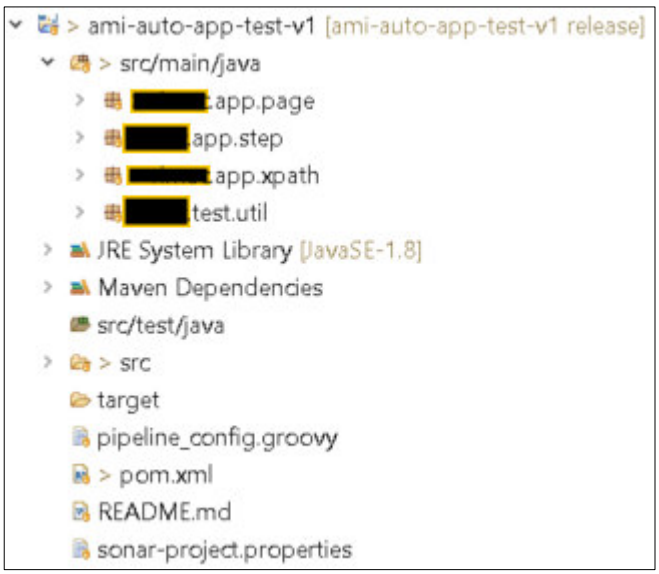
- **empresa.app.xpath:** Estas clases Java son las encargadas del mapeo de los objetos de la aplicación, este paquete se usará si el proyecto a automatizar es web.
- **empresa.test.util:** Estas clases Java contienen utilitarios propios del proyecto, donde incluye los métodos recursivos, nombre de rutas, equivalencias parametrizadas propias del proyecto.

Raíz del proyecto:

- **Pom.xml:** Es un archivo XML de configuración donde se registran las dependencias, nombres y atributos propios del proyecto para su construcción.
- **Pipeline_config:** Archivo de configuración para la publicación de librerías en Nexus.
- **Sonar-project:** Es un archivo de configuraciones para la ejecución del sonarQube mediante Jenkins.

En la figura 17, se muestra la estructura de carpetas del proyecto web.

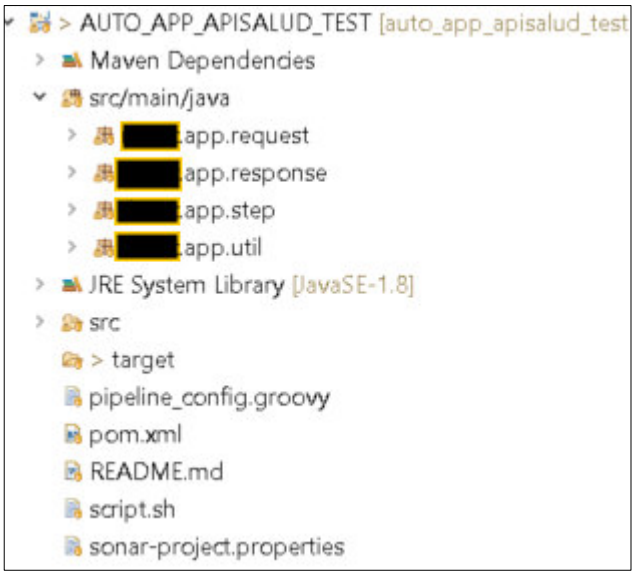
Figura 17: Estructura de carpetas del proyecto web



Fuente: Estructura de carpetas del framework de automatización

En la figura 18, se muestra la estructura de carpetas del proyecto de APIs.

Figura 18: Estructura de carpetas del proyecto de APIs



Fuente: Estructura de carpetas del framework de automatización

Proyecto: utilitarios-auto

Paquete: src/main/java/

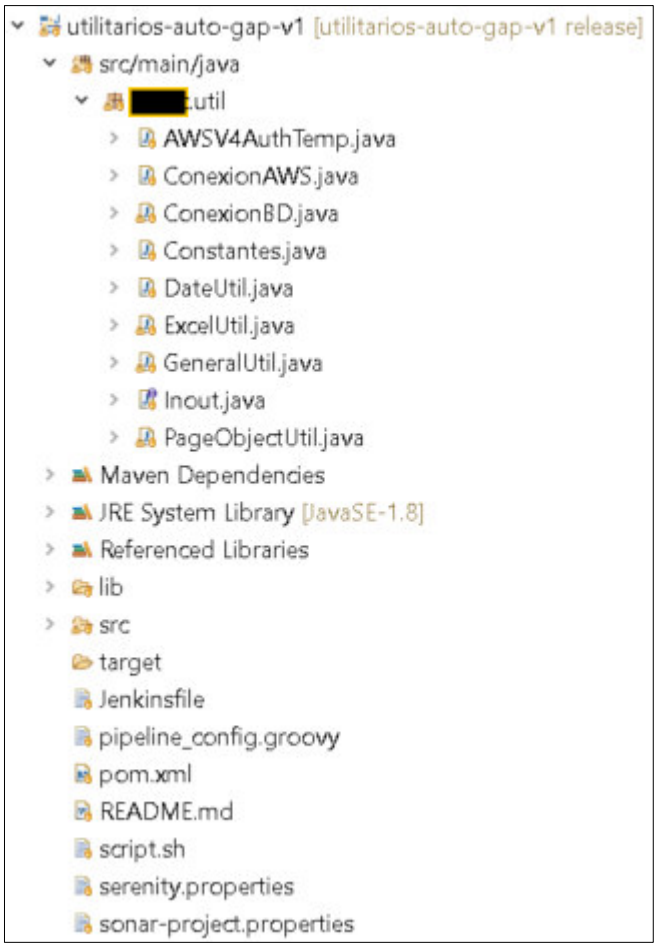
- **empresa.util:** Contiene clases Java con utilitarios que brindan soporte a nivel general para todos los proyectos de automatización de pruebas.

Raíz:

- **Pom.xml:** Es un archivo XML de configuración donde se registran las dependencias (librerías, plugin), nombres y atributos propios del proyecto para su construcción.
- **pipeline_config:** Archivo de configuración para la publicación de librerías.
- **sonar-project:** Es un archivo con configuraciones para la ejecución del SonarQube mediante Jenkins.

En la figura 19, se muestra la estructura de carpetas del proyecto utilitario.

Figura 19: Estructura de carpetas del proyecto utilitarios



Fuente: Estructura de carpetas del proyecto de automatización

Diseño de la arquitectura del framework de automatización:

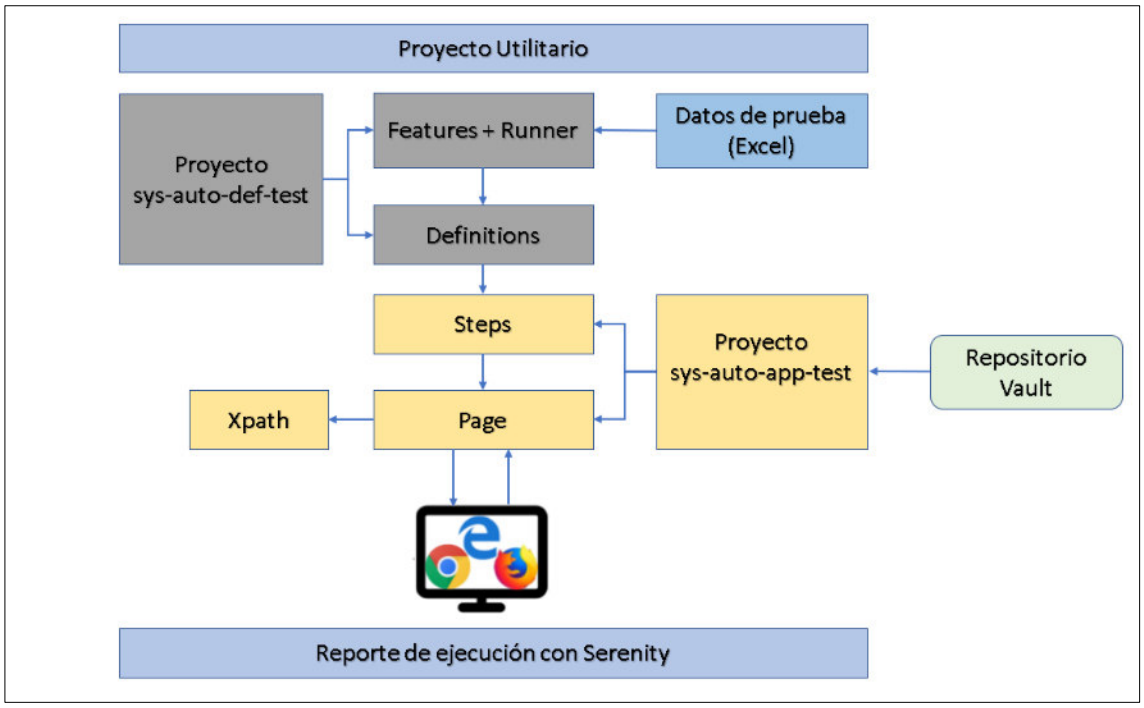
Para desarrollar este framework se utilizaron varias tecnologías que se han tenido que integrar entre sí para dar lugar a un marco de trabajo robusto y fiable, la arquitectura básicamente consta de 3 proyectos donde se organizaran las capas para que funcionen de manera óptima, donde el proyecto utilitarios-auto dedicado a dar soporte a todos los proyectos de automatización, este proyecto contiene clases con utilidades genéricas, el proyecto sys-auto-def-env contiene las clases y paquetes orientadas al negocio donde se definen los escenarios de pruebas, el proyecto sys-auto-app-env contiene las clases y paquetes orientados a la interacción con el sistema a automatizar (Web y APIS); en este proyecto se

incluirá los mapeos de los objetos, las acciones de estos y los paso a paso o steps para ser utilizados en la ejecución de los escenarios.

Al momento de realizar la integración con Jenkins el proyecto sys-auto-def-env incluirá como dependencias a los proyectos utilitarios-app y sys-auto-app-env. Se ha tenido que usar GitHub como plataforma de control de versiones en el cual se ha almacenado la totalidad del código del proyecto y Vault de donde se consumirán las credenciales para ser usados en los proyectos.

En la figura 20, se muestra la arquitectura del framework para realizar automatizaciones web.

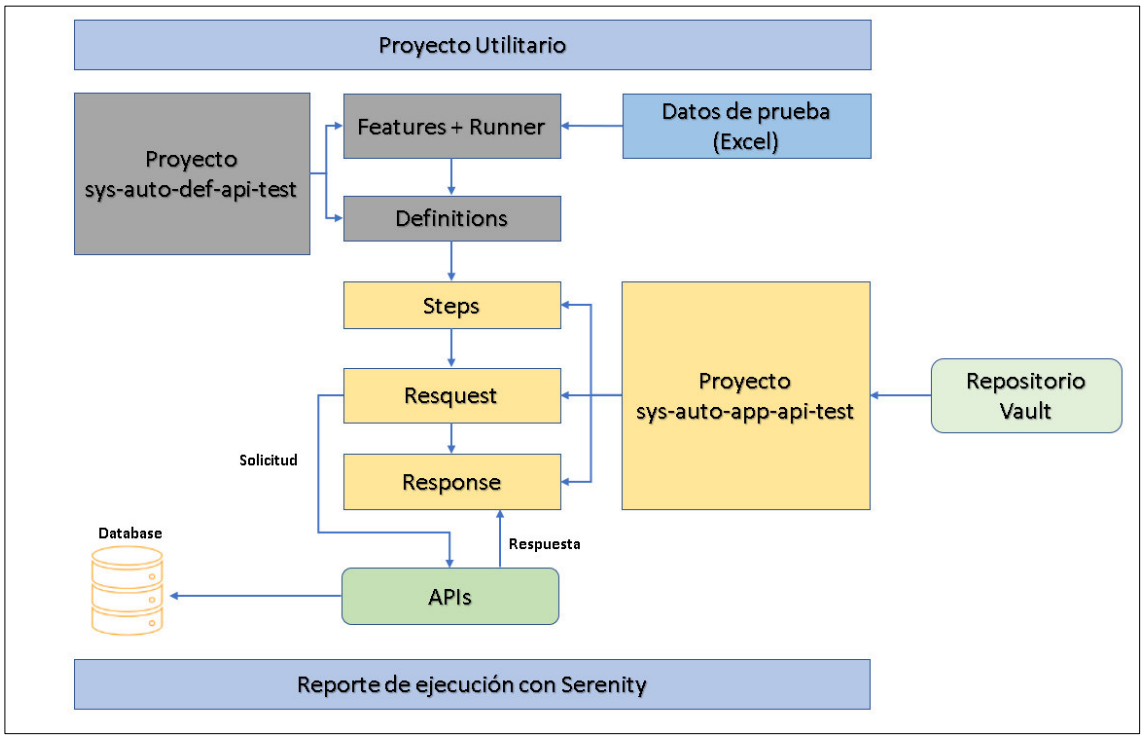
Figura 20: Arquitectura del framework de automatización web



Fuente: Elaboración del equipo de gobierno de automatización

En la figura 21, se muestra la arquitectura del framework para realizar automatizaciones de APIs.

Figura 21: Arquitectura del framework de automatización de APIs



Fuente: Elaboración del equipo de gobierno de automatización

Control de versiones con la herramienta GitHub:

Se define la estrategia para realizar el versionamiento:

- Se debe crear una rama por funcionalidad / historia de usuario.
- El nombre de la rama de desarrollo creada debe hacer referencia a la funcionalidad que se está implementando.
- Una vez que la funcionalidad funciona correctamente y se ha probado, se combina con la rama de integración (una copia de la rama principal/Master) cuya única finalidad es hacer que la rama principal nunca esté rota.
- Se debe validar que antes de realizar merge a la rama Master, se realice una prueba desde la rama de integración, para validar la convivencia de los nuevos cambios.

- Si las pruebas se realizaron satisfactoriamente, se debe realizar el push a la rama Master, caso contrario se deben realizar las correcciones en las ramas de desarrollo y volver a ser probadas en la rama de integración.
- Finalmente se procede con borrar las ramas de desarrollo de la funcionalidad / historia implementada.
- En caso de que ya se tenga el repositorio, se debe de actualizar las fuentes para que agreguen más funcionalidades a validar.
- Cada commit que se realice debe describir brevemente cada cambio que se está realizando.
- Se debe crear y configurar 2 pipelines, uno apuntando a la rama "Integration" para validar la compatibilidad con Jenkins, y otro pipeline apuntando a la rama "Master" para las pruebas de regresión, ambas configuraciones estarán en coordinación entre el equipo de automatización y el equipo de DevOps.

Configuraciones y publicación de librerías:

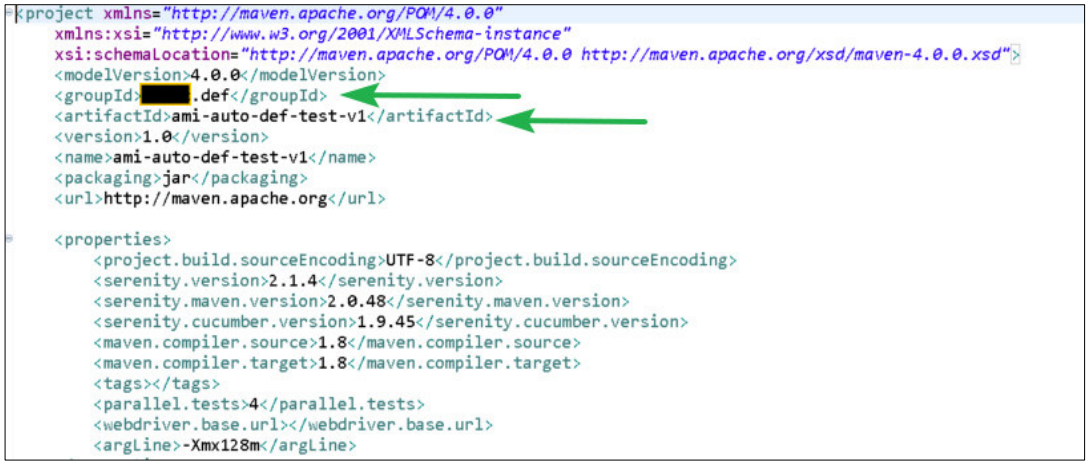
Tal como se muestra en la figura 23, existen dependencias entre los proyectos de definiciones, aplicaciones y utilitarios. Para esto, los proyectos sys-auto-app-env y utilitarios-auto serán publicados con técnicas de integración continua en el repositorio de artefactos Nexus; ambos serán compilados en librarías JAR y de este modo ser consumidas por el proyecto de definiciones y aplicaciones.

- **Configuración de archivo pom.xml para su publicación:**
    - Los proyectos deben mantener la siguiente nomenclatura al nombrar los campos correspondientes al id de grupo y al id de artefacto, como se muestra en el ejemplo el grupoId deberá ser fijo (auto.def, auto.util, auto.app), solo debe cambiar por el tipo de proyecto. Por otro lado, el campo correspondiente al artifactId deberá hacer referencia al nombre de nuestro proyecto. Esto es indispensable para realizar correctamente la publicación de las librerías.

- La información referente a la versión se deberá definir de la siguiente manera:
  - En caso se suba cambios en las ramas feature o reléase, no será necesario especificar una nueva versión ya que el versionamiento lo realizará el repositorio de artefactos Nexus.
  - En caso se suba cambios a la rama master se deberá consultar en Nexus en que versión se encuentra para poder subir los cambios incrementando la versión. Se debe tener en cuenta que la rama master no permite reemplazar sobre una versión existente.

En la figura 22, se muestra la configuración del archivo pom.xml del proyecto ami-auto-def-test-v1.

Figura 22: Configuración del groupId y artifacId



Fuente: Código fuente del framework de automatización

- El archivo pom.xml para la configuración de dependencias deberá referenciar al repositorio Nexus para que sea posible realizar las descargas. Esto no aplica para el proyecto utilitarios-auto ya que este no depende de otro proyecto (ver figura 23).

Figura 23: Configuración de la URL de Nexus

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <serenity.version>2.1.4</serenity.version>
    <serenity.maven.version>2.0.48</serenity.maven.version>
    <serenity.cucumber.version>1.9.45</serenity.cucumber.version>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <tags></tags>
    <parallel.tests>4</parallel.tests>
    <webdriver.base.url></webdriver.base.url>
    <argLine>-Xmx128m</argLine>
</properties>

<repositories>
    <repository>
        <id>[redacted]-nexus</id>
        <name>maven-public</name>
        <url>http://[redacted]/repository/maven-public</url>
    </repository>
    <repository>
```

Fuente: Código fuente del framework de automatización

- **Configuración del archivo sonar-project:**

  Los proyectos sys-auto-app-env y utilitarios-auto contienen en la raíz el archivo properties sonar-project, este archivo debe hacer referencia al nombre del proyecto para su ejecución. Tener en cuenta que todas las ramas (feature, reléase y master) están configuradas para el escaneo de análisis estáticos de código con sonarQube (ver figura 24).

Figura 24: Configuración de Sonar

```
1 sonar.projectKey=ami-auto-app-test-v1
2 sonar.projectName=ami-auto-app-test-v1
3 sonar.projectVersion=1.0.0
4 sonar.language=java
5 sonar.java.source=1.8
6 sonar.sourceEncoding=UTF-8
7 sonar.java.coveragePlugin=cobertura
8 sonar.cobertura.reportPath=target/site/cobertura/coverage.xml
9 sonar.surefire.reportsPath=target/surefire-reports
10 sonar.analysis.mode=publish
11 sonar.scm.enabled=false
12 sonar.scm-stats.enabled=false
13 sonar.jacoco.reportMissing.force.zero=False
14 sonar.sources=src/main
15 sonar.java.binaries=target/classes
16
```

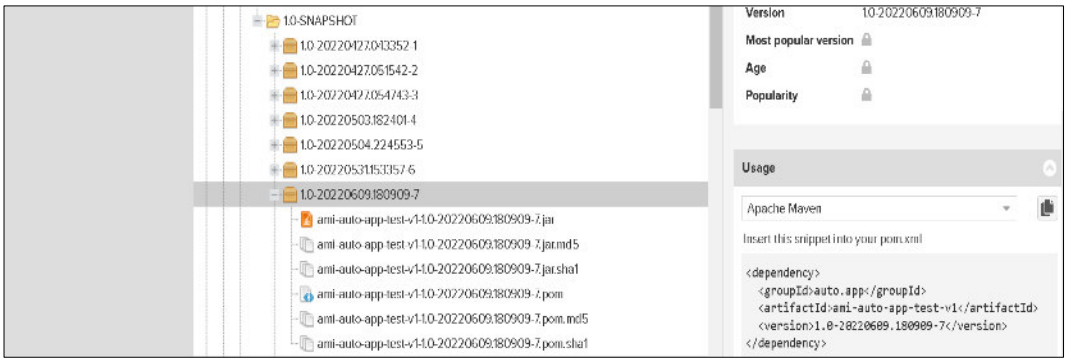
Fuente: Código fuente del framework de automatización

- **Pasos para crear un pipeline de publicación de librerías:**
    - Se tiene configurado una carpeta principal destinado a las automatizaciones de pruebas para la publicación de los Jars. El Pipeline está diseñado para ejecutarse automáticamente cada vez que se realice un cambio en la rama reléase y master.
    - La configuración de la carpeta principal no debe ser modificado.
    - Dentro de la carpeta principal se deberá crear un pipeline multibranch para poder referenciar al repositorio solo del proyecto sys-auto-app-env.
    - Una vez realizado estas configuraciones recién podemos inyectar cambios a nuestros repositorios.
    - Para el caso del proyecto utilitario-auto, por defecto ya se encontrará configurado para las nuevas publicaciones.
    - Los pipelines de publicación de librerías ya se encuentran configurados con etapas para el escaneo de la calidad del código y las vulnerabilidades para todas las ramas. En caso concluyan con hallazgos, estos deberán ser analizados para su corrección.
    - Las ramas features no están configuradas para crear versiones en el repositorio Nexus ya que estas están destinadas a probar de forma local.

- **Configuraciones del archivo pom.xml para las dependencias:**
    - Los archivos pom.xml de cada proyecto deben contener la dependencia de proyectos de acuerdo con el grafico mostrado.
        - El proyecto sys-auto-def-env deberá tener como dependencias tanto el proyecto de aplicaciones como utilitario.
        - El proyecto sys-auto-app-env deberá tener como dependencias solo el proyecto utilitario.
    - La versión de los artefactos debe ser obtenidos del repositorio Nexus de acuerdo con la publicación realizada.

En la figura 25, se muestra las librearías y versiones generadas en la herramienta Nexus.

Figura 25: Repositorio Nexus donde se publican los Jars generados



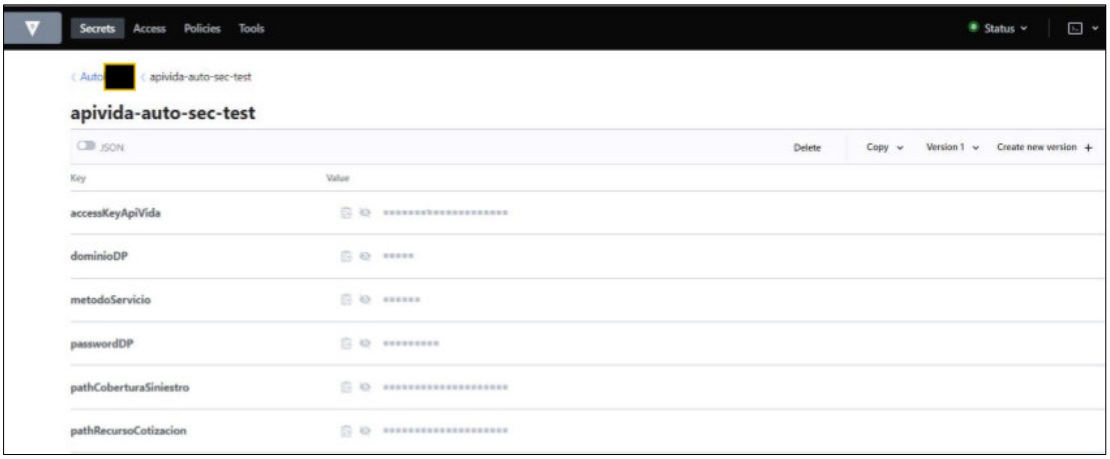
Fuente: Nexus del proyecto de automatización

Creación de secretos en Vault:

- Para la gestión de secretos se utilizará la herramienta Hashicorp Vault.
- Se deberá crear un proyecto para almacenar los secretos dentro de la carpeta “AutoEmpresa”, el proyecto a crear deberá contar con la siguiente nomenclatura sys-auto-sec-env. Ejemplo: apisalud-auto-sec-test.
- Los secretos para crear o añadir deben ser de acuerdo con el contenido del archivo secret.properties.

En la figura 26, se muestra la creación y configuración de Vault

Figura 26: Configuración de las credenciales en Vault



Fuente: Servidor Vault del proyecto de automatización

- En el archivo Jenkinsfile del proyecto auto-def se debe considerar:
  - La declaración de las variables de entorno por cada secreto registrado en Vault.
  - La ruta donde se encuentran almacenados los secretos.

En la figura 27, se muestra la configuración del archivo Jenkinsfile para consumir los secretos desde Vault.

Figura 27: Configuración del Jenkinsfile con Vault

```
def secrets = [
  [path: 'Auto[redacted]/apivida-auto-sec-test', engineVersion: 2, secretValues: [
    [envVar: 'varFlag', vaultKey: 'varFlag'],
    [envVar: 'accessKeyApiVida', vaultKey: 'accessKeyApiVida'],
    [envVar: 'dominioDP', vaultKey: 'dominioDP'],
    [envVar: 'passwordDP', vaultKey: 'passwordDP'],
    [envVar: 'pathRegistroSiniestro', vaultKey: 'pathRegistroSiniestro'],
    [envVar: 'pathCoberturaSiniestro', vaultKey: 'pathCoberturaSiniestro'],
    [envVar: 'pathRecursoCotizacion', vaultKey: 'pathRecursoCotizacion'],
```

Fuente: Código fuente del framework de automatización

- **Creación de pipeline para la ejecución de pruebas:**
    - Se debe contar con un pipeline para la ejecución de pruebas, este pipeline solo debe hacer referencia al proyecto de definiciones sys-auto-def-env para su ejecución.

3.2.5.2 Relevamiento de información:

Se realizó dos reuniones con el Product Owner, Scrum Master, Analistas de pruebas funcionales, donde nos realizaron la presentación de sus procesos macro de las aplicaciones, la cual nos permitió evaluar y definir que se realizara la automatización de escenarios de pruebas Web y APIs.

3.2.5.3 Gestión de información:

Se realizó la validación del ambiente de pruebas, donde se verificó que el ambiente es una copia idéntica del ambiente de producción y la definición del alcance y la estrategia de automatización siguiendo los lineamientos definidos en el gobierno de automatización y priorizando el backlog del Sprint, siendo las actividades para realizar las definidas en cada Sprint.

El alcance será la automatización de 3 historias de usuario:

Web:

- HU1: Emitir póliza Seguro de Salud
  Donde se definió 5 escenarios y 22 casos de pruebas.
- HU2: Emitir póliza Seguro de Salud TYP
  Donde se definió 6 escenarios y 48 casos de pruebas.

APIs:

- HU3: Validar la cotización del producto AMI
  Donde se definió 8 escenarios y 200 casos de pruebas.

3.2.5.4 Estimación de esfuerzos:

Luego de tener definido el alcance de los escenarios de pruebas a automatizar se realizó la estimación siendo el tiempo de seis meses que se dividirán en 13 Sprints para realizar la implementación del framework.

3.2.5.5 Diseño de casos de pruebas:

Se realizó el diseño de los escenarios de pruebas aplicando el enfoque de Desarrollo guiado por el comportamiento (BDD) usando el lenguaje Gherkin.

Historias de usuarios que se automatizaran para la aplicación Web:

Historia de usuario: Emitir póliza Seguro de Salud

Como usuario

Quiero comprar una póliza de seguro de salud de forma digital

Para validar la compra exitosa del seguro

Donde los escenarios y casos de pruebas de la historia de usuario son los siguientes:

Tabla 9: Escenarios de pruebas de la historia emitir póliza seguro de salud

#	Escenarios	Casos
1	Emisión Seguro de Salud con tercero registrado sin familiares.	8
2	Emisión Seguro de Salud con tercero registrado con familiares existentes.	8
3	Emisión Seguro de Salud con tercero nuevo sin familiares.	8
4	Emisión Seguro de Salud con tercero existente con familiares nuevos.	8
5	Emisión Seguro de Salud con tercero nuevo con familiares registrados.	8
6	Emisión Seguro de Salud con tercero nuevo con familiares nuevos.	8

Fuente: Elaboración propia

Historia de usuario: Emitir póliza Seguro de Salud TYP

Como usuario

Quiero comprar una póliza de seguro de salud de forma digital

Para validar mensaje de derivación con un asesor

Donde los escenarios y casos de pruebas se muestran en la tabla 10.

Tabla 10: Escenarios de pruebas de la historia emitir póliza seguro de salud TPY

#	Escenarios	Casos
1	Emisión Seguro de Salud TYP cuando el tercero tiene una póliza activa	4
2	Emisión Seguro de Salud TYP cuando tercero es menor de edad	4
3	Emisión Seguro de Salud TYP cuando tercero es mayor de 60 años	4
4	Emisión Seguro de Salud TYP cuando tercero tiene siniestros	4
5	Emisión Seguro de Salud TYP cuando tercero marca con la opción “si” a las preguntas de la declaración jurada	6

Fuente: Elaboración propia

Previo a la implementación se deben diseñar estos escenarios de pruebas en lenguaje Gherkin siguiendo las buenas prácticas definidas en el punto 3.2.3 (Ver anexo 1 A).

Para este caso práctico se realizará el flujo de desarrollo del escenario “Emisión Seguro de Salud con tercero registrado sin familiares”.

Scenario Outline: 1- Emisión Seguro de Salud con tercero registrado sin familiares

Given ingresar a la web seguro de salud

And registrar datos del usuario "<tipoDocumento>", "<nroDocumento>", "<celular>"

When seleccionar un plan "<plan>"

And registrar datos personales "<correo>"

And seleccionar con no todas las preguntas

And registrar la dirección "<dpto>", "<provincia>", "<distrito>", "<dirección>"

And realizar el pago "<nroTarjeta>", "<fVen>", "<ccv>", "<nombre>", "<apellido>", "<correo>"

Then validar emisión del seguro de forma satisfactoria

Ahora veremos la historia de usuario que se automatizaran para las APIs:

Historia de usuario: Validar la cotización del producto AMI

Como usuario.

Quiero realizar cotizaciones del producto.

Para validar la prima neta y generación de la cotización de forma correcta.

Donde los escenarios y casos de pruebas son los siguientes:

Tabla 11: Escenarios de pruebas de la historia Validar la cotización del producto AMI

#	Escenarios	Casos
1	Validar la cotización del producto.	25
2	Validar el servicio de seleccionar cotización.	25
3	Agregar Terceros a cotización.	25
4	Agregar Ruta Anexo a cotización.	25
5	Agregar suscripción a cotización.	25
6	Enviar emisión sin pago cotización.	25
7	Enviar emisión con suscripción cuenta bancaria de cotización.	25
8	Obtener datos de cotización.	25

Fuente: Elaboración propia

Previo a la implementación se deben diseñar estos escenarios de pruebas en lenguaje Gherkin siguiendo las buenas prácticas definidas en punto 3.2.3 (Ver anexo 2 A).

Para este caso práctico se realizará el flujo de desarrollo del escenario “Validar la cotización del producto”.

Scenario Outline: 1- Validar la cotización del producto

Given accedo al servicio de cotización
When ingreso los datos correspondientes a moneda "<idpMoneda>", codigo de canal "<ideCanal>", inicio de vigencia "<fecIniVig>" y fecha fin de vigencia "<fecFinVig>"
And ingreso los datos del plan de financiamiento tipo de facturacion "<tipFacturacion>", periodo "<periodo>", comision "<comision>"

And ingreso el detalle del financiamiento como el plan "<idePlanFinanciamiento>", numero de cuota "<numCuota>", fecha de inicio "<fecIni>"
And ingreso los datos de los asegurados como tipo de documento "<tipDocumento>", numero de documento "<numDocumento>", parentesco "<parentesco>", sexo "<sexo>", fecha de nacimiento "<fecNacimiento>“, nombre "<nombreAsegurado>",apePaterno "<apePaternoAsegurado>",apeMaterno "<apeMaternoAsegurado>"
And ingreso los descuentos como tipo "<tipo>", indicador "<indTipo>", frencuencia "<codFrecuencia>", clasificacion "<codClasificacion>", porcentaje "<porcDescuento>", fecha inicio "<fechIniDesc>", fecha fin "<fechFinDesc>"
Then se muestra la prima neta y el número de cotizacion del produco AMI

3.2.5.6 Implementación, versionamiento e integración continua:

Ahora detallaremos las herramientas utilizadas para configurar el ambiente de trabajo.

Herramientas comunes

- Java Open JDK v8 u221
- IDE Eclipse
- Maven 3.6.*
- Git 2.26.2
- GitHub
- Cucumber

Para las aplicaciones web

- Serenity versión 2.08.81
- Selenium Web Driver

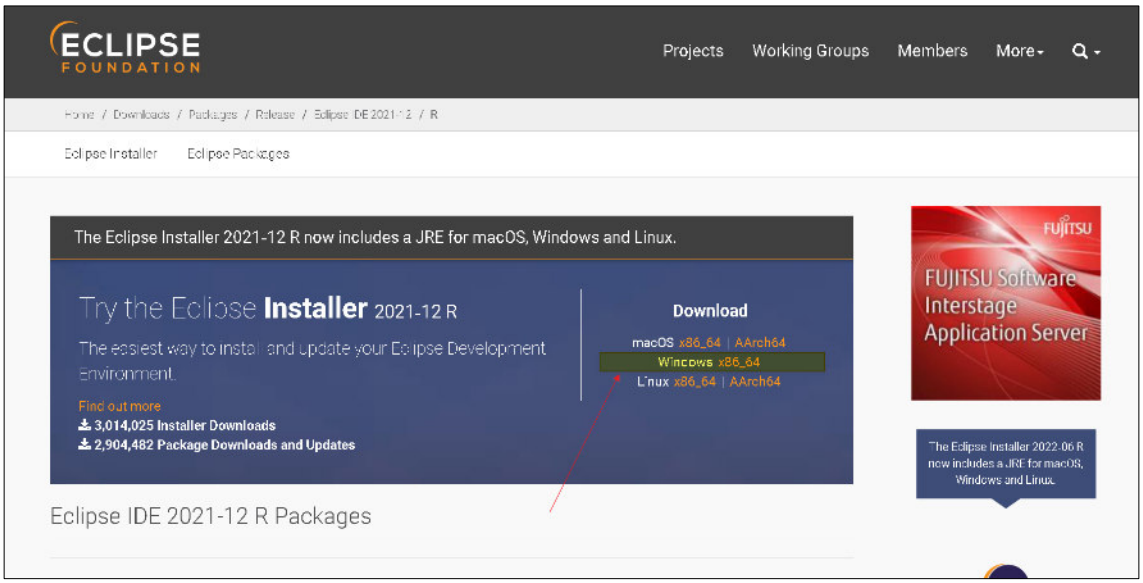
Para las APIs

- Rest Assured 3.0.2
- Serenity Rest 2.0.81

Configuración del ambiente de trabajo:

- Realizar la descarga de la herramienta eclipse y descomprimir (ver figura 28).

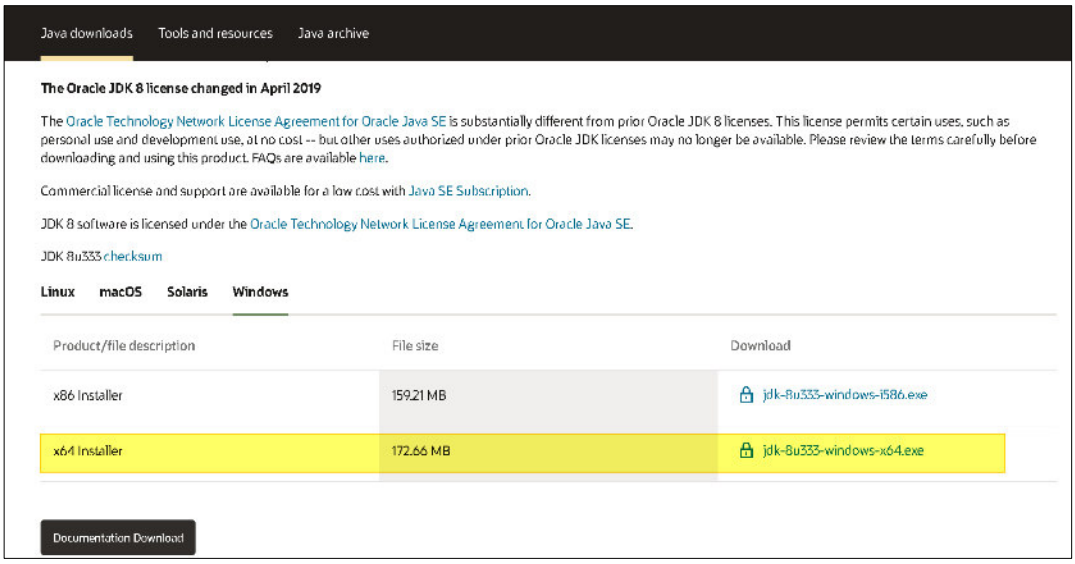
Figura 28: Sitio web oficial de descarga de Eclipse



Fuente: https://www.eclipse.org/downloads/

- Descargar e instalar Java JDK Java Open JDK v8 u221de la página oficial de Oracle (ver figura 29).

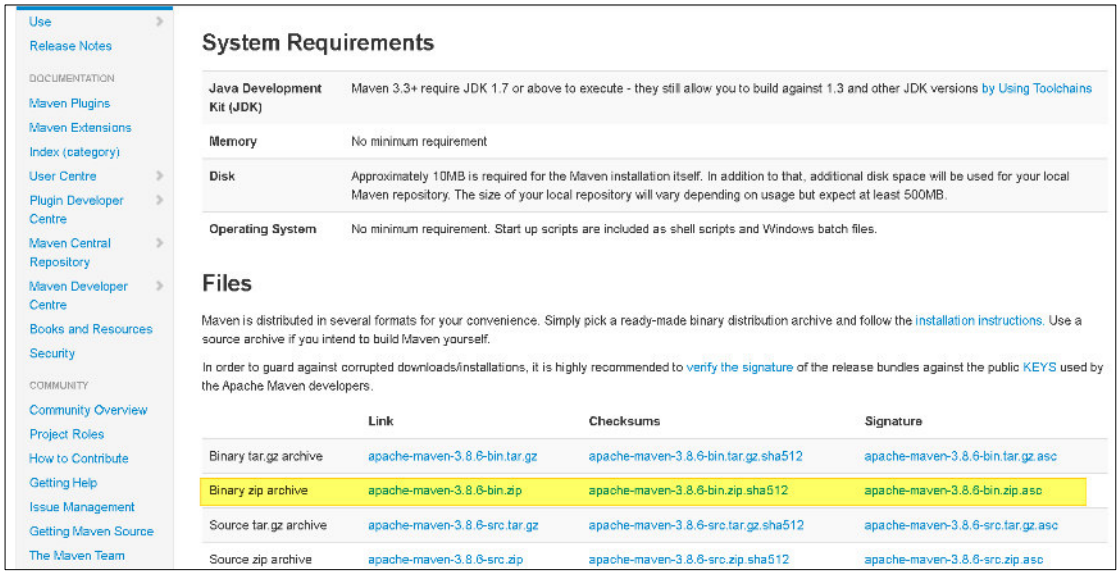
Figura 29: Sitio web oficial de descarga del JDK



Fuente: https://www.oracle.com/java/technologies/downloads/

- Realizar la descarga de Maven, descomprimir y configurar en las variables de entorno (ver figura 30).

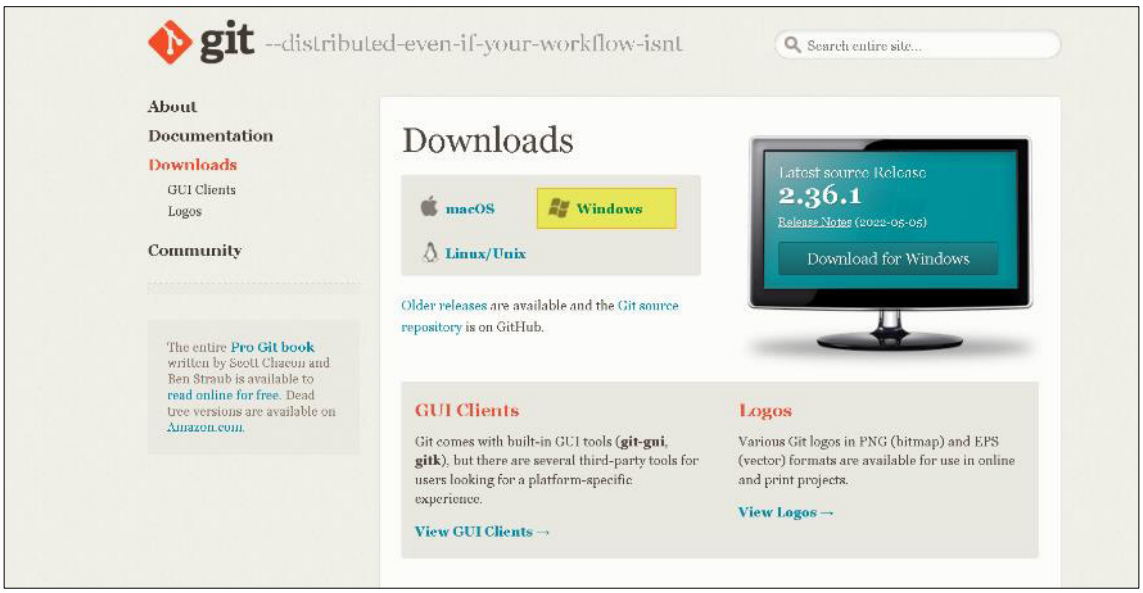
Figura 30: Sitio web oficial de descarga de Maven



Fuente: https://maven.apache.org/download.cgi

- Realizar la descarga de la herramienta GIT e instalar (ver figura 31).

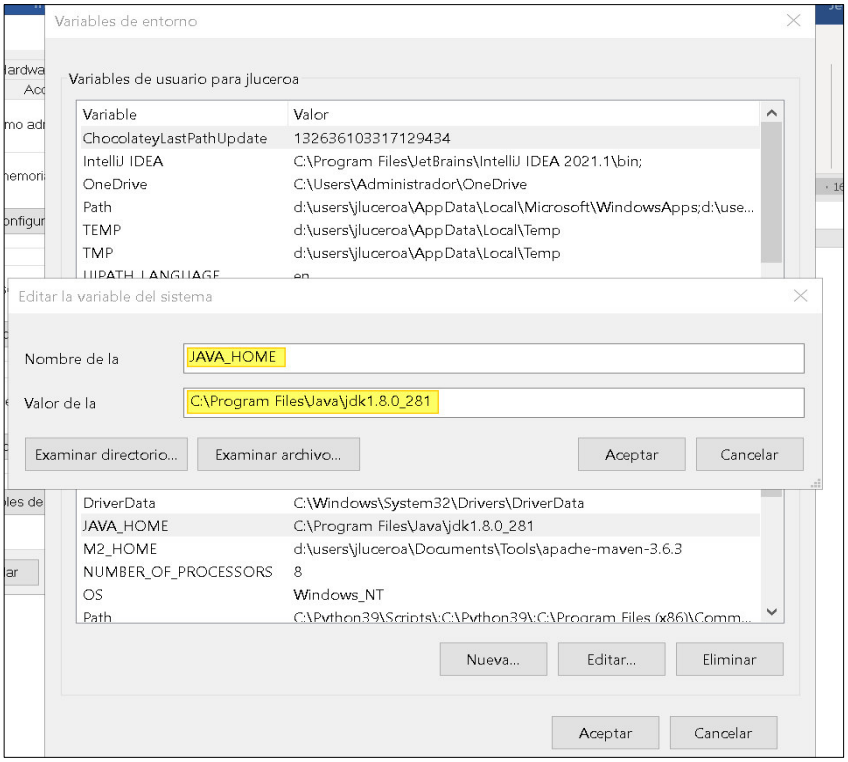
Figura 31: Sitio web oficial de descarga de Git



Fuente: https://git-scm.com/downloads

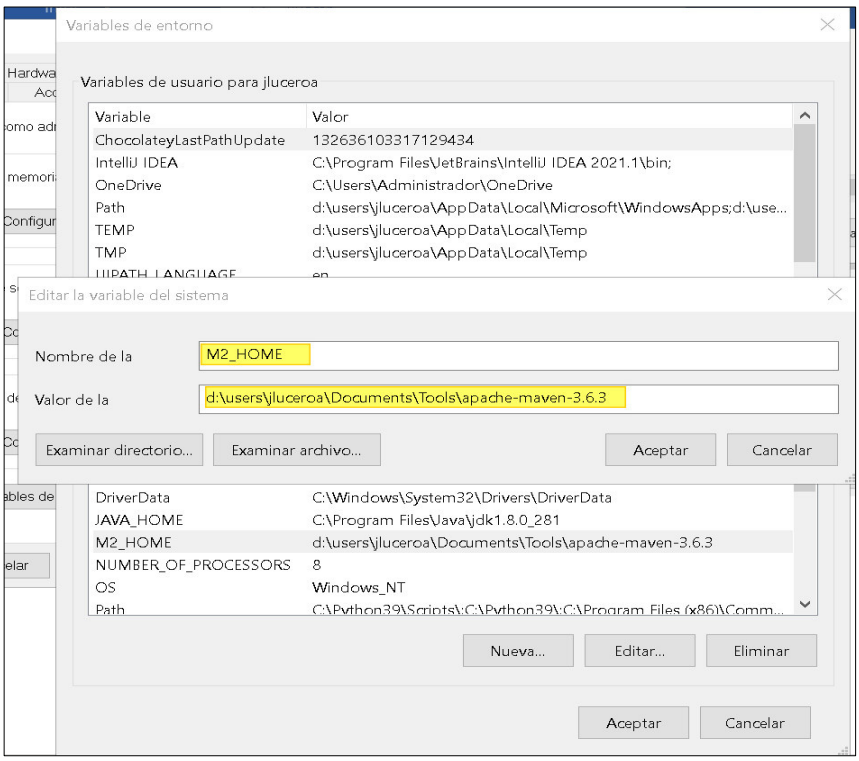
- Crear y agregar las variables de Java y Maven en las variables de entorno del sistema, abrir el panel de control e ingresar a la ventana de variable de entorno del sistema y crear las variables JAVA_HOME y M2_MAVEN e indicar la ruta donde se encuentran (ver figura 32 y 33).

Figura 32: Configuración de la variable de entorno Java



Fuente: Elaboración propia

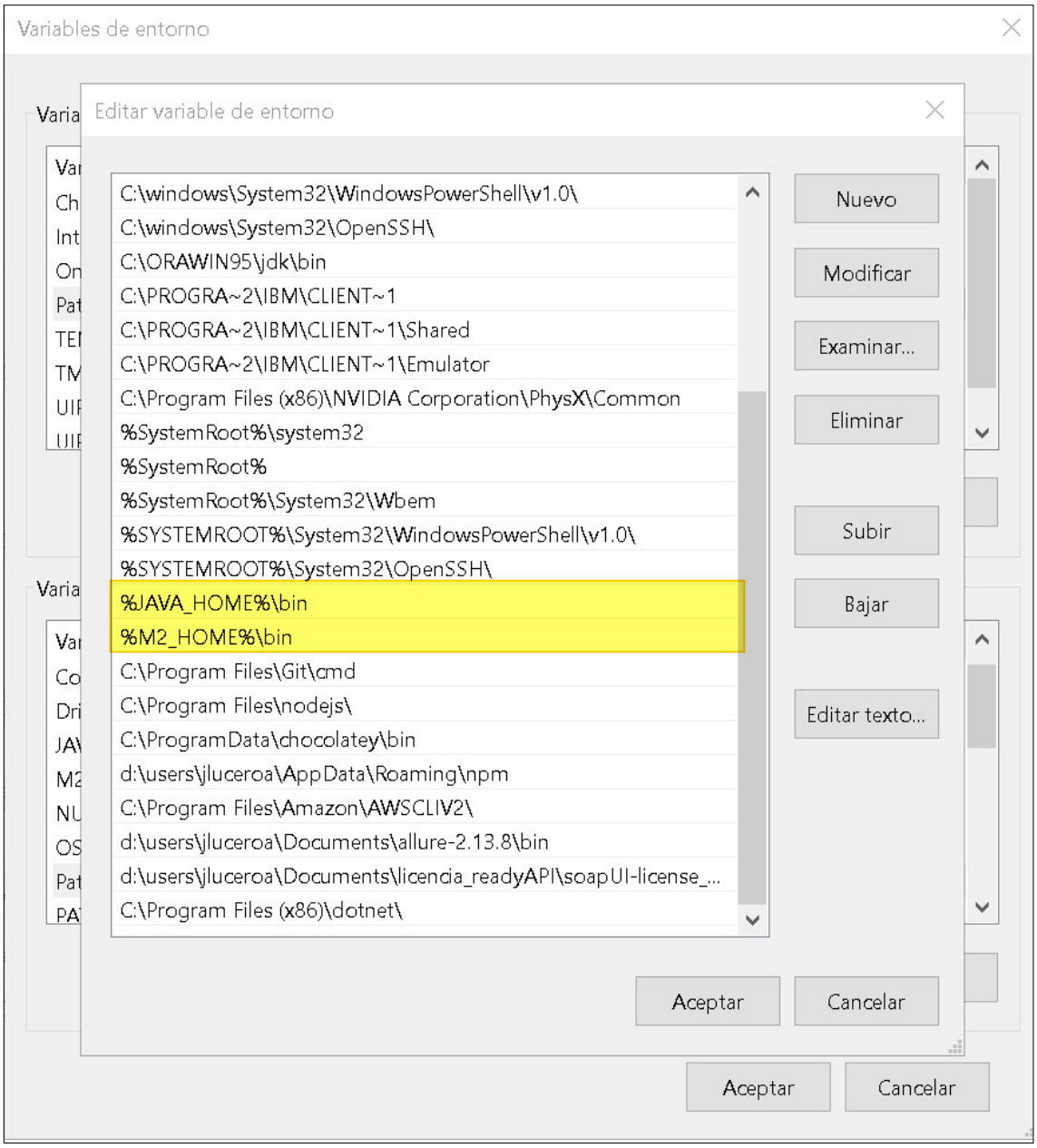
Figura 33: Configuración de la variable de entorno Maven



Fuente: Elaboración propia

- Luego de a ver creado las variables JAVA_HOME y M2_HOME, seleccionar la variable de sistema path para editar e ingresar estas variables creadas (ver figura 34).

Figura 34: Configuración de las variables de entorno en el path del sistema

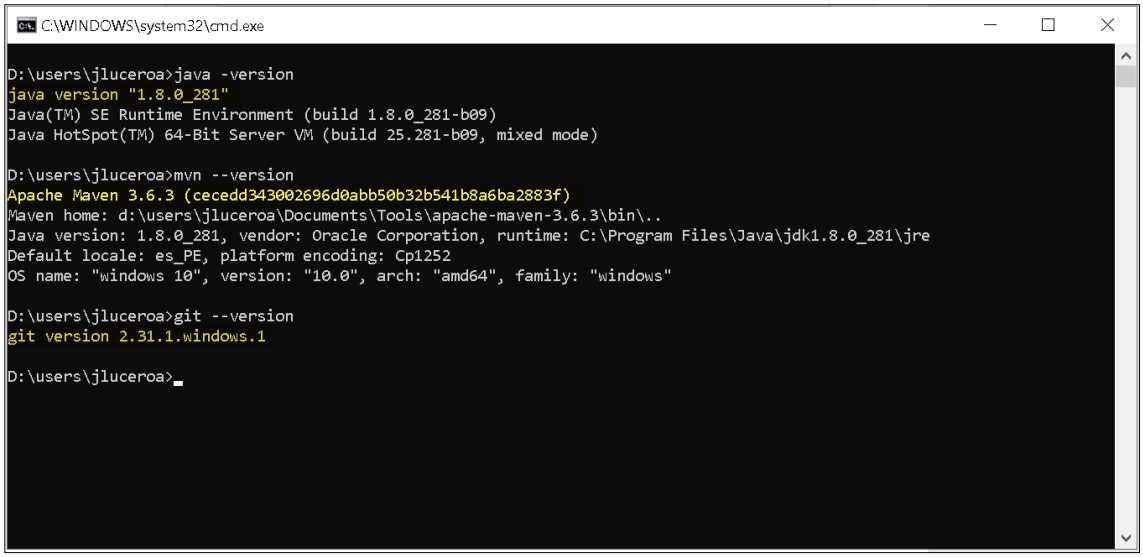


Fuente: Elaboración propia

- Validar las configuraciones e instalaciones realizadas con los siguientes comandos (ver figura 35).

    - java --v
    - mvn --v
    - git –version

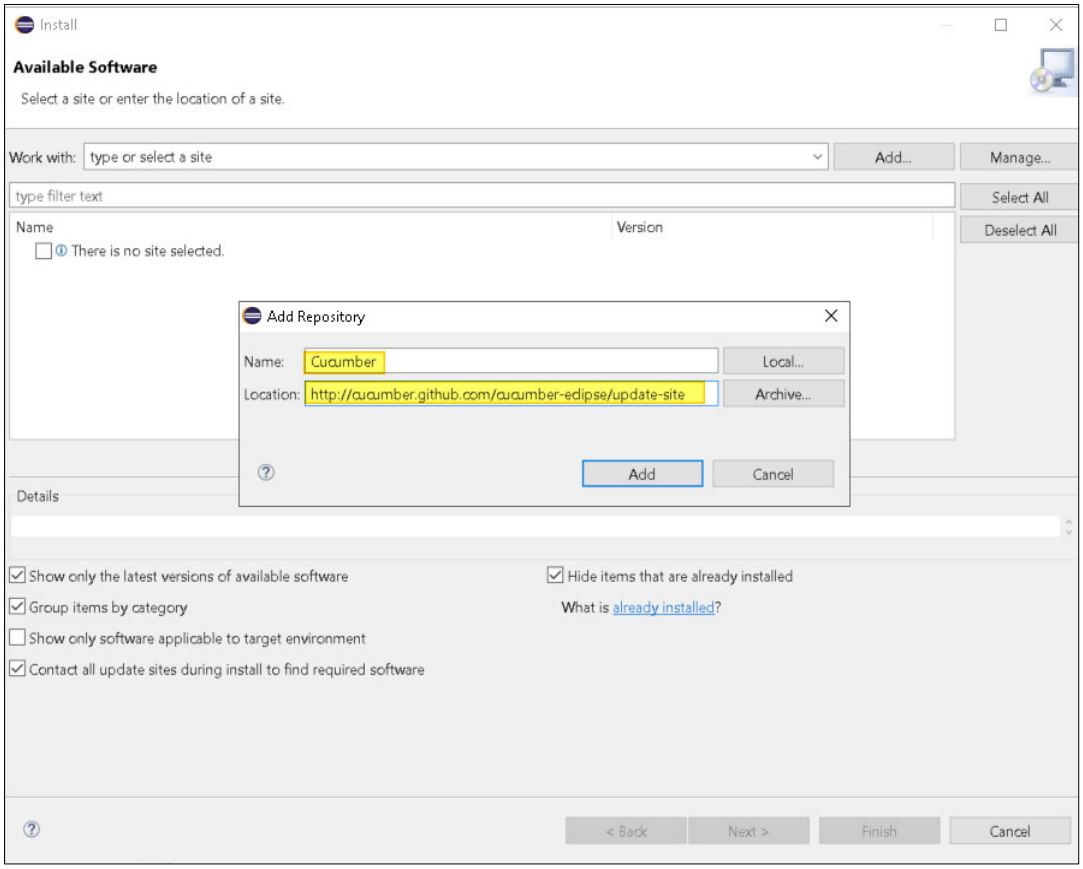
Figura 35: Validación de las configuraciones en la consola de Windows



Fuente: Elaboración propia

- Crear el espacio de trabajo, iniciar Eclipse e instalar el plugin de Cucumber para poder usar Gherkin (ver figura 36).

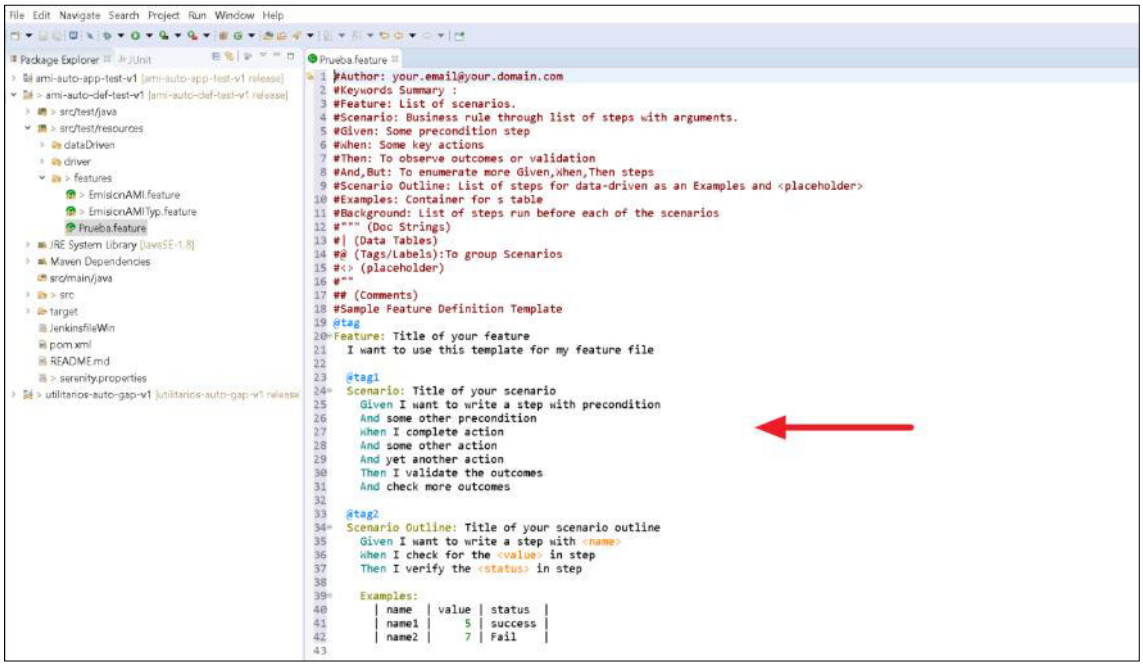
Figura 36: Instalación de plugin de Cucumber en Eclipse



Fuente: Elaboración propia

- Crear el proyecto para empezar a realizar la implementación de las pruebas automatizadas y validar que el plugin de Cucumber se haya instalado de forma correcta. Se observa en la siguiente figura que al crear un archivo tipo feature por defecto contiene un ejemplo de historia de usuario con las características de Gherkin (ver figura 37).

Figura 37: Validación de la instalación del plugin de Cucumber



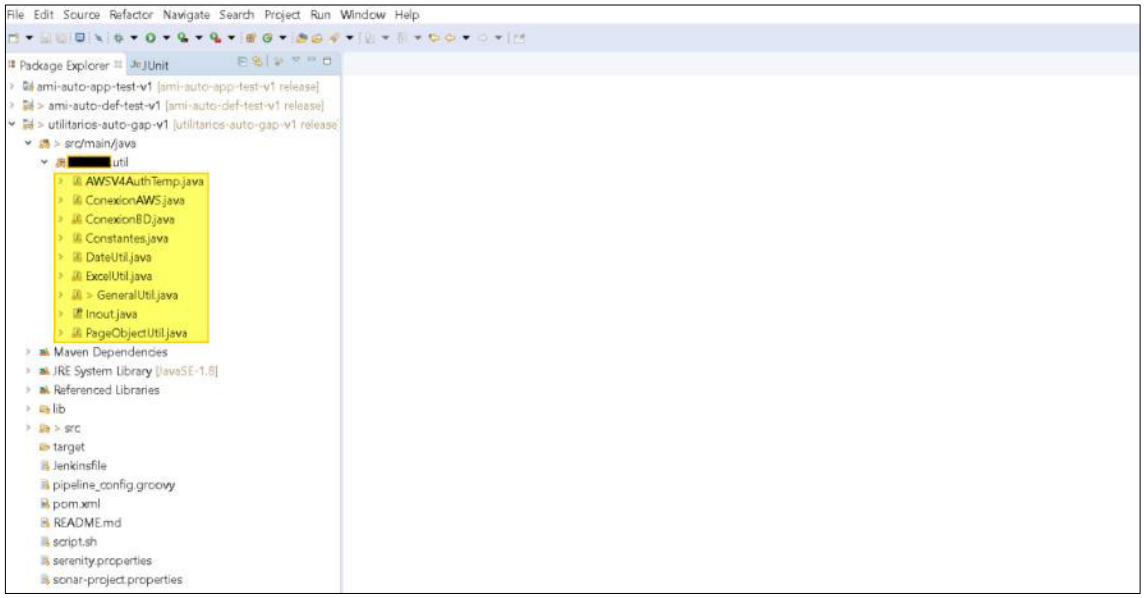
Fuente: Código fuente del framework de automatización

Creación del proyecto utilitario_auto:

En este proyecto se implementará los métodos genéricos que van a ser utilizados por los proyectos sys-auto-def-env y sys-auto-app-env, tanto para proyectos Web y APIs, donde solo se podrá adicionar nuevos métodos y no modificar los ya existentes porque estos métodos serán usados por todos los proyectos de automatización. Este proyecto utilitario-auto fue implementado en conjunto con el equipo de automatización de pruebas.

En la figura 38, se muestra la estructura de carpetas del proyecto con las clases creadas.

Figura 38: Estructura y clases creadas para su implementación del proyecto utilitario



Fuente: Estructura de carpetas del framework de automatización

Donde las clases:

- **ConexionAWS:** Contiene los métodos y las conexiones hacia los servicios de AWS que serán usados en el proyecto de automatización de APIs (ver figura 39).

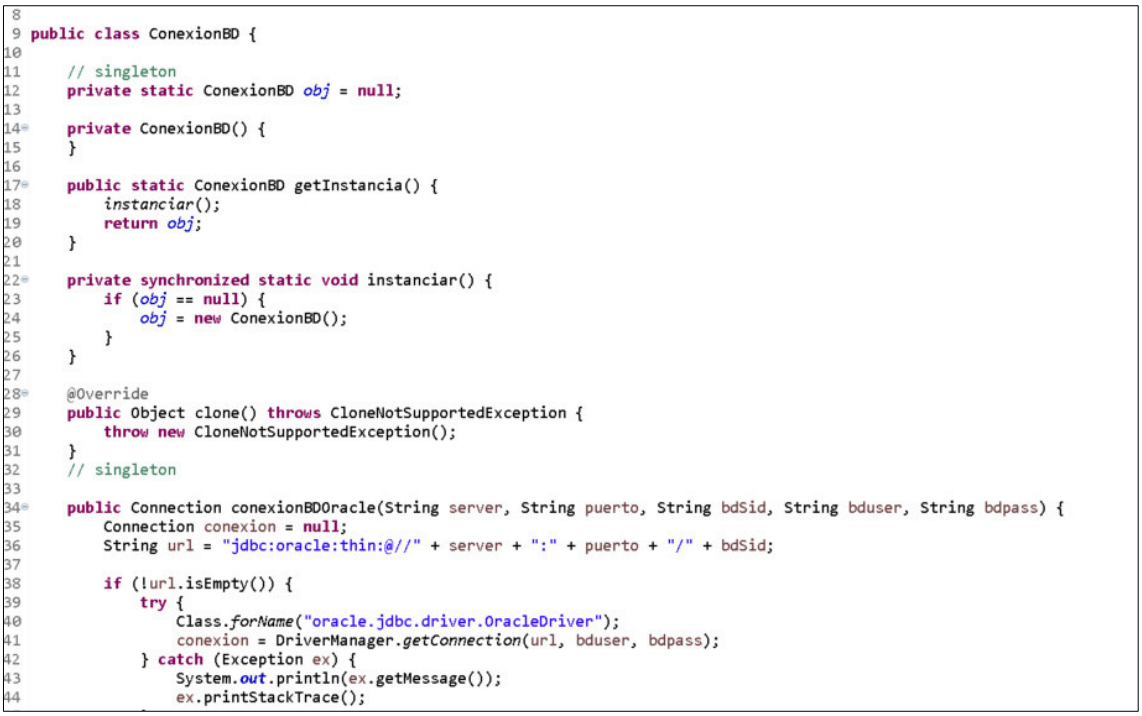
Figura 39: Clase que contiene las conexiones a los servicios AWS

```
9  public class ConexionAWS {
10
11     public Response conexion(String protocolo,String region,String metodo,String host, String urlPath, String apiKey, String secretKey, String json
12         System.out.println(protocolo + " "+ region+" "+ metodo+" "+host+" "+urlPath+" "+apiKey+" "+secretKey+" "+jsonRequest+ " "+ restAssuredReque
13
14         TreeMap<String, String> awsHeaders = new TreeMap<String, String>();
15          awsHeaders.put("host", host);
16          awsHeaders.put("content-type", "[redacted]");
17          awsHeaders.put("x-amz-target", "[redacted]");
18          awsHeaders.put("content-encoding", "[redacted]");
19
20         AWSV4AuthTemp awsV4Auth = new AWSV4AuthTemp.Builder(apiKey, secretKey)
21                 .region(region)//
22                 .service("execute-api")// es - elastic search. use your service name
23                 .httpMethodName(metodo)// GET, PUT, POST, DELETE, etc...
24                 .path(urlPath)// end point
25                 // .queryParametes(null)// query parameters if any
26                 .headers(awsHeaders)// aws header parameters
27                 .body(jsonRequest)// payload if any
28 //              .debug()// turn on the debug mode
29                 .build();
30
31         Map<String, String> header = awsV4Auth.getHeaders();
32         for (Map.Entry<String, String> entrySet : header.entrySet()) {
33             String key = entrySet.getKey();
34             String value = entrySet.getValue();
35             restAssuredRequest.header(key, value);
36         }
```

Fuente: Código fuente del framework de automatización

- **ConexionDB:** Contiene los métodos y las conexiones a distintas bases de datos y se hace uso del patrón Singleton (ver figura 40).

Figura 40: Clase que contiene la conexiones a las bases de datos



```
8
9  public class ConexionBD {
10
11     // singleton
12     private static ConexionBD obj = null;
13
14     private ConexionBD() {
15     }
16
17     public static ConexionBD getInstancia() {
18         instanciar();
19         return obj;
20     }
21
22     private synchronized static void instanciar() {
23         if (obj == null) {
24             obj = new ConexionBD();
25         }
26     }
27
28     @Override
29     public Object clone() throws CloneNotSupportedException {
30         throw new CloneNotSupportedException();
31     }
32     // singleton
33
34     public Connection conexionBDOracle(String server, String puerto, String bdSid, String bduser, String bdpass) {
35         Connection conexion = null;
36         String url = "jdbc:oracle:thin:@//" + server + ":" + puerto + "/" + bdSid;
37
38         if (!url.isEmpty()) {
39             try {
40                 Class.forName("oracle.jdbc.driver.OracleDriver");
41                 conexion = DriverManager.getConnection(url, bduser, bdpass);
42             } catch (Exception ex) {
43                 System.out.println(ex.getMessage());
44                 ex.printStackTrace();
```

Fuente: Código fuente del framework de automatización

- **ExcelUtil:** Contiene los métodos de lectura y escritura del archivo Excel hacia los features del proyecto (ver figura 41).

Figura 41: Clase que contiene los métodos de lectura y escritura hacia los features

```
3 import java.io.BufferedReader;
34
35 public class ExcelUtil {
36
37 //    static XSSFWorkbook wrkbook;
38 //    static XSSFSheet wrksheet;
39     static Hashtable dict = new Hashtable();
40     static Hashtable dictData = new Hashtable();
41     static Hashtable dictDataRow = new Hashtable();
42     static int columna = 0;
43     static String archivo = "";
44     static Workbook workbook;
45     static Sheet sheet;
46
47     public void cerrarIn(FileInputStream fileInputStream) throws Exception {
48         if (fileInputStream != null) {
49             fileInputStream.close();
50         }
51     }
52
53     public void cerrarWb(Workbook workbook) throws Exception {
54         if (workbook != null) {
55             workbook.close();
56         }
57     }
58
59     public void cerrarOut(FileOutputStream fileOutputStream) throws Exception {
60         if (fileOutputStream != null) {
61             fileOutputStream.close();
62         }
63     }
64
65     public void cerrarWb2() throws Exception {
66
67             workbook.close();
68
69     }
```

Fuente: Código fuente del framework de automatización

- **GeneralUtil:** Contiene métodos genéricos que pueden ser usados en los distintos proyectos y se hace uso del patrón Singleton (ver figura 42).

Figura 42: Clase que contiene los métodos genéricos que serán consumidos por distintos proyectos

```
27 public class GeneralUtil {
28
29     private ConexionBD conexionBD = ConexionBD.getInstancia();
30
31     // singleton
32     private static GeneralUtil obj = null;
33
34     private GeneralUtil() {
35     }
36
37     public static GeneralUtil getInstancia() {
38         instanciar();
39         return obj;
40     }
41
42     private synchronized static void instanciar() {
43         if (obj == null) {
44             obj = new GeneralUtil();
45         }
46     }
47
48     @Override
49     public Object clone() throws CloneNotSupportedException {
50         throw new CloneNotSupportedException();
51     }
52     // singleton
53
54     public String fechaAgregarTiempo(String fecha, String formato, int cale, int cant) {
55         try {
56             SimpleDateFormat simpleDateFormat = new SimpleDateFormat(formato);
57             Date date = simpleDateFormat.parse(fecha);
58             Calendar calendar = Calendar.getInstance();
```

Fuente: Código fuente del framework de automatización

- **Inout:** Contiene métodos de escritura y lectura hacia los Excel luego de la ejecución de los escenarios (ver figura 43).

Figura 43: Clase que contiene los métodos de escritura de los resultados de la ejecución

```
3  import java.util.List;
4
5  public interface Inout {
6
7      List<List<String>> leerDD(String hoja) throws Exception;
8
9      void escribirDD(List<String> listaString, String id) throws Exception;
10
11     void escribirDD(String string, String id) throws Exception;
12
13     void escribirNuevoDD(List<String> listaString, int dataDriven) throws Exception;
14
15     void escribirNuevoDD(String string, int dataDriven) throws Exception;
16 }
17
```

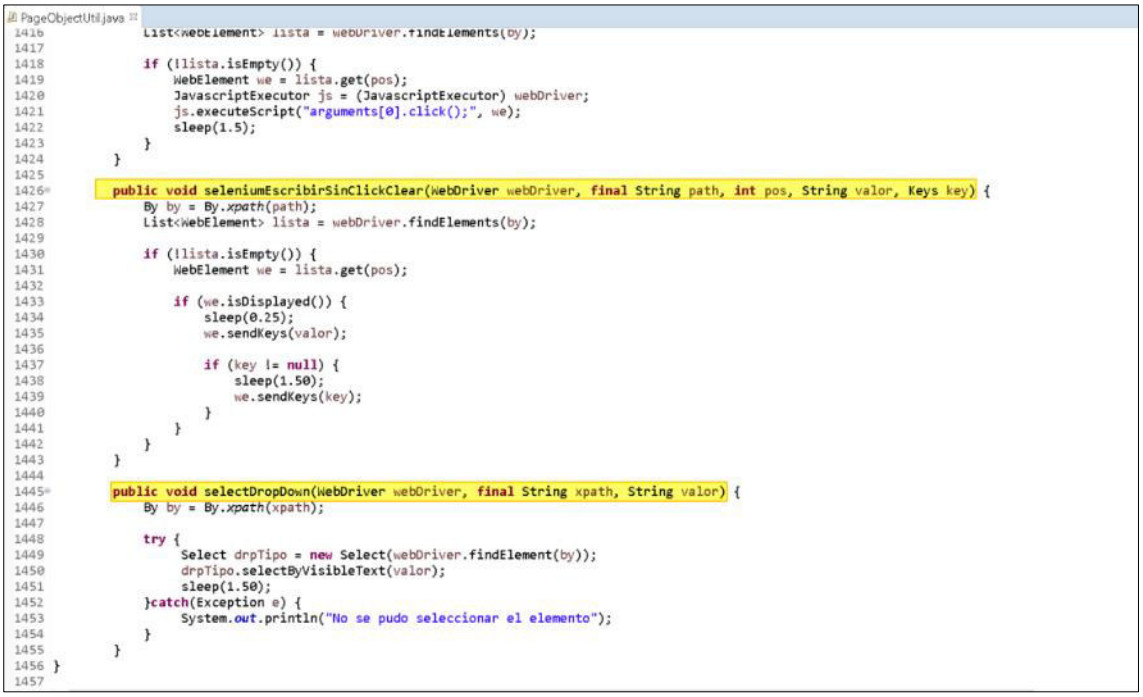
Fuente: Código fuente del framework de automatización

- **PageObjectUtil:** Contiene los métodos para realizar las interacciones con las aplicaciones web (ejemplo: dar clic, escribir en un campo, seleccionar un combo, verificar que un texto este visible en la web, etc..).

  Como se muestra en la siguiente figura 45, estos métodos serán usados al momento de realizar las acciones al momento de automatizar aplicaciones web, donde todos los métodos creados deben enviar como parámetro la instancia de Selenium, el localizador y posición del elemento.

  Como se observa en la figura, los métodos selectDropDown y seleniumEscribirSinClickClear reciben como parámetros la instancia de Selenium, el localizador del elemento, la posición del elemento y el valor que se desea escribir y seleccionar (ver figura 44).

Figura 44: Métodos de interacciones con la aplicación web

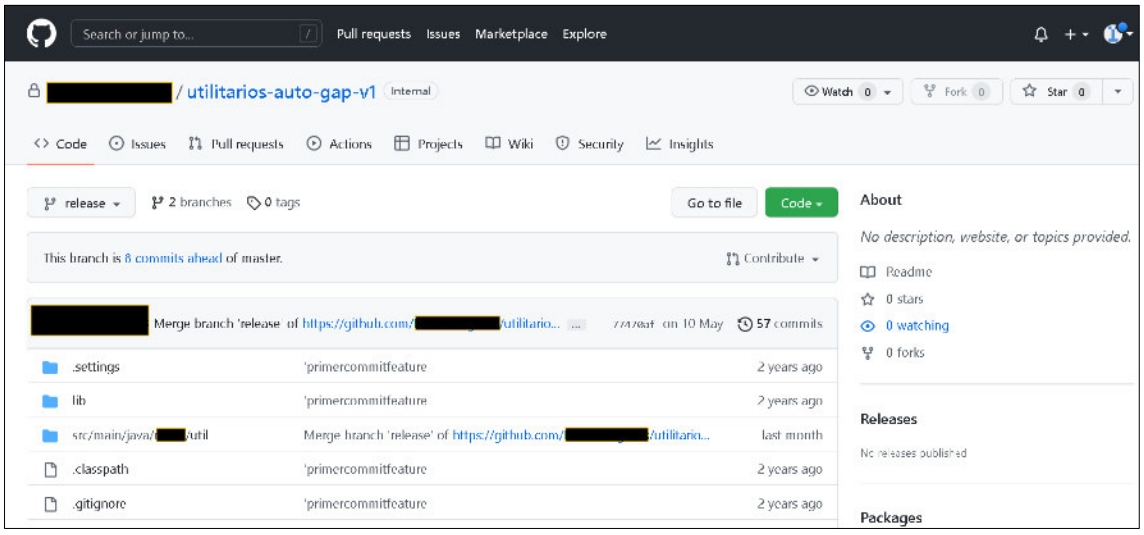


```
PageObjectUtil.java
1416            List<WebElement> lista = webDriver.findElements(by);
1417
1418            if (!lista.isEmpty()) {
1419                WebElement we = lista.get(pos);
1420                JavascriptExecutor js = (JavascriptExecutor) webDriver;
1421                js.executeScript("arguments[0].click();", we);
1422                sleep(1.5);
1423            }
1424        }
1425
1426        public void seleniumEscribirSinClickClear(WebDriver webDriver, final String path, int pos, String valor, Keys key) {
1427            By by = By.xpath(path);
1428            List<WebElement> lista = webDriver.findElements(by);
1429
1430            if (!lista.isEmpty()) {
1431                WebElement we = lista.get(pos);
1432
1433                if (we.isDisplayed()) {
1434                    sleep(0.25);
1435                    we.sendKeys(valor);
1436
1437                    if (key != null) {
1438                        sleep(1.50);
1439                        we.sendKeys(key);
1440                    }
1441                }
1442            }
1443        }
1444
1445        public void selectDropDown(WebDriver webDriver, final String xpath, String valor) {
1446            By by = By.xpath(xpath);
1447
1448            try {
1449                Select drpTipo = new Select(webDriver.findElement(by));
1450                drpTipo.selectByVisibleText(valor);
1451                sleep(1.50);
1452            }catch(Exception e) {
1453                System.out.println("No se pudo seleccionar el elemento");
1454            }
1455        }
1456    }
1457
```

Fuente: Código fuente del framework de automatización

Luego de terminar de implementar el proyecto utilitario-auto, se realiza el versionamiento en el repositorio GitHub (ver figura 45).

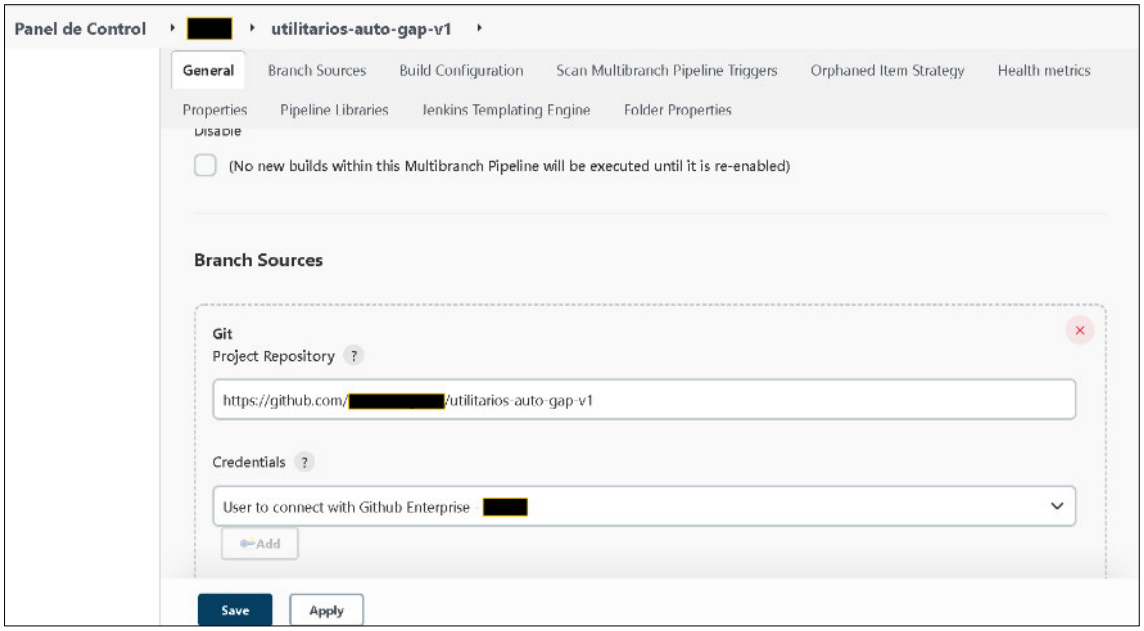
Figura 45: Versionamiento del proyecto utilitario-auto en GitHub



Fuente: Repositorio del proyecto de automatización

Después ingresar a Jenkins y crear un pipeline multibranch y realizar las configuraciones para que consuma el proyecto desde GitHub y genere la librería y lo publique en Nexus, cada vez que se realice un cambio en el proyecto se generara una nueva versión de la librería (ver figura 46).

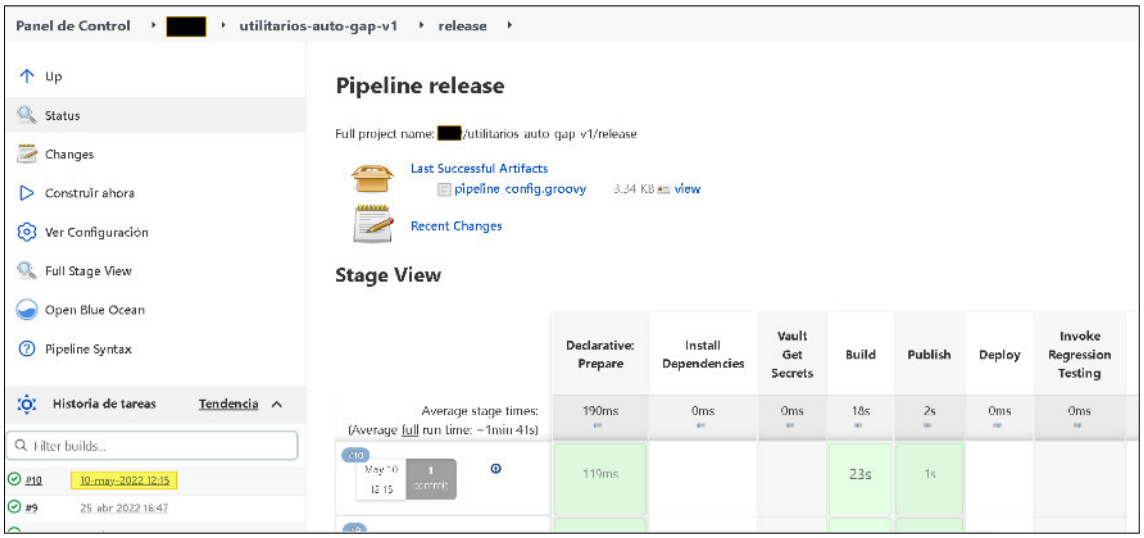
Figura 46: Configuración del pipeline multibranch utilitario-auto



Fuente: Jenkins del proyecto de automatización

Después de realizar la configuración, realizar la ejecución del pipeline (ver figura 47).

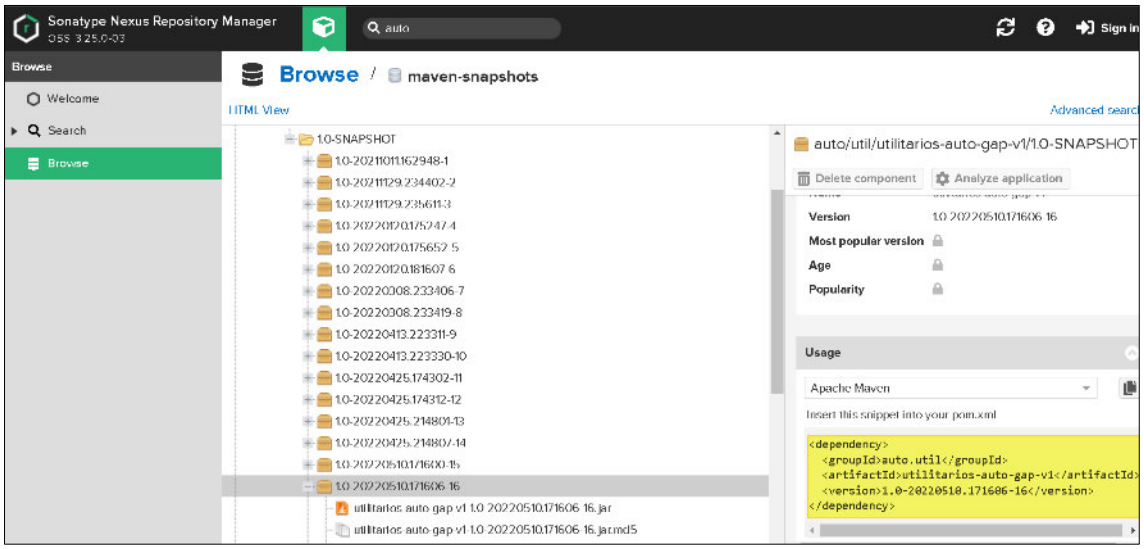
Figura 47: Ejecución del pipeline multibranch utilitario-auto



Fuente: Jenkins del proyecto de automatización

Una vez que termine la ejecución, ingresar a Nexus para obtener la versión de la dependencia, esta librería será usada en los proyectos ami-auto-def-test y ami-auto-app-test (ver figura 48).

Figura 48: Publicación de la librería utilitario-auto en Nexus



Fuente: Nexus del proyecto de automatización

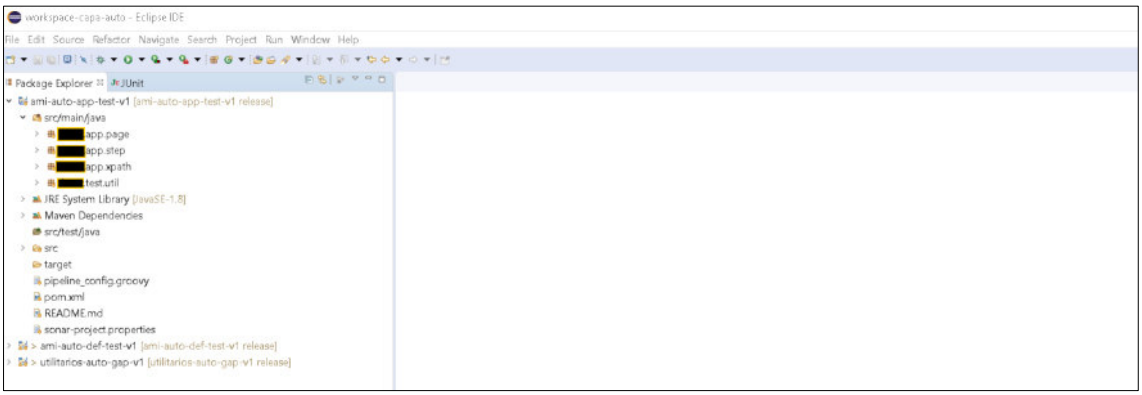
Desarrollo de los scripts de pruebas automatizadas para aplicaciones web:

Crear el proyecto ami-auto-app-test con la estructura definida.

- **Proyecto:** ami-auto-app-test

En la figura 49, se muestra la estructura del proyecto ami-auto-app-test.

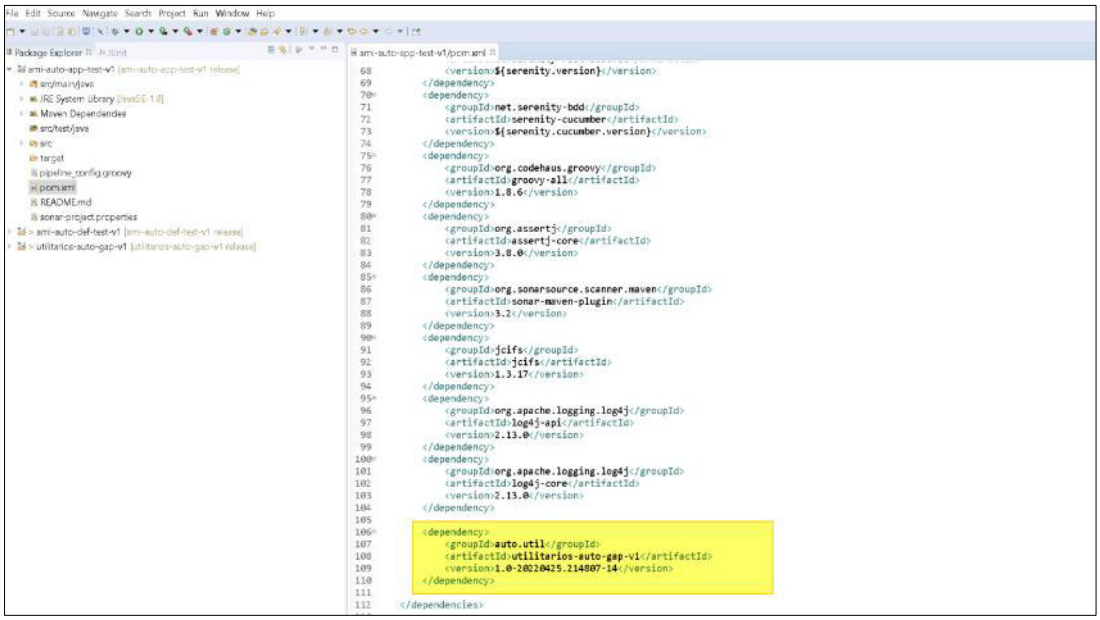
Figura 49: Estructura de carpetas del proyecto ami-auto-app-test



Fuente: Estructura de carpetas del framework de automatización

- Luego agregar las dependencias de Serenity, Selenium y del proyecto utilitario-auto en el archivo pom.xml para hacer uso de los métodos genéricos (ver figura 50).

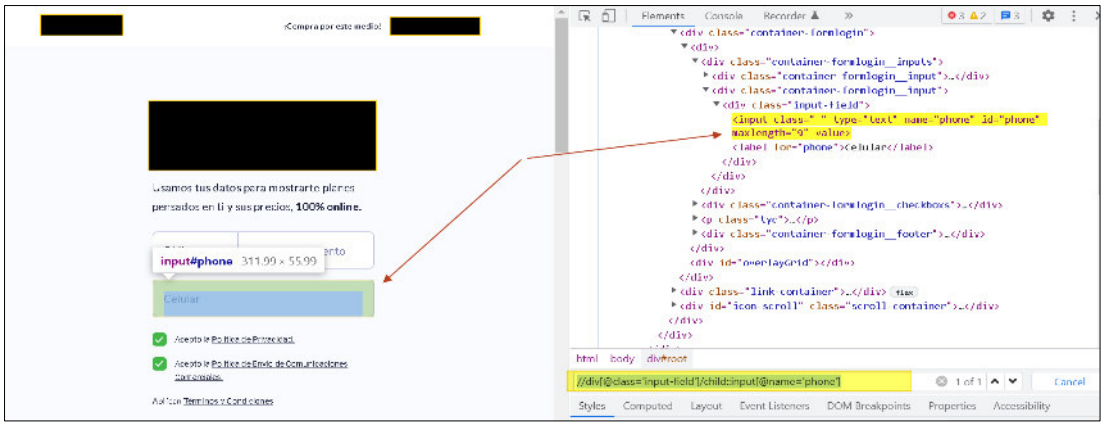
Figura 50: Dependencias en el archivo pom.xml del proyecto ami-auto-app-test



Fuente: Código fuente del framework de automatización

- Realizar el mapeo de los objetos requeridos para interactuar con los elementos de la aplicación web, para esto usar los localizadores tipo xpath relativo simple y compuesto ya que nos brindan una mayor precisión para ubicar los elementos (ver figura 51).

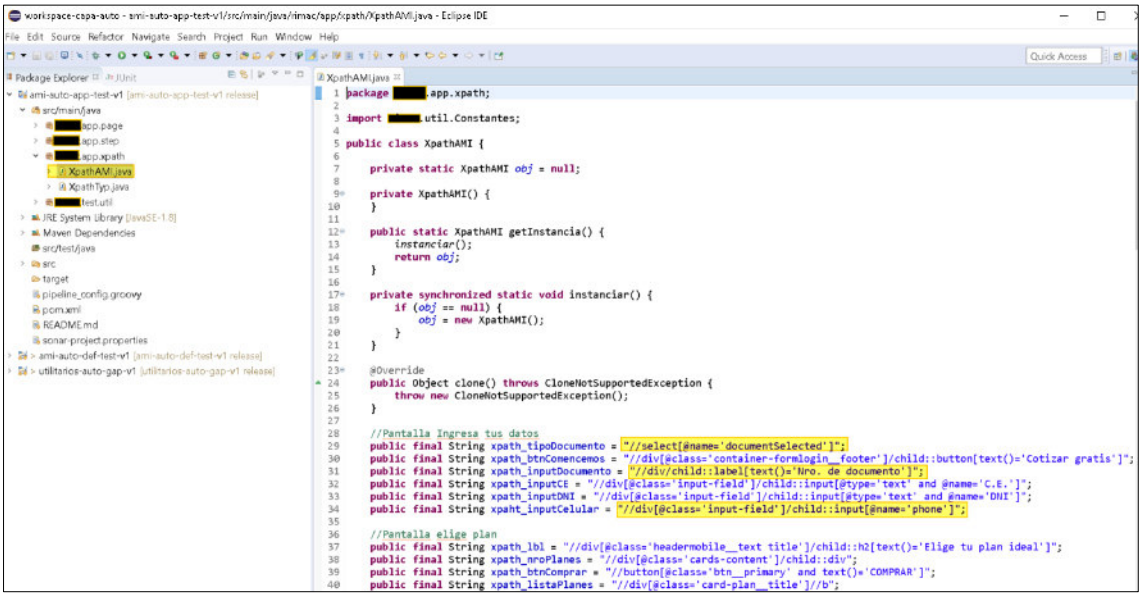
Figura 51: Mapeo de objetos de la aplicación web



Fuente: Sitio web del proyecto a automatizar

- Una vez mapeado los objetos de la aplicación web, se debe escribir en la clase XpathAMI ya que si cambian las características de los elementos solo será necesario modificarlas en esta clase y no en las que las van a consumir (ver figura 52).

Figura 52: Clase que contiene los objetos mapeados de la aplicación web



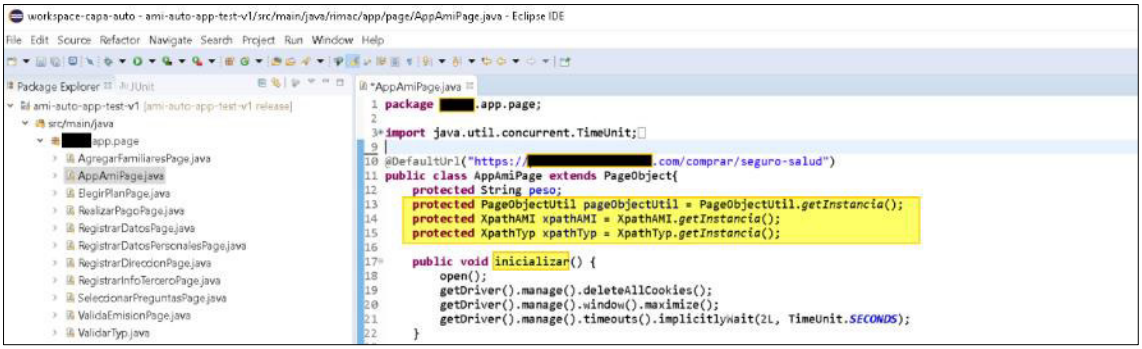
Fuente: Código fuente del framework de automatización

- Luego de mapear los objetos con los que vamos a interactuar, crear la clase base (AppAmiPage), donde:

    - Se especifica la URL de la aplicación web que se está automatizando.
    - Se instancia los xpath y pageObjectUtil para poder usar en los demás page que se crearan.
    - Se inicializa y abre el navegador web

    En la figura 53, se muestra la clase base page que contiene la URL y xpath de la aplicación web que se va automatizar.

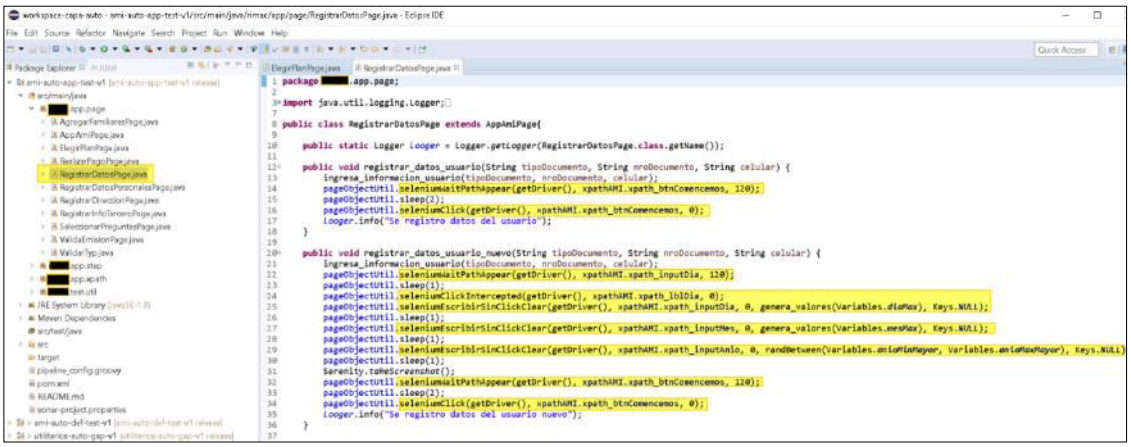
Figura 53: Clase base page



Fuente: Código fuente del framework de automatización

- Después crear una clase por pantalla de la aplicación web que se va a automatizar y heredar de la clase AppAmiPage para realizar las interacciones y acciones, como podemos ver en la figura 55, contiene las acciones y los paso a paso de las pruebas utilizando los xpath y métodos genéricos definidos en los utilitarios, también utilizar el método Serenity.takeScreenshot() para capturar las evidencias mientras se realiza la ejecución (ver figura 54).

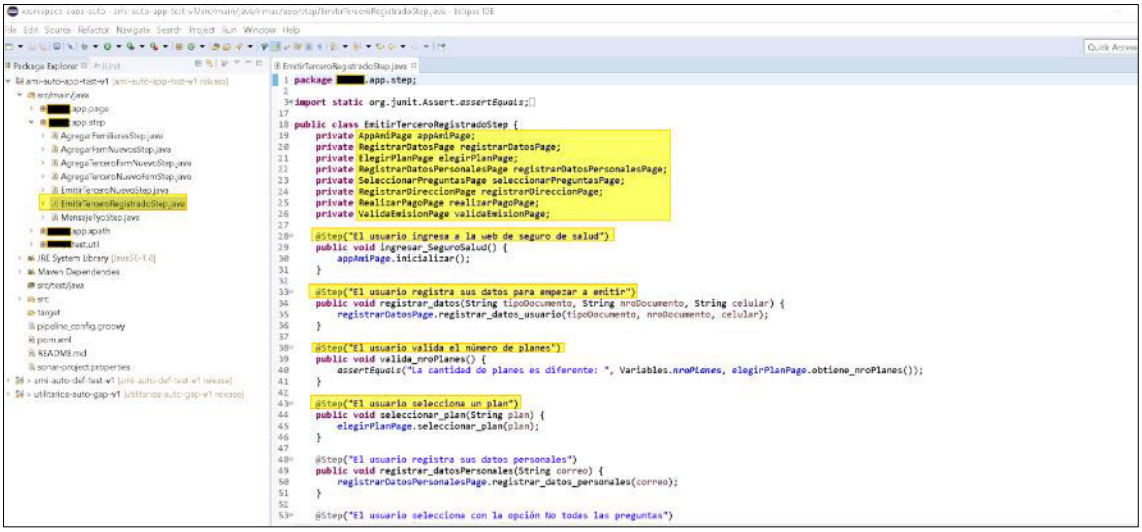
Figura 54: Clase page que hereda de la clase AppAmiPage



Fuente: Código fuente del framework de automatización

- Luego de mapear todas las pantallas de la aplicación web, crear las clases Step donde se instancia los page que se utilizara para definir los paso a paso y las validaciones que luego serán usados en los definition (ver figura 55).

Figura 55: Clase Step que utiliza los métodos page para realizar las acciones

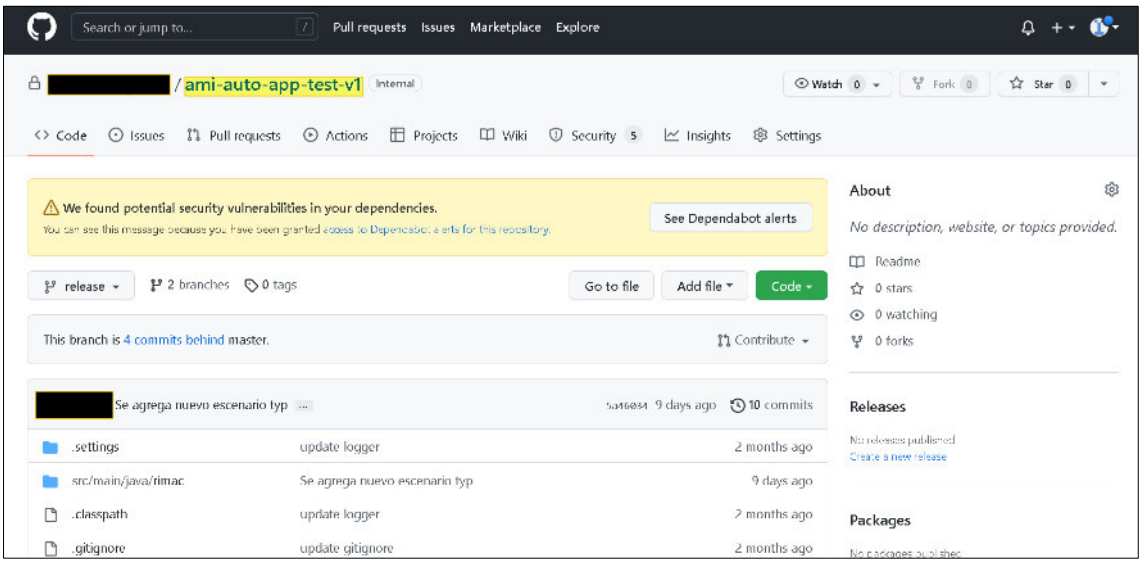


```
package [redacted].app.step;

import static org.junit.Assert.assertEquals;

public class EmitirTerceroRegistradoStep {
    private AppAmiPage appAmiPage;
    private RegistrarDatosPage registrarDatosPage;
    private ElegirPlanPage elegirPlanPage;
    private RegistrarDatosPersonalesPage registrarDatosPersonalesPage;
    private SeleccionarPreguntasPage seleccionarPreguntasPage;
    private RegistrarDireccionPage registrarDireccionPage;
    private RealizarPagoPage realizarPagoPage;
    private ValidaEmisionPage validaEmisionPage;

    @Step("El usuario ingresa a la web de seguro de salud")
    public void ingresar_SeguroSalud() {
        appAmiPage.inicializar();
    }

    @Step("El usuario registra sus datos para empezar a emitir")
    public void registrar_datos(String tipoDocumento, String nroDocumento, String celular) {
        registrarDatosPage.registrar_datos_usuario(tipoDocumento, nroDocumento, celular);
    }

    @Step("El usuario valida el número de planes")
    public void valida_nroPlanes() {
        assertEquals("La cantidad de planes es diferente: ", Variables.nroPlanes, elegirPlanPage.obtiene_nroPlanes());
    }

    @Step("El usuario selecciona un plan")
    public void seleccionar_plan(String plan) {
        elegirPlanPage.seleccionar_plan(plan);
    }

    @Step("El usuario registra sus datos personales")
    public void registrar_datosPersonales(String correo) {
        registrarDatosPersonalesPage.registrar_datos_personales(correo);
    }

    @Step("El usuario selecciona con la opción No todas las preguntas")
```

Fuente: Código fuente del framework de automatización

- Una vez finalizado el proyecto se realiza el versionamiento en el repositorio GitHub (ver figura 56).

Figura 56: Versionamiento en GitHub del proyecto ami-auto-app-test

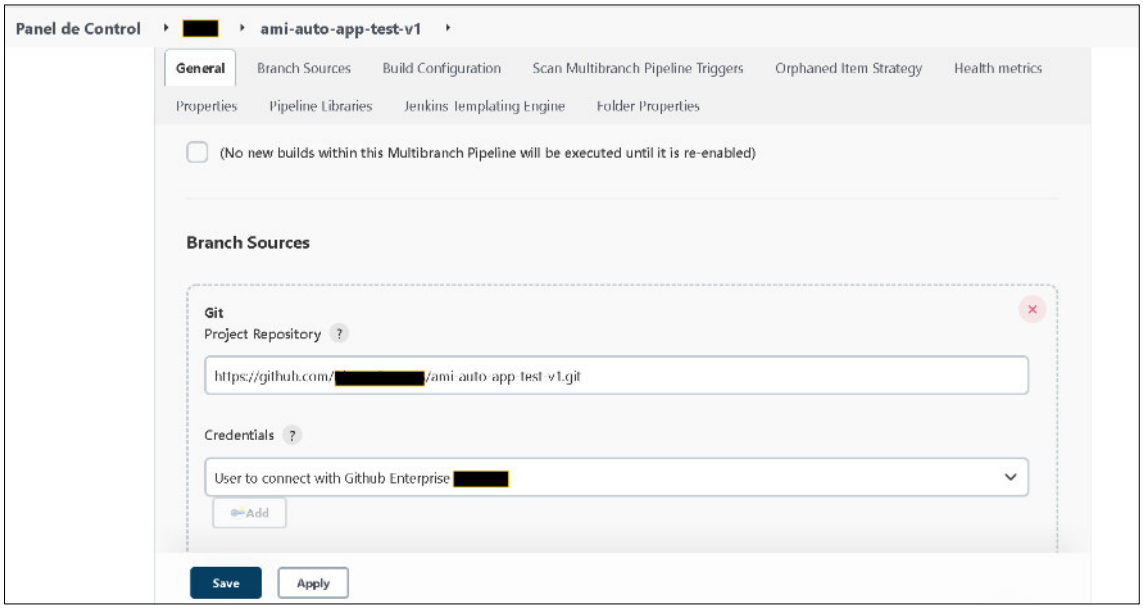


Fuente: Repositorio del proyecto de automatización

- Después de realizar el versionamiento ingresar a Jenkins y crear un pipeline multibranch para realizar las configuraciones para que consuma el proyecto ami-auto-app-test desde GitHub y genere la librería y lo

publique en Nexus, cada vez que se realice un cambio en el proyecto se generara una nueva versión de la librería (ver figura 57).

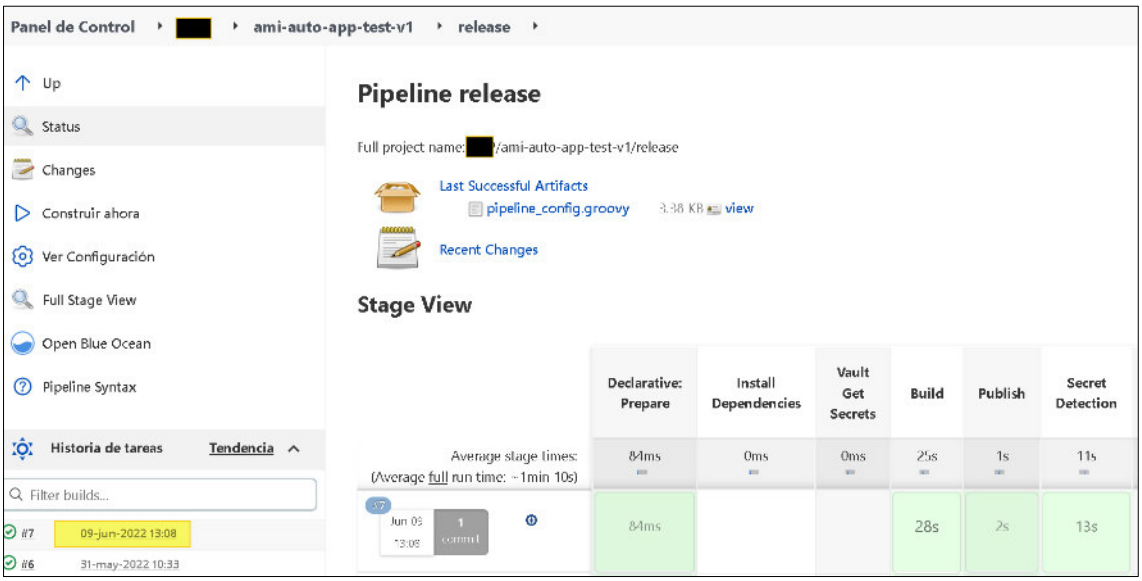
Figura 57: Configuración de proyecto ami-auto-app-test en un pipeline multibranch



Fuente: Jenkins del proyecto de automatización

En la figura 58, se muestra la ejecución del pipeline para la generación de la librería en Nexus.

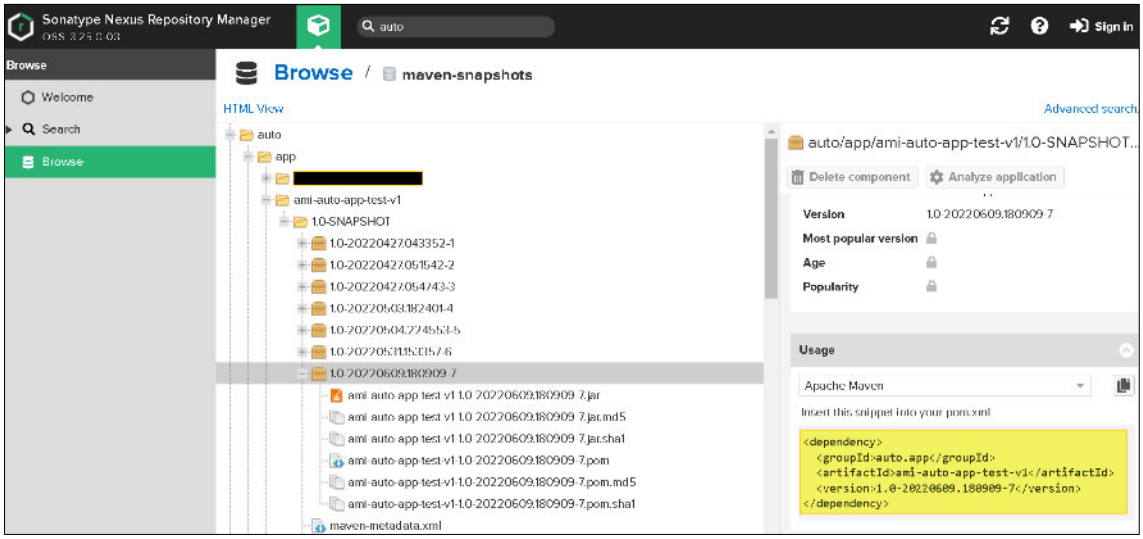
Figura 58: Ejecución del pipeline ami-auto-app-test para la generación de librería



Fuente: Jenkins del proyecto de automatización

Luego de la ejecución, ingresar a Nexus para obtener la versión de la dependencia, esta librería será usada en el proyecto ami-auto-def-test (ver figura 59).

Figura 59: Publicación de la librería en Nexus del proyecto ami-auto-app-test

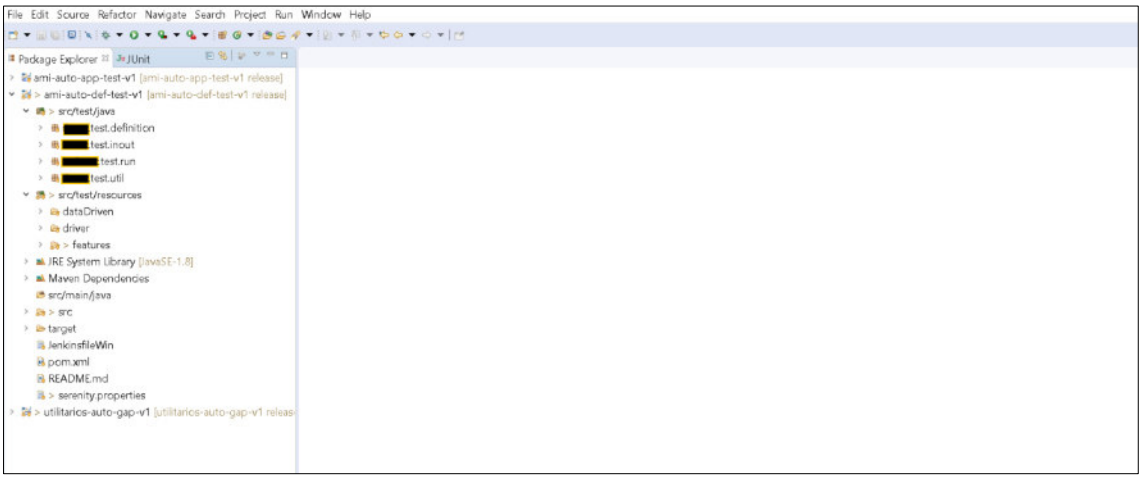


Fuente: Nexus del proyecto de automatización

Ahora se debe crear el proyecto ami-auto-def-test con la estructura de carpetas definida (ver figura 60).

- **Proyecto:** ami-auto-def-test

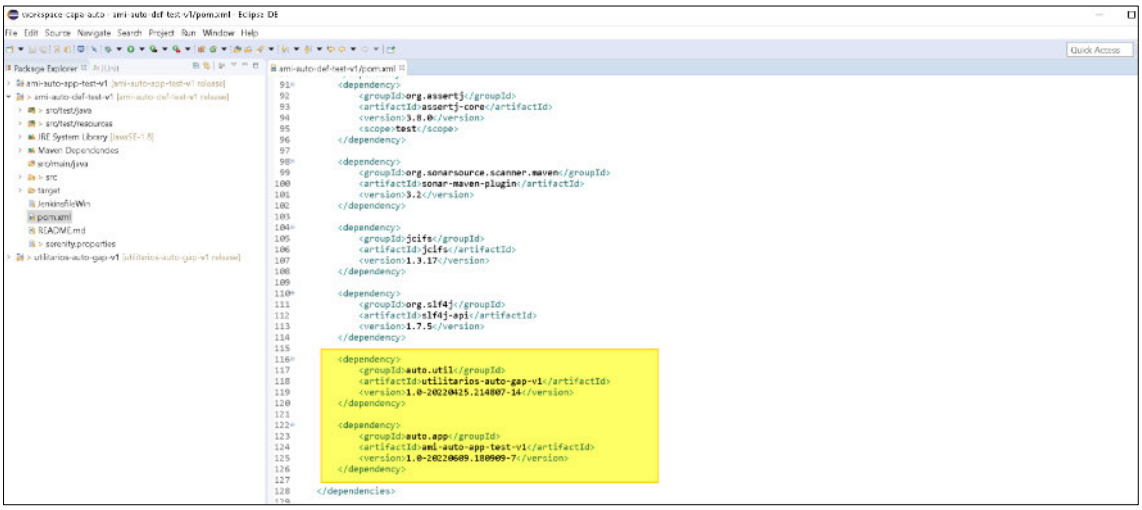
Figura 60: Estructura de carpetas del proyecto ami-auto-def-test



Fuente: Estructura de carpetas del proyecto de automatización

- Lo primero que se debe realizar es agregar como dependencia en el archivo pom.xml el proyecto utilitario-auto y ami-auto-app-test para hacer uso de los métodos genéricos creados (ver figura 61).

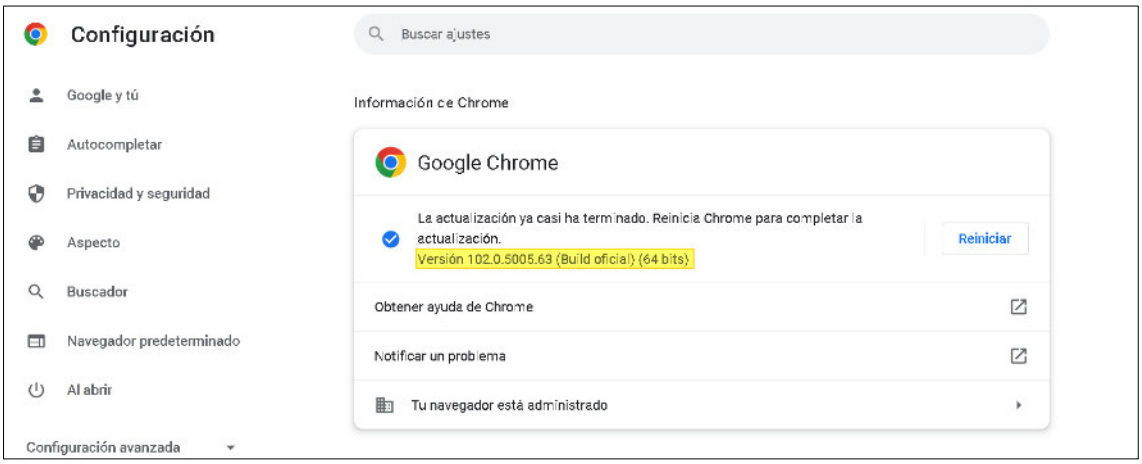
Figura 61: Configuración del archivo pom.xml con las dependencias de los proyectos ami-auto-app-test y utilitarios-auto



Fuente: Código fuente del framework de automatización

- Ahora descargar el driver del navegador con el que se va a trabajar, en este caso se trabajara con Chrome; se debe validar la versión de Chrome para realizar la descarga del driver (ver figura 62).

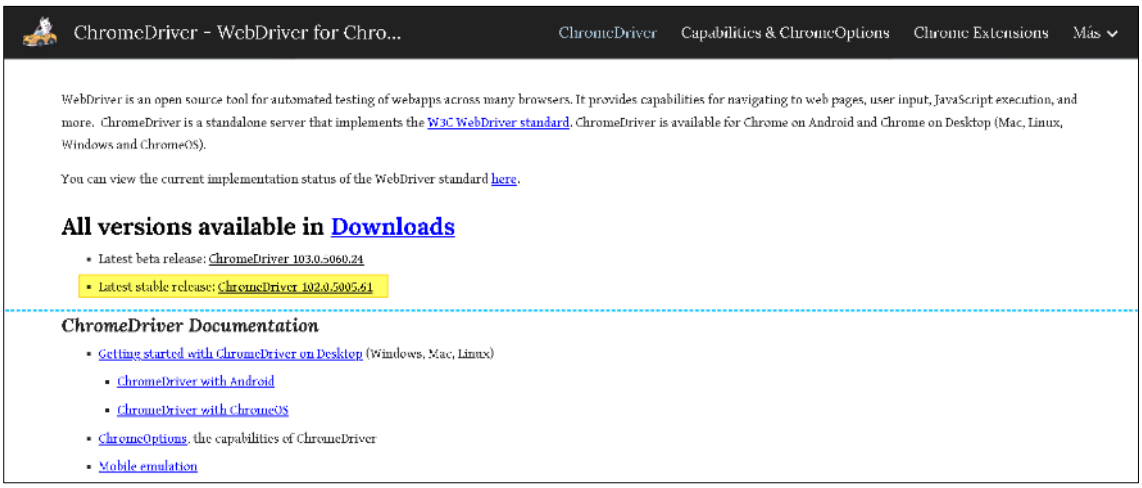
Figura 62: Validación de la versión del navegador



Fuente: Navegador web utilizado para la automatización

- Después de validar la versión, ingresar a descargar la versión 102 del driver de Chrome (ver figura 63).

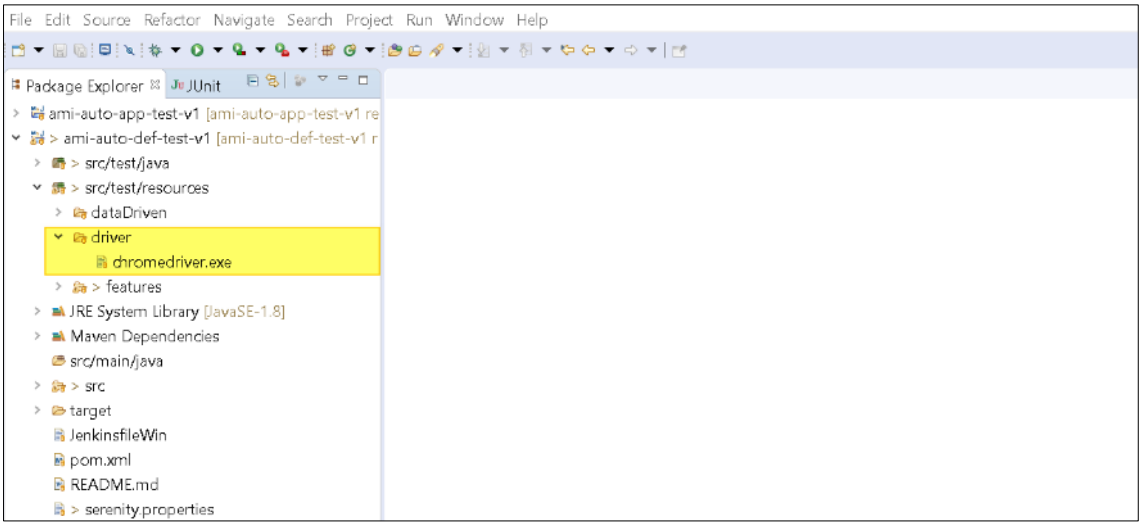
Figura 63: Página oficial de descarga del driver de Chrome



Fuente: https://chromedriver.chromium.org/downloads

- Una vez que se descarga el driver, se debe agregar el driver en la carpeta especifica del proyecto (ver figura 64).

Figura 64: Ubicación de la carpeta driver en el proyecto

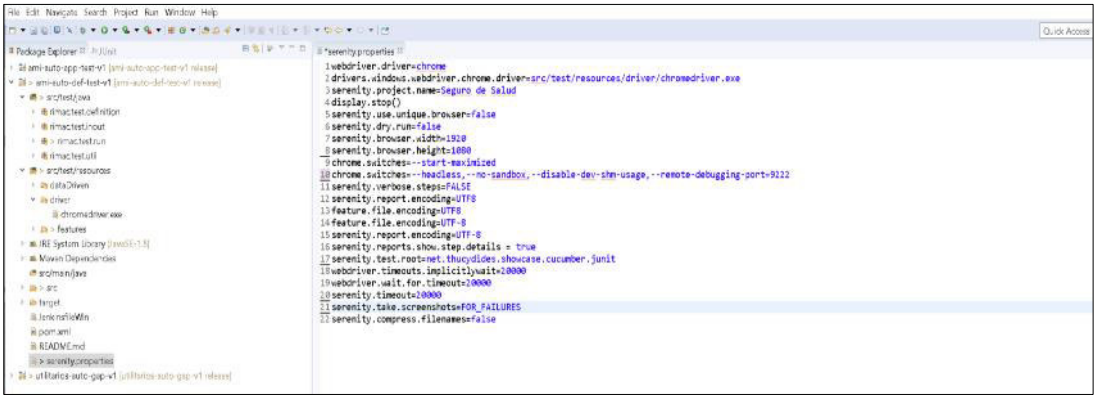


Fuente: Código fuente del framework de automatización

- Luego se debe realizar las configuraciones en el archivo serenity.properties del proyecto (ver figura 65):

    - Se indica el navegador con el que se va a trabajar.
    - Se indica al artefacto las rutas donde se encuentran los drivers de los navegadores que vamos a usar.
    - Se indica que se va a usar varios navegadores.
    - Se indica que el navegador se inicie maximizado.
    - Se indica para que reconozca los caracteres especiales en los reportes que se van a generar.
    - Se indica un tiempo específico de espera para que reconozca los componentes de la web, de lo contrario debe mostrar un Timeout.
    - Se indica para que realice capturas de pantallas cuando falla la aplicación web y no se tiene mapeado.

Figura 65: Configuraciones del archivo serenity.properties

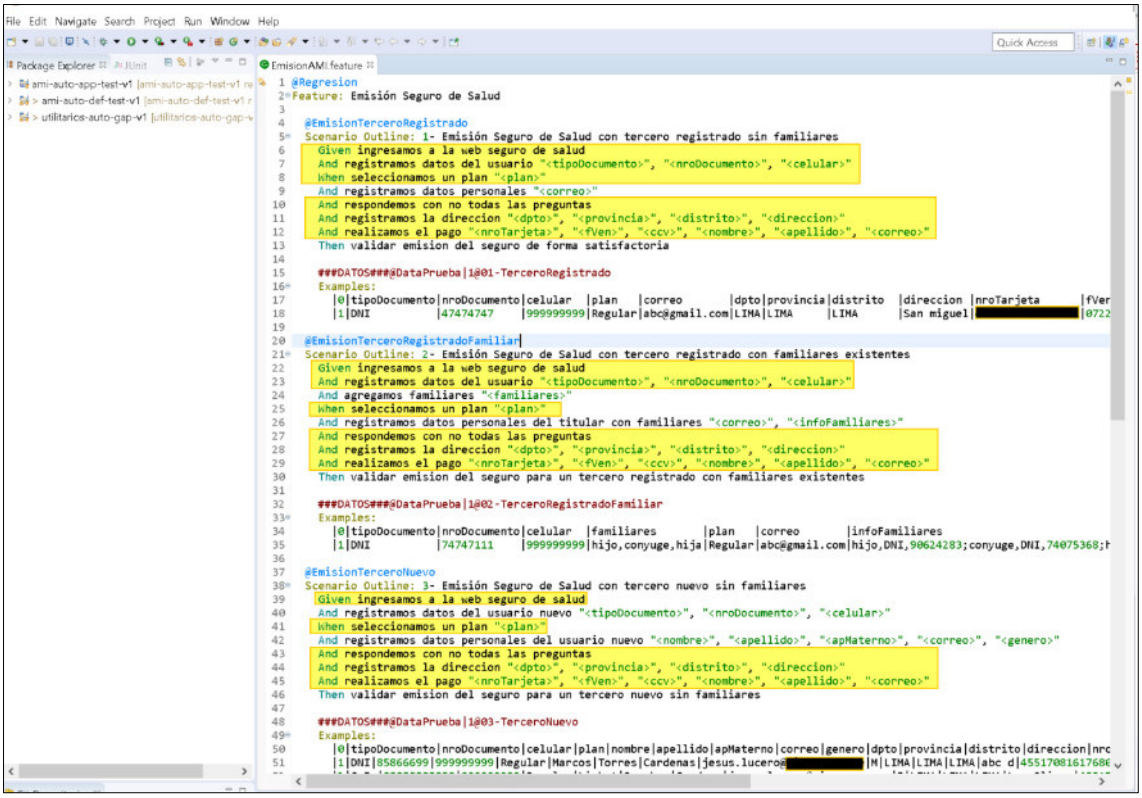


Fuente: Código fuente del framework de automatización

- Luego se debe escribir el escenario de prueba diseñado en lenguaje Gherkin en Cucumber y los datos de entrada serán consumidos desde un Excel al momento de realizar las ejecuciones.

  Como se puede visualizar las líneas sombreadas de amarillo escritas en el primer escenario son reutilizados por los demás escenarios permitiendo agilizar la implementación (ver figura 66).

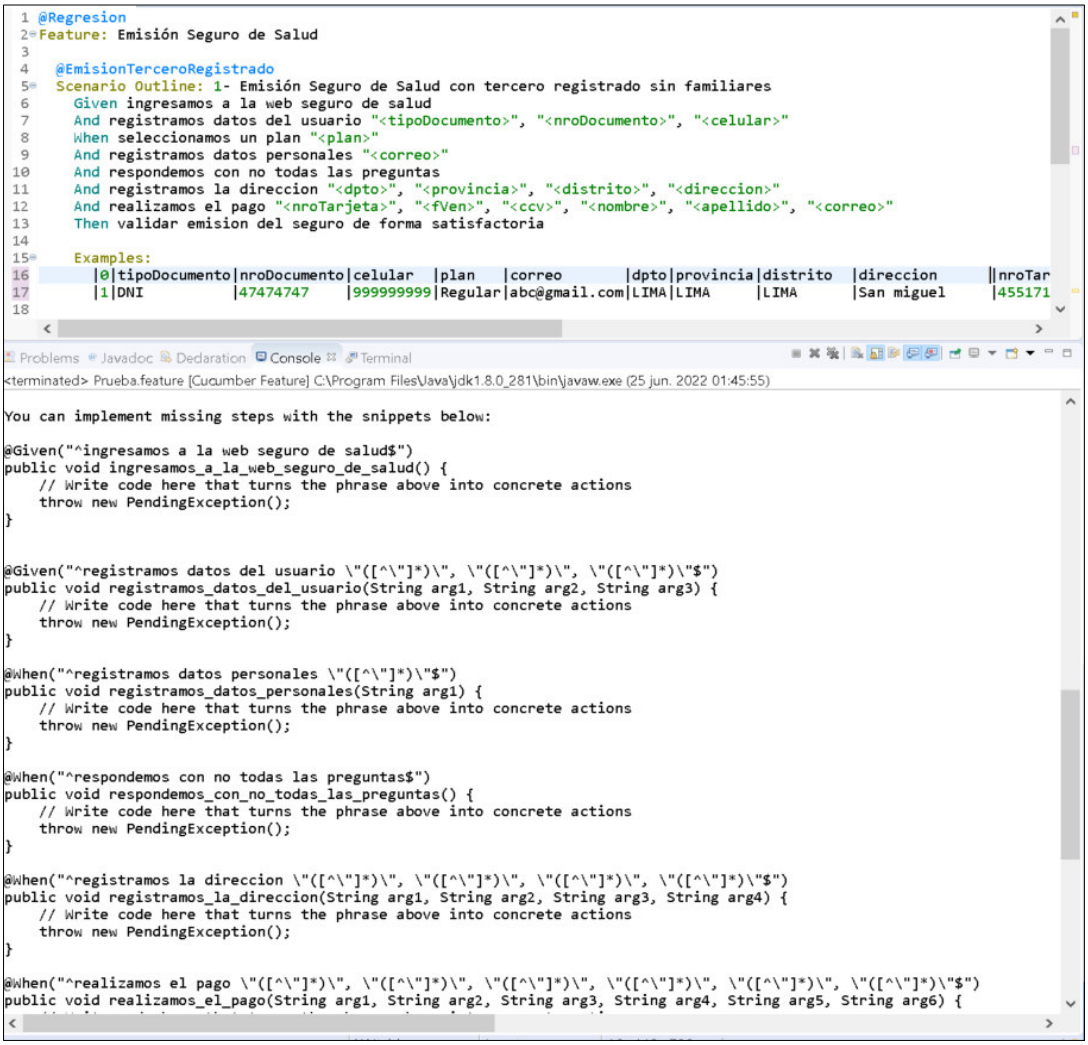
Figura 66: Diseño del escenario de prueba en el feature en lenguaje Gherkin



Fuente: Código fuente del framework de automatización

- Luego ejecutar el archivo feature para generar los métodos a implementar en la clase definitions (Cada línea del escenario es equivalente a un método que será utilizado en los definitions) (ver figura 67).

Figura 67*:* Generación de métodos desde los feature



```
@Regresion
Feature: Emisión Seguro de Salud

  @EmisionTerceroRegistrado
  Scenario Outline: 1- Emisión Seguro de Salud con tercero registrado sin familiares
    Given ingresamos a la web seguro de salud
    And registramos datos del usuario "<tipoDocumento>", "<nroDocumento>", "<celular>"
    When seleccionamos un plan "<plan>"
    And registramos datos personales "<correo>"
    And respondemos con no todas las preguntas
    And registramos la direccion "<dpto>", "<provincia>", "<distrito>", "<direccion>"
    And realizamos el pago "<nroTarjeta>", "<fVen>", "<ccv>", "<nombre>", "<apellido>", "<correo>"
    Then validar emision del seguro de forma satisfactoria

    Examples:
      |0|tipoDocumento|nroDocumento|celular  |plan   |correo        |dpto|provincia|distrito  |direccion     ||nroTar
      |1|DNI          |47474747    |999999999|Regular|abc@gmail.com|LIMA|LIMA     |LIMA      |San miguel    |455171
```

Problems Javadoc Dedaration Console Terminal

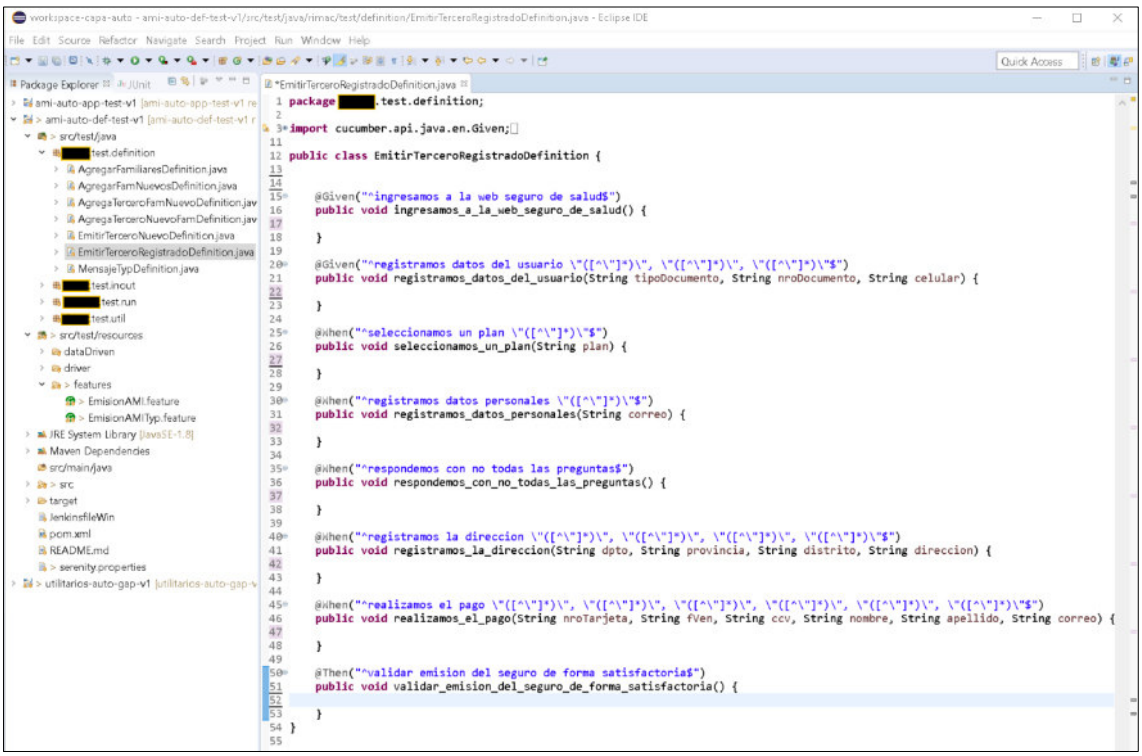
<terminated> Prueba.feature [Cucumber Feature] C:\Program Files\Java\jdk1.8.0_281\bin\javaw.exe (25 jun. 2022 01:45:55)

```
You can implement missing steps with the snippets below:

@Given("^ingresamos a la web seguro de salud$")
public void ingresamos_a_la_web_seguro_de_salud() {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}


@Given("^registramos datos del usuario \"([^\"]*)\", \"([^\"]*)\", \"([^\"]*)\"$")
public void registramos_datos_del_usuario(String arg1, String arg2, String arg3) {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^registramos datos personales \"([^\"]*)\"$")
public void registramos_datos_personales(String arg1) {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^respondemos con no todas las preguntas$")
public void respondemos_con_no_todas_las_preguntas() {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^registramos la direccion \"([^\"]*)\", \"([^\"]*)\", \"([^\"]*)\", \"([^\"]*)\"$")
public void registramos_la_direccion(String arg1, String arg2, String arg3, String arg4) {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^realizamos el pago \"([^\"]*)\", \"([^\"]*)\", \"([^\"]*)\", \"([^\"]*)\", \"([^\"]*)\", \"([^\"]*)\"$")
public void realizamos_el_pago(String arg1, String arg2, String arg3, String arg4, String arg5, String arg6) {
```

Fuente: Código fuente del framework de automatización

- Luego de generar los métodos desde los feature, se deben implementar en la capa de los definitions (Son componentes reutilizables por otros escenarios al momento de realizar las automatizaciones) donde cada línea Gherkin definida en la feature mapea con una anotación (@Given, @When, @Then) en la definition (ver figura 68).

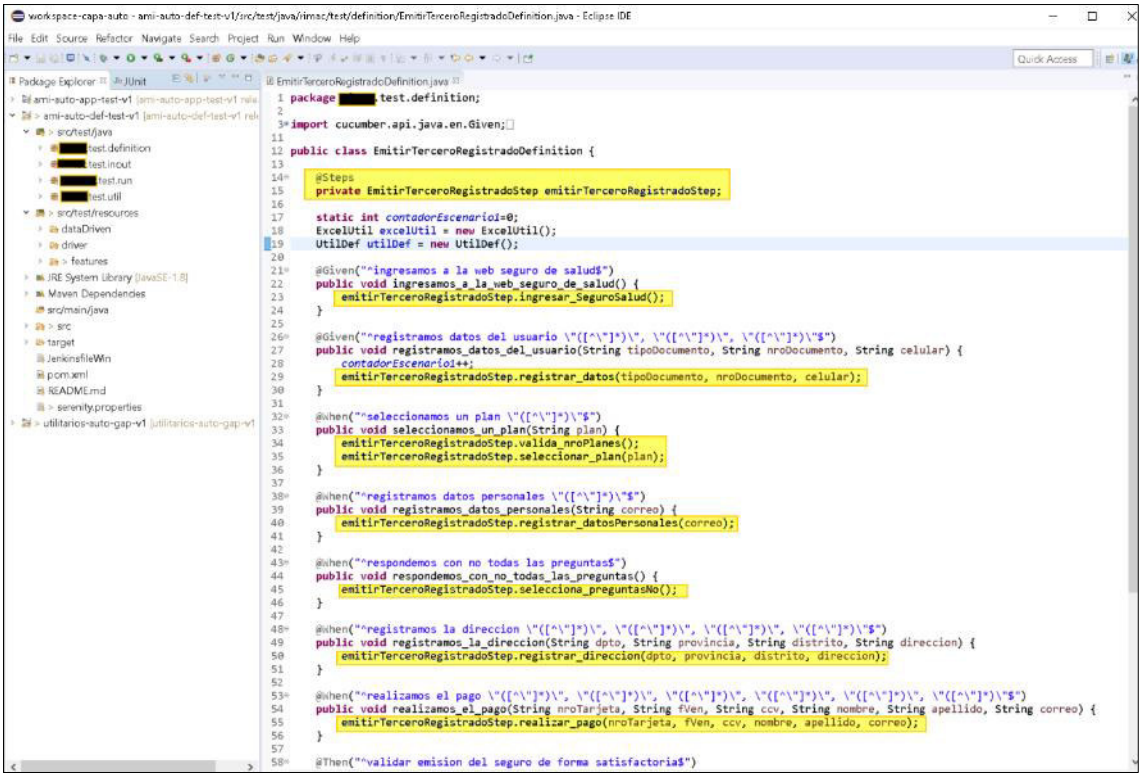
Figura 68: Métodos mapeados con anotaciones en los definitions



Fuente: Código fuente del framework de automatización

- En estas clases no se realiza ninguna lógica de programación solo se invocan a los métodos de los Steps (Son las clases con los pasos o acciones a realizar para una definition) y así enlazar la clase definition con la clase Step por lo tanto estas clases definitions no se ve impactado si en la aplicación web cambian las características de los elementos mapeados (ver figura 69).

Figura 69: Instanciación de las clases Steps en los definitions



Fuente: Código fuente del framework de automatización

- También se debe agregar los datos de pruebas que estarán ubicados en la carpeta dataDriven y estos serán leídos y escritos en los features al momento de ejecutar las pruebas de forma local (ver figura 70).

Figura 70: Ubicación de los datos de pruebas para la ejecución local

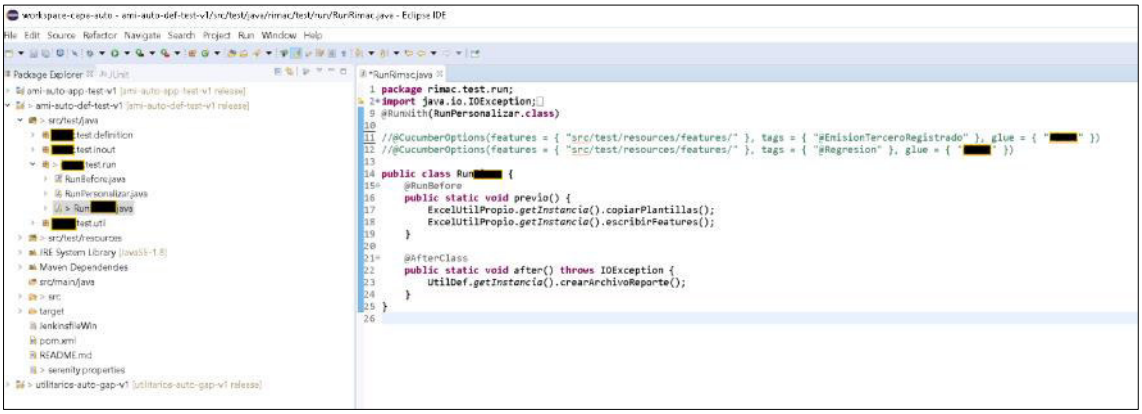
Fuente: Código fuente del framework de automatización

- Ahora se debe definir a la clase ejecutora que es la conexión entre Cucumber y el código, donde la clase runner es el orquestador y gracias a esta clase el proyecto podrá ser ejecutado y permitirá esa conexión, donde le especificará los escenarios y tags de una feature en particular para su ejecución, donde:

    - **Features:** Aquí se coloca la ubicación del archivo feature.
    - **Glue:** Aquí se coloca la ubicación de los definitions.
    - **Tags:** Aquí se inserta el escenario o escenarios que se van a ejecutar.

  Cada vez que se ejecute el runner buscara el archivo con extensión .feature que se ha especificado, donde se encuentra cada uno de los paso a paso de los escenarios que se desea realizar (ver figura 71).

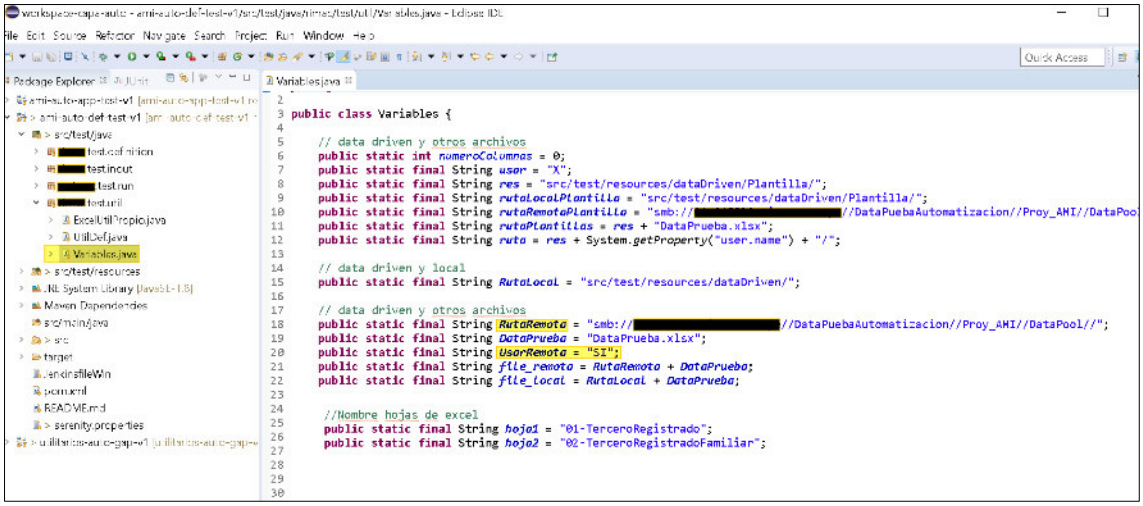
Figura 71: Configuración de la clase runner



Fuente: Código fuente del framework de automatización

- Luego de implementar de forma local, se debe configurar el proyecto para que consuma los datos de prueba de una ruta compartida para realizar la integración con Jenkins (ver figura 72).

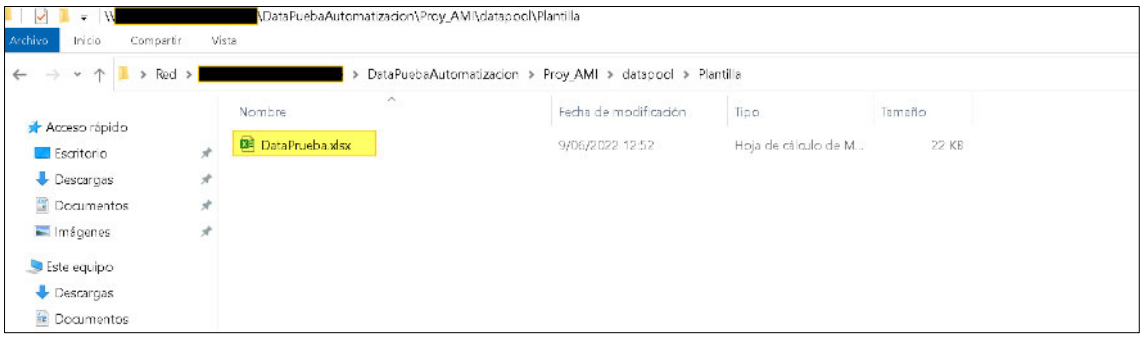
Figura 72: Configuración de los datos de prueba de una ruta compartida



Fuente: Código fuente del framework de automatización

En la figura 73, se muestra la ruta compartida de la carpeta de donde se consumirán los datos para las ejecuciones de las pruebas.

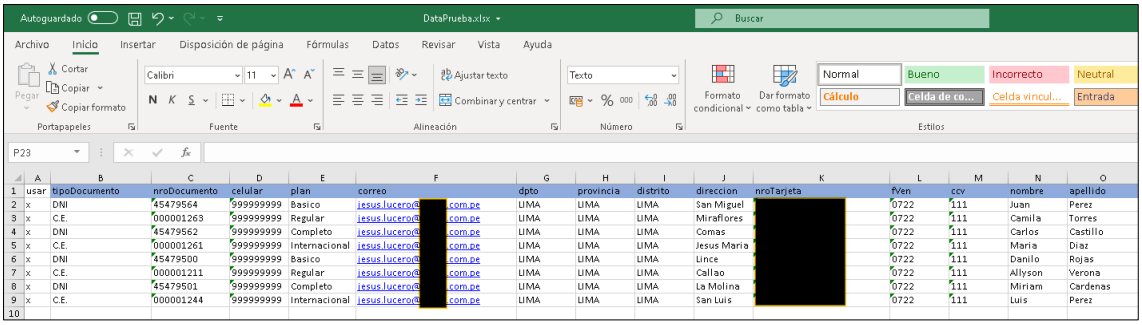
Figura 73: Ubicación del Excel en una ruta compartida



Fuente: Ruta compartida del proyecto de automatización

En la figura 74, se muestra la estructura del Excel de la carpeta compartida.

Figura 74: Archivo Excel compartido con los datos de pruebas

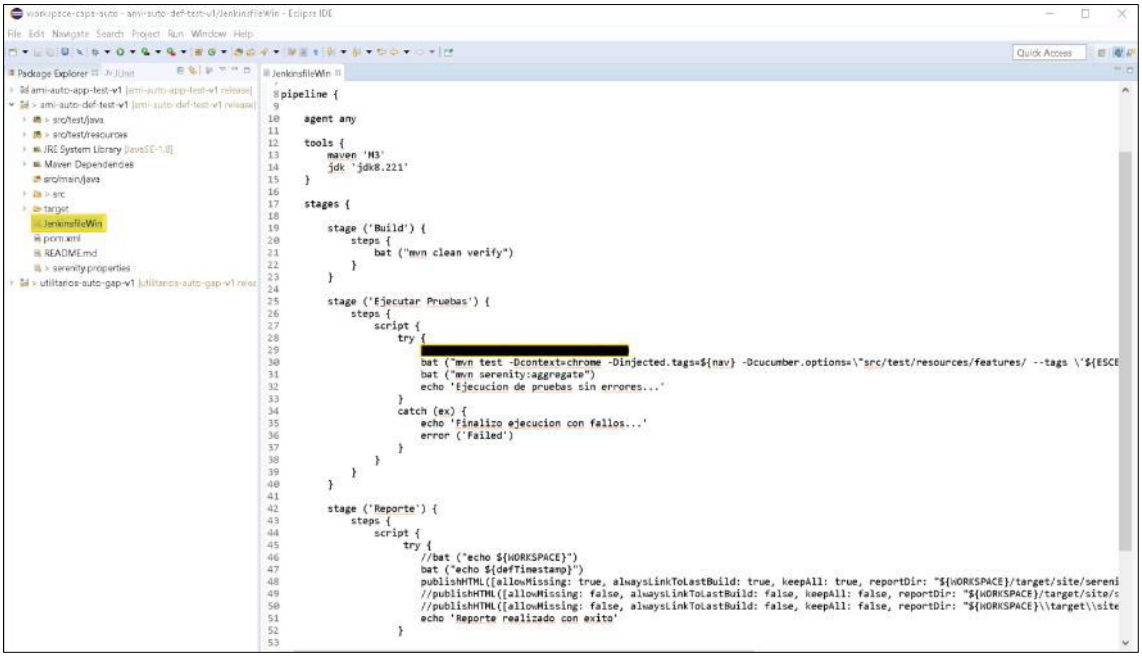


usar	tipoDocumento	nroDocumento	celular	plan	correo	dpto	provincia	distrito	direccion	nroTarjeta	fVen	ccv	nombre	apellido
x	DNI	45479564	999999999	Basico	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	San Miguel	[redacted]	0722	111	Juan	Perez
x	C.E.	000001263	999999999	Regular	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	Miraflores		0722	111	Camila	Torres
x	DNI	45479562	999999999	Completo	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	Comas		0722	111	Carlos	Castillo
x	C.E.	000001261	999999999	Internacional	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	Jesus Maria		0722	111	Maria	Diaz
x	DNI	45479500	999999999	Basico	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	Lince		0722	111	Danilo	Rojas
x	C.E.	000001211	999999999	Regular	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	Callao		0722	111	Allyson	Verona
x	DNI	45479501	999999999	Completo	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	La Molina		0722	111	Miriam	Cardenas
x	C.E.	000001244	999999999	Internacional	jesus.lucero[redacted].com.pe	LIMA	LIMA	LIMA	San Luis		0722	111	Luis	Perez

Fuente: Excel utilizado en el proyecto de automatización

- Luego se debe realizar las configuraciones en el archivo Jenkinsfile, que va a permitir realizar la integración con Jenkins para la ejecución y la generación del reporte, donde se debe usar la siguiente sintaxis (ver figura 75):
    - **Pipeline:** Es donde empieza y termina el pipeline, así como los pasos que tiene.
    - **Agent:** Aquí se le especifica cuando se ejecuta el pipeline y se hace uso del comando Any para ejecutar el pipeline siempre y cuando haya un ejecutor libre en Jenkins.
    - **Stages:** Es el bloque donde se define una serie de estados a realizar dentro del pipeline.
    - **Stage:** Es el bloque donde se define una serie de tareas que se va a realizar dentro del pipeline, en este caso se hace uso de: Build, Ejecutar Pruebas, Reporte.
    - **Steps:** Son todos los pasos que se van a realizar dentro de un Stage, se puede definir uno o más pasos.

Figura 75: Configuración del archivo Jenkinsfile del proyecto web

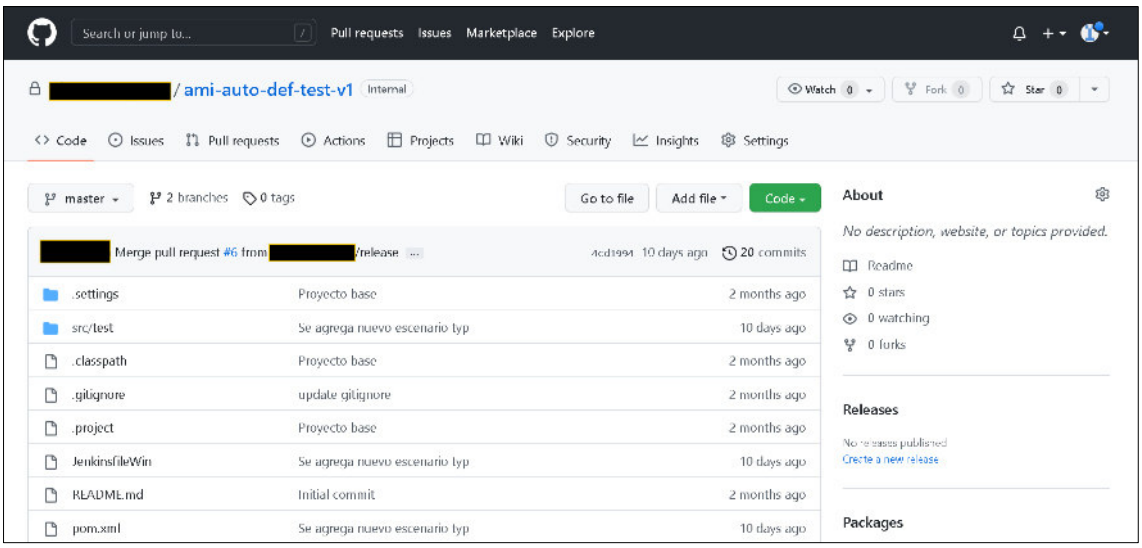


```
pipeline {

    agent any

    tools {
        maven 'M3'
        jdk 'jdk8.221'
    }

    stages {

        stage ('Build') {
            steps {
                bat ("mvn clean verify")
            }
        }

        stage ('Ejecutar Pruebas') {
            steps {
                script {
                    try {
                        [redacted]
                        bat ("mvn test -Dcontext=chrome -Dinjected.tags=${nav} -Dcucumber.options=\"src/test/resources/features/ --tags \'${ESCE
                        bat ("mvn serenity:aggregate")
                        echo 'Ejecucion de pruebas sin errores...'
                    }
                    catch (ex) {
                        echo 'Finalizo ejecucion con fallos...'
                        error ('Failed')
                    }
                }
            }
        }

        stage ('Reporte') {
            steps {
                script {
                    try {
                        //bat ("echo ${WORKSPACE}")
                        bat ("echo ${defTimestamp}")
                        publishHTML([allowMissing: true, alwaysLinkToLastBuild: true, keepAll: true, reportDir: "${WORKSPACE}/target/site/sereni
                        //publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false, reportDir: "${WORKSPACE}/target/site/s
                        //publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false, reportDir: "${WORKSPACE}\\target\\site
                        echo 'Reporte realizado con exito'
                    }
```

Fuente: Código fuente del framework de automatización

- Ahora se debe realizar el versionamiento del proyecto en el repositorio de GitHub (ver figura 76).

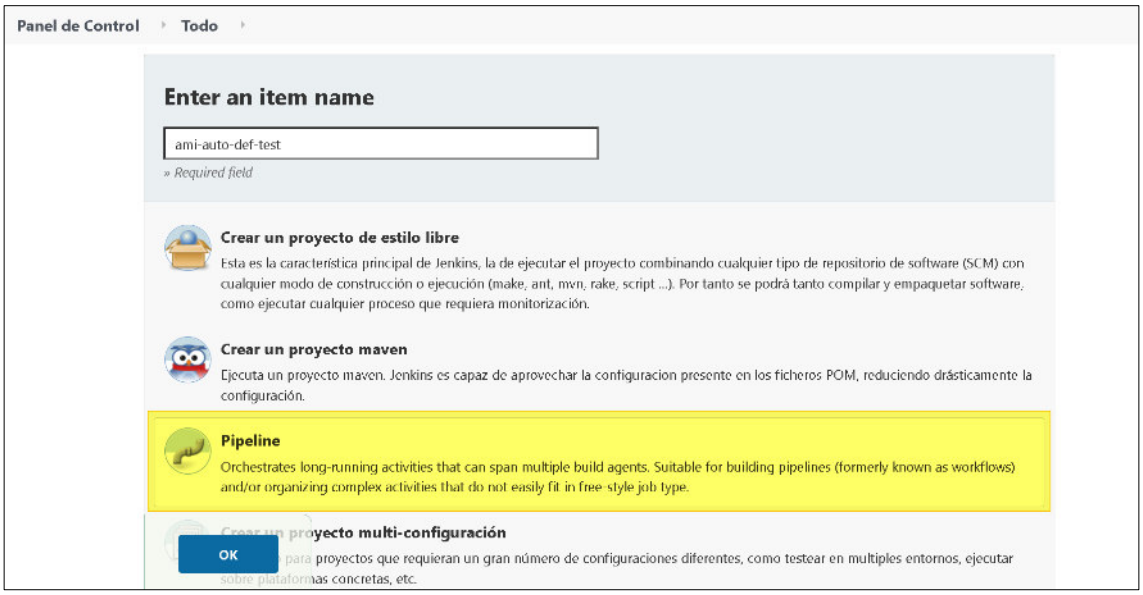
Figura 76: Versionamiento en GitHub del proyecto ami-auto-def-test



Fuente: Repositorio del proyecto de automatización

- Una vez que se tiene versionado el proyecto ami-auto-def-test, ingresar a Jenkins para crear el pipeline y realizar las configuraciones (ver figura 77).

Figura 77: Creación del pipeline para el proyecto ami-auto-def-test



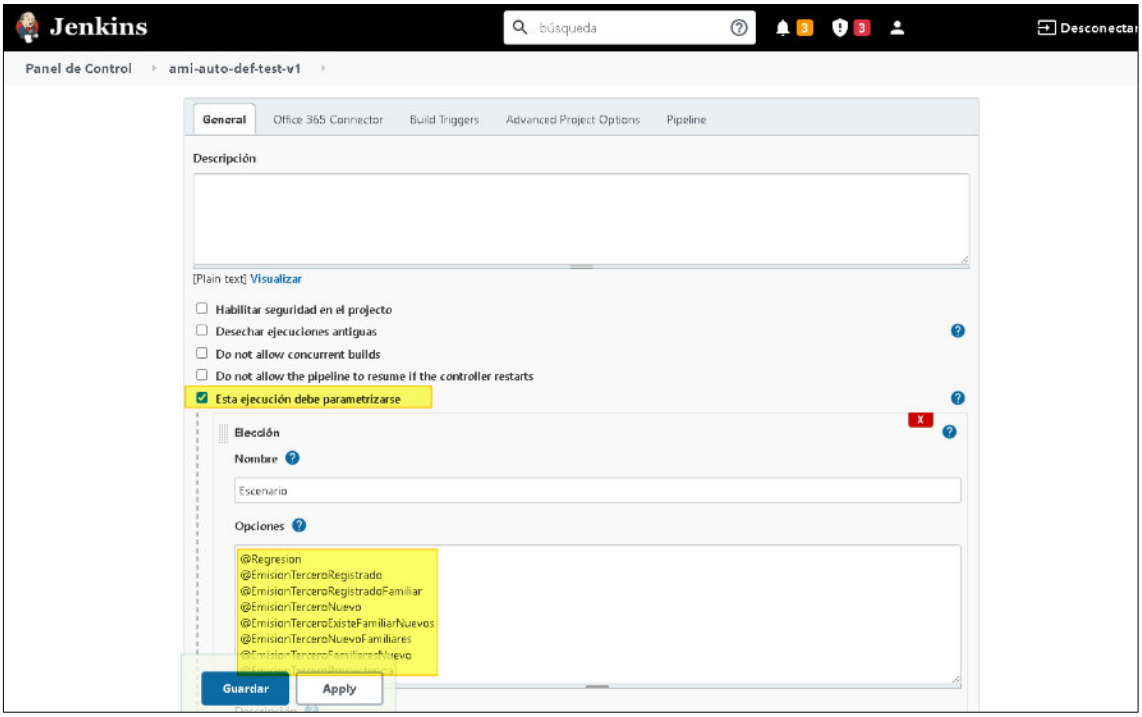
Fuente: Jenkins del proyecto de automatización

- A continuación, se debe añadir la configuración siguiente:

  - Seleccionar la opción de ejecución debe parametrizarse y colocar los escenarios individuales y grupales (regresión).
  - Se debe hacer referencia a la URL del repositorio GitHub y la rama donde se tiene el código fuente.
  - Colocar el nombre del archivo Jenkinsfile en la opción script path.

  En la figura 78, se muestra la configuración de los escenarios automatizados en Jenkins.

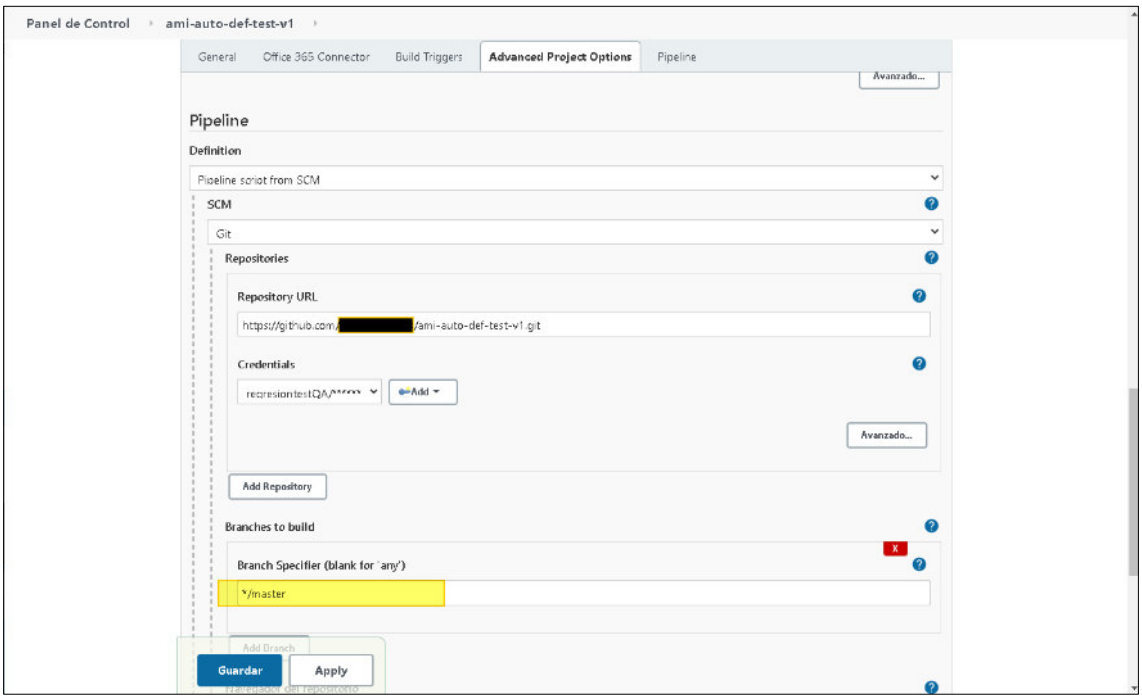
Figura 78: Configuración para la ejecución parametrizado



Fuente: Jenkins del proyecto de automatización

En la figura 79, se muestra la configuración en Jenkins del proyecto desde GitHub.

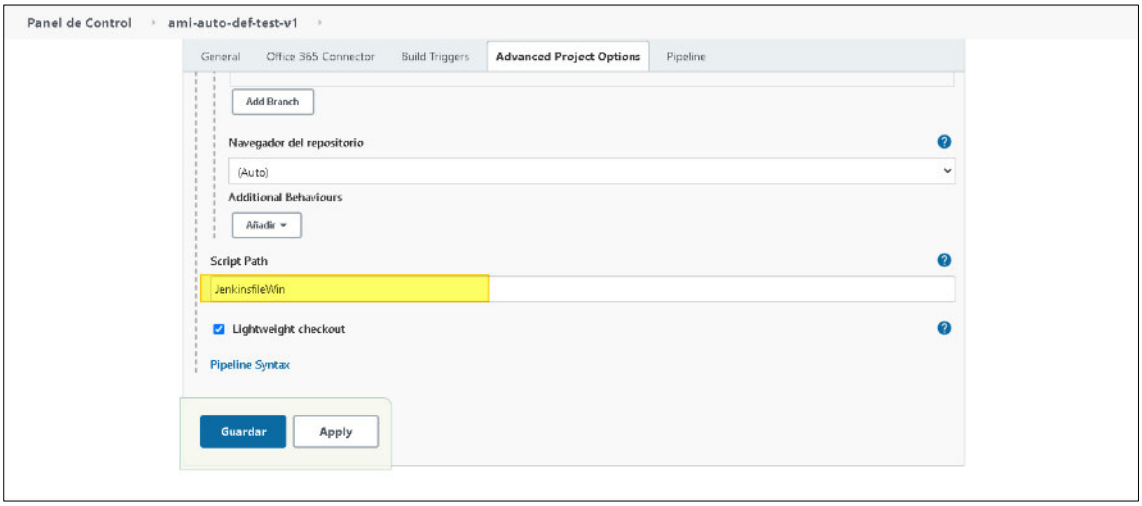
Figura 79: Referencia al proyecto auto-ami-def-test versionado



Fuente: Jenkins del proyecto de automatización

En la figura 80, se muestra la configuración en Jenkins del archivo Jenkinsfile para su ejecución.

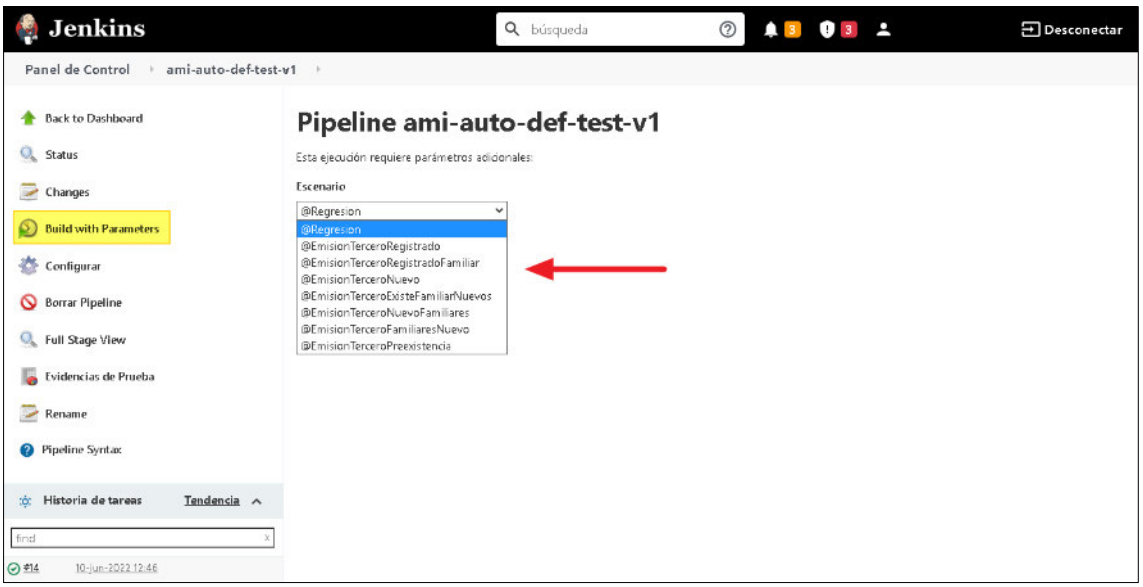
Figura 80: Configuración del archivo Jenkinsfile en Jenkins



Fuente: Jenkins del proyecto de automatización

- Luego realizar la ejecución parametrizada desde la herramienta Jenkins (ver figura 81).

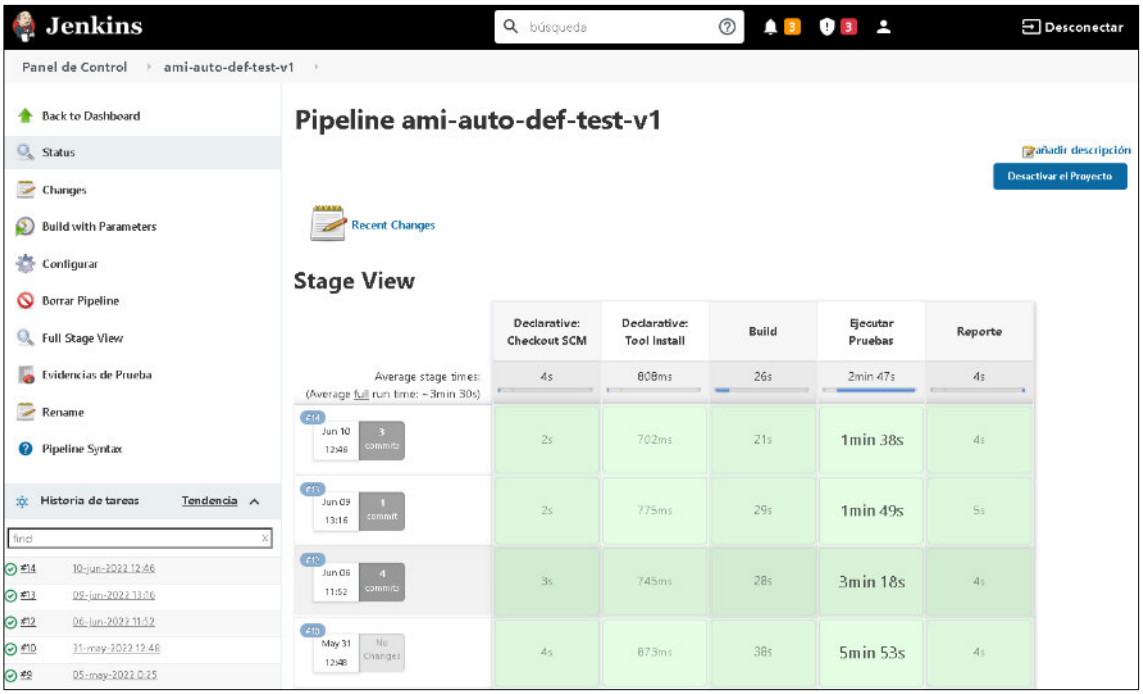
Figura 81: Ejecución de un escenario de pruebas en Jenkins



Fuente: Jenkins del proyecto de automatización

En la figura 82, se muestra las ejecuciones de las pruebas automatizadas desde la herramienta Jenkins.

Figura 82: Visualización de escenarios de pruebas ejecutados desde Jenkins

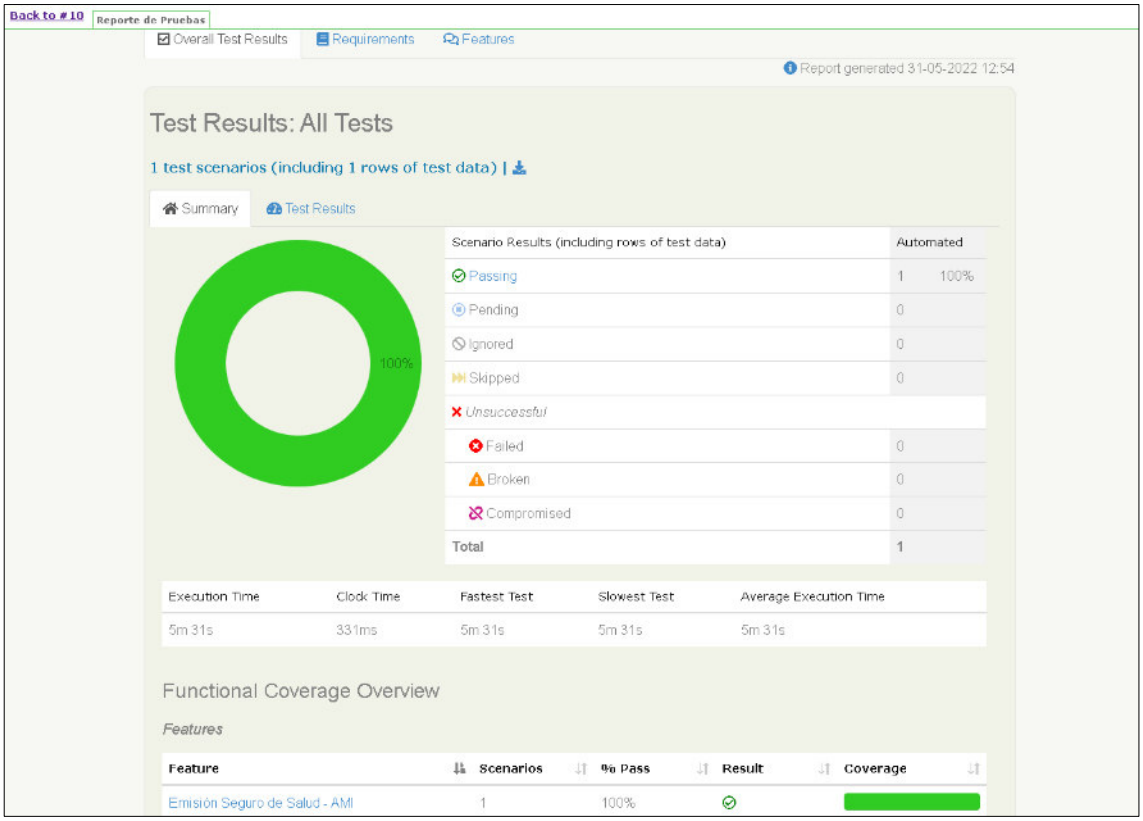


Fuente: Jenkins del proyecto de automatización

- Una vez finalizado la ejecución, ingresar a visualizar el reporte generado por Serenity, las cuales tiene las siguientes secciones:

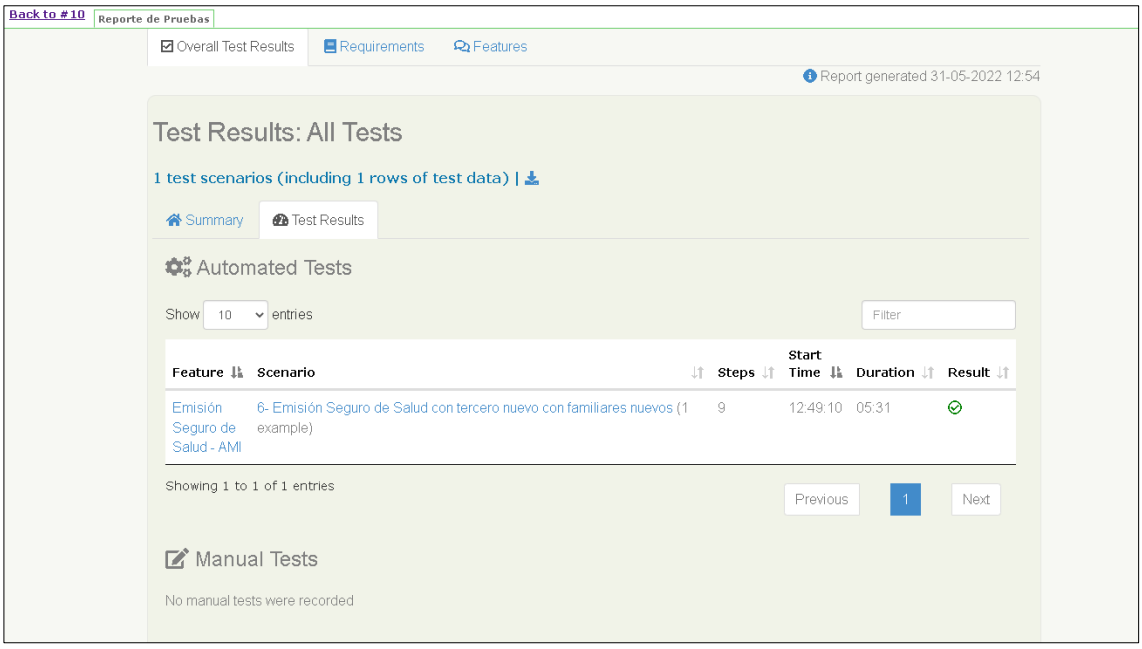
    - **Overall Test Results:** Donde se muestra los resultados generales de las pruebas, el feature ejecutado, el tiempo de ejecución, el grafico, la cantidad de pruebas que pasaron, fallaron, pendiente, etc (ver figura 83).
    - **Requirements:** Muestra la especificación del escenario ejecutado e indica la cantidad de pasos, fecha de inicio de la ejecución, tiempo de ejecución y el resultado.
    - **Features:** Muestra el detalle de los features del proyecto, la cantidad de escenarios ejecutados y el resultado (ver figura 85).
    - **Test Results:** En esta pestaña se muestra las pruebas de aceptación que se ejecutaron (ver figura 84).

Figura 83: Sección Overall Test Results en el reporte Serenity



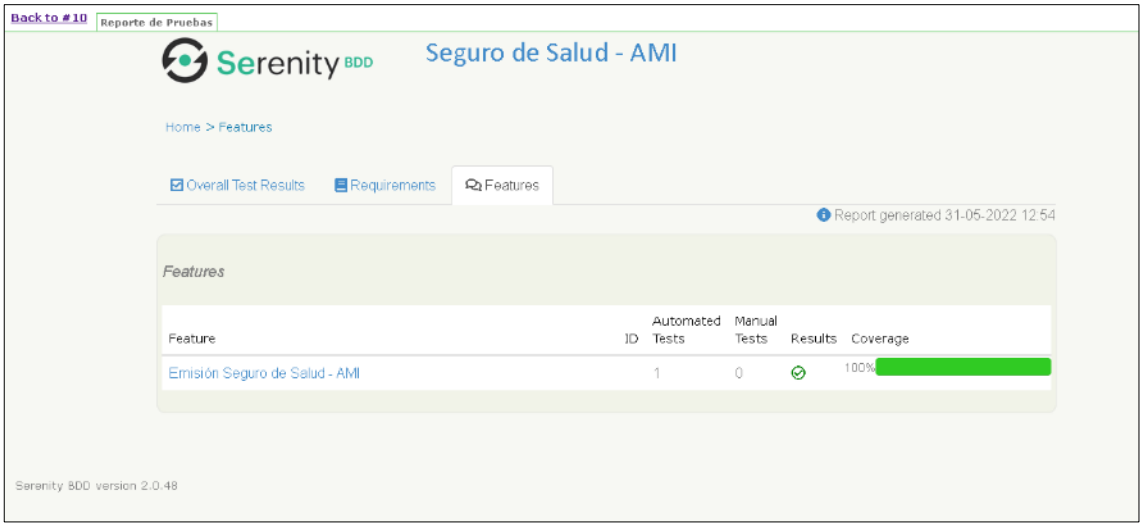
Fuente: Reporte generado del proyecto de automatización

Figura 84: Sección Test Results del reporte Serenity



Fuente: Reporte generado del proyecto de automatización

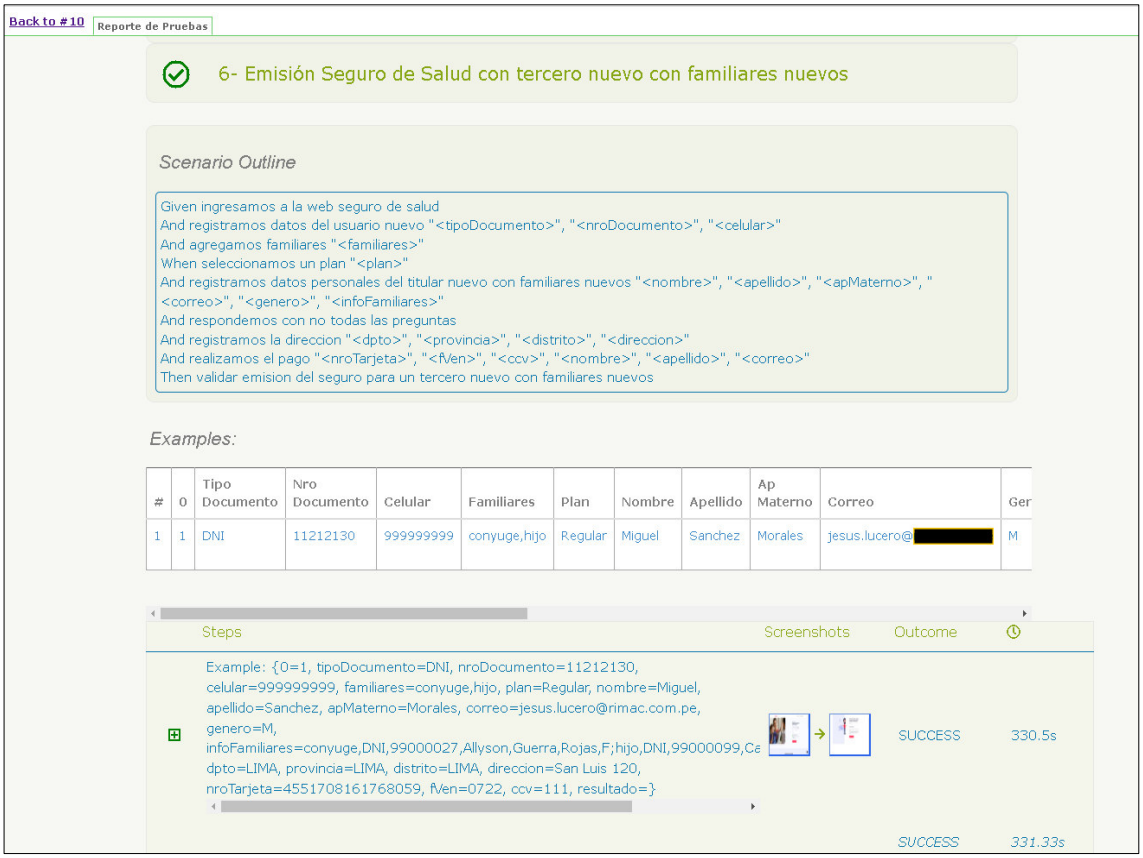
Figura 85: Sección features del reporte Serenity



Fuente: Reporte generado del proyecto de automatización

- También muestra la información que se usa para la ejecución de la prueba y los paso a paso con sus respectivas evidencias (ver figura 86).

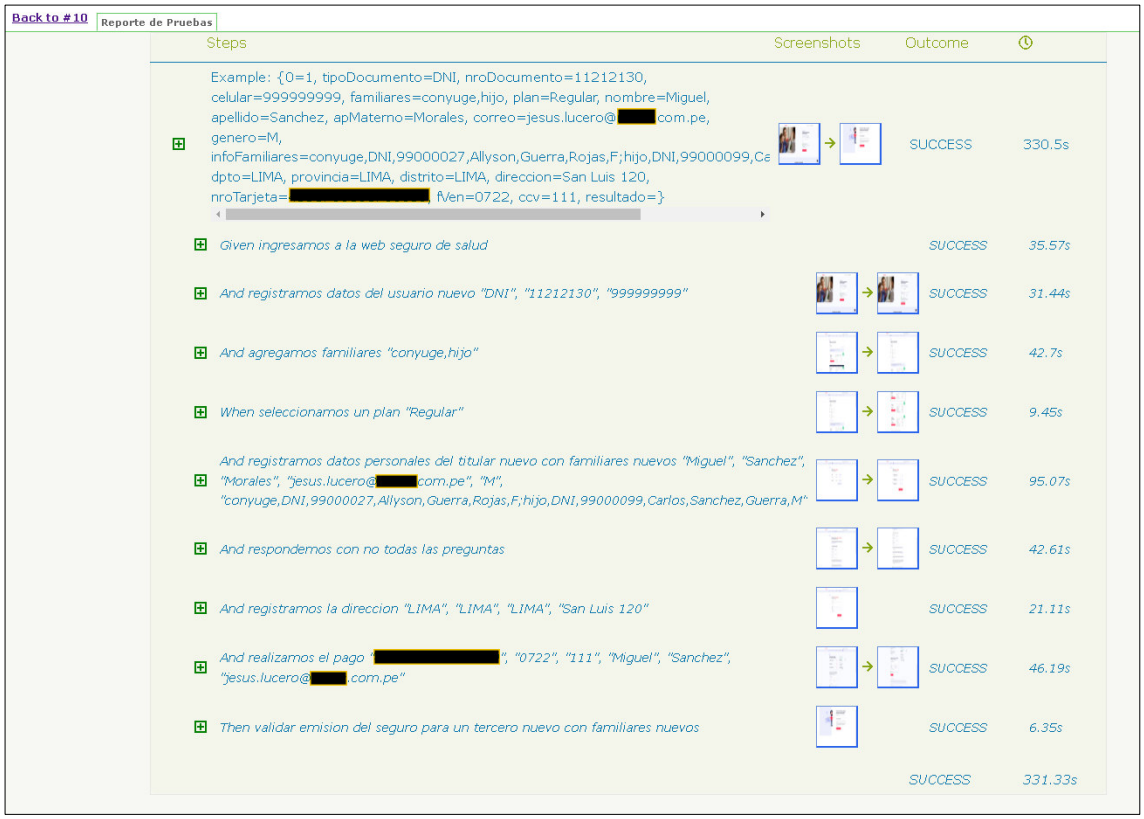
Figura 86: Visualización del escenario web ejecutado en el reporte Serenity



Fuente: Reporte generado del proyecto de automatización

En la figura 87, se muestra los paso a paso de la ejecución de la prueba automatizada.

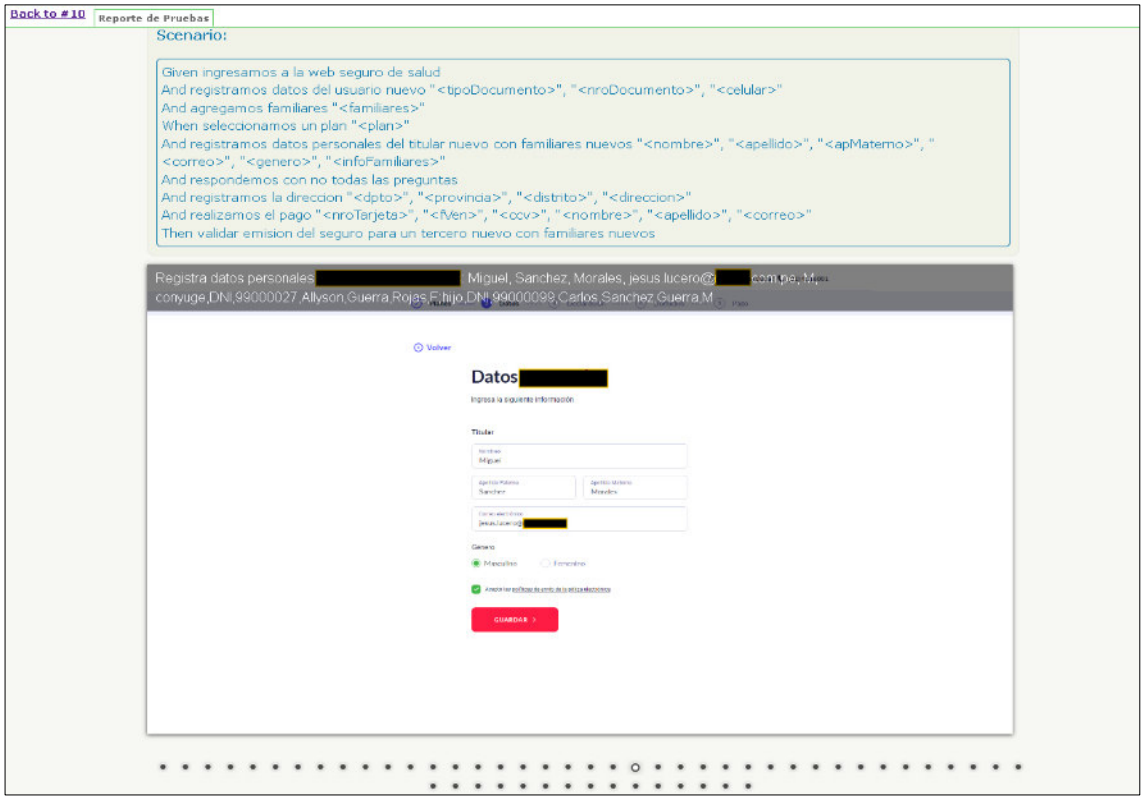
Figura 87: Visualización de los paso a paso del escenario web con las evidencias generadas



Fuente: Reporte generado del proyecto de automatización

En la figura 88, se muestra las capturas de pantalla de las pruebas.

Figura 88: Visualización de las videncias web generadas a detalle



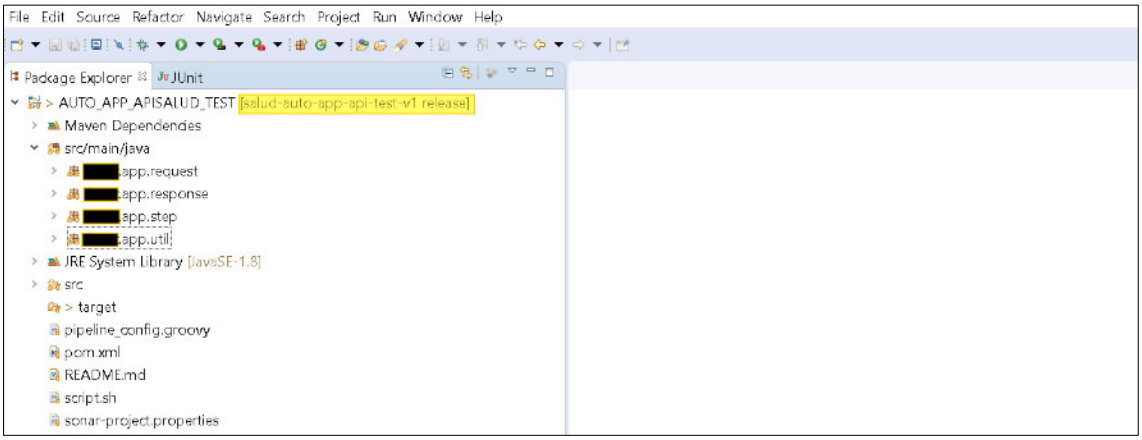
Fuente: Reporte generado del proyecto de automatización

Desarrollo de scripts de pruebas automatizadas para APIs:

Ahora crear el proyecto para APIs salud-auto-app-api-test con la estructura definida (ver figura 89).

- **Proyecto:** salud-auto-app-api-test

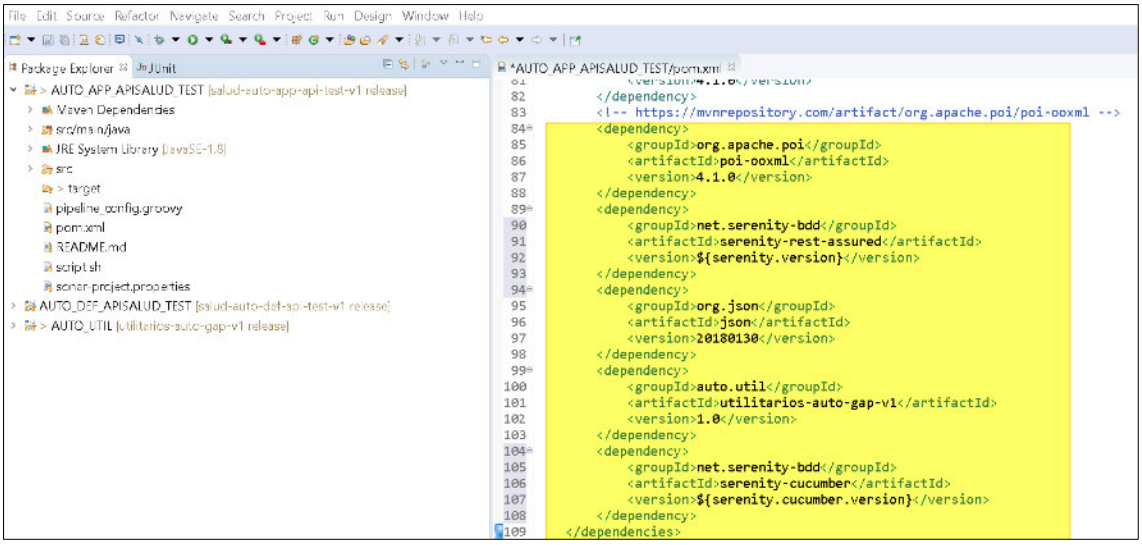
Figura 89: Estructura de carpetas del proyecto salud-auto-app-api-test



Fuente: Estructura de carpetas del proyecto de automatización

- Luego se debe agregar las dependencias del proyecto utilitario, Serenity Rest Assured, Cucumber, org.json en el archivo pom.xml (ver figura 90).

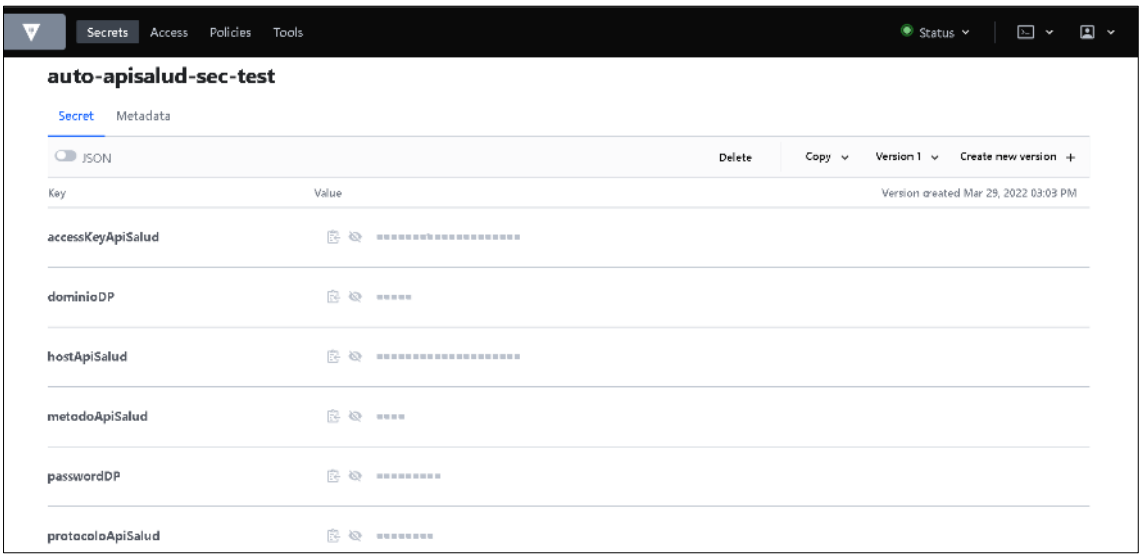
Figura 90: Dependencias en el archivo pom.xml del proyecto salud-auto-app-api-test



Fuente: Código fuente del framework de automatización

- Luego se debe crear y asignar las credenciales de las APIs a las variables en Vault (ver figura 91).

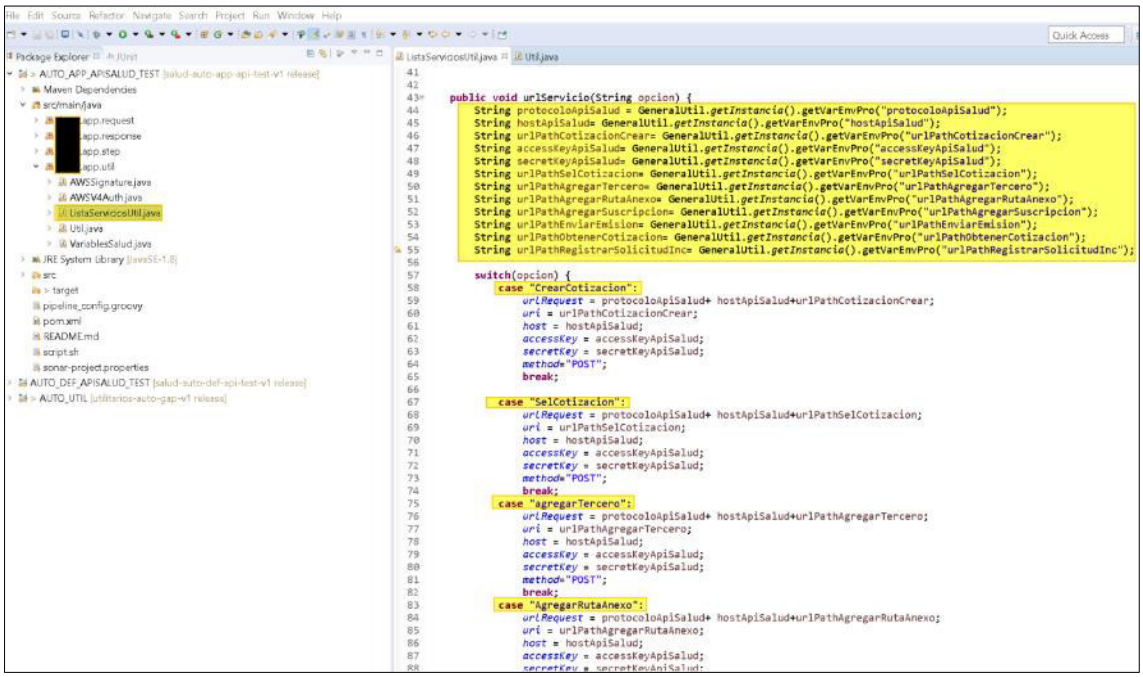
Figura 91: Configuración de las variables en Vault



Fuente: Herramienta Vault del proyecto de automatización

- Ahora en esta clase se debe consumir las credenciales y URL de las APIs que se van a utilizar desde Vault con métodos definidos en el utilitario (ver figura 92).

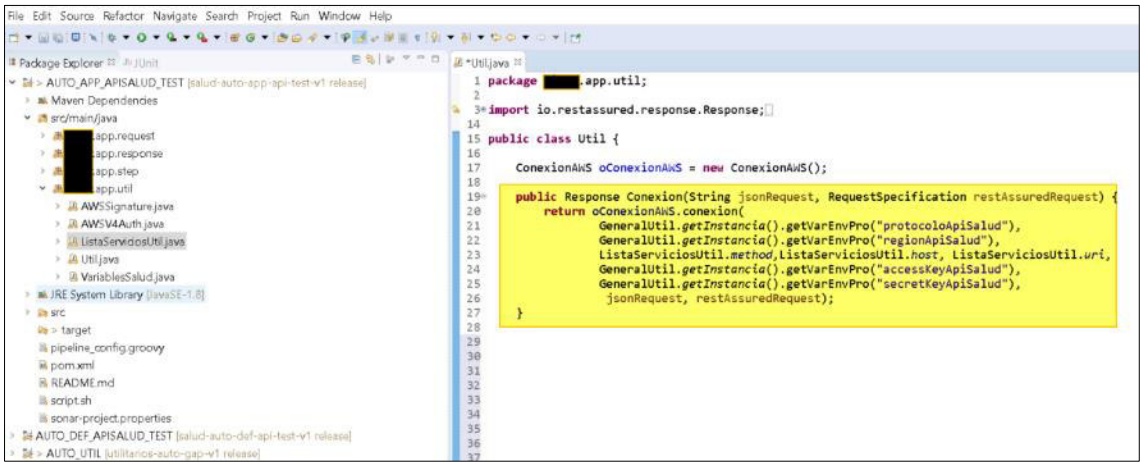
Figura 92: Mapeo de credenciales desde Vault



Fuente: Código fuente del framework de automatización

- Luego se debe realizar la conexión a los servicios utilizando los métodos definidos en el utilitario, donde le enviamos como parámetros las credenciales y el request del servicio (ver figura 93).

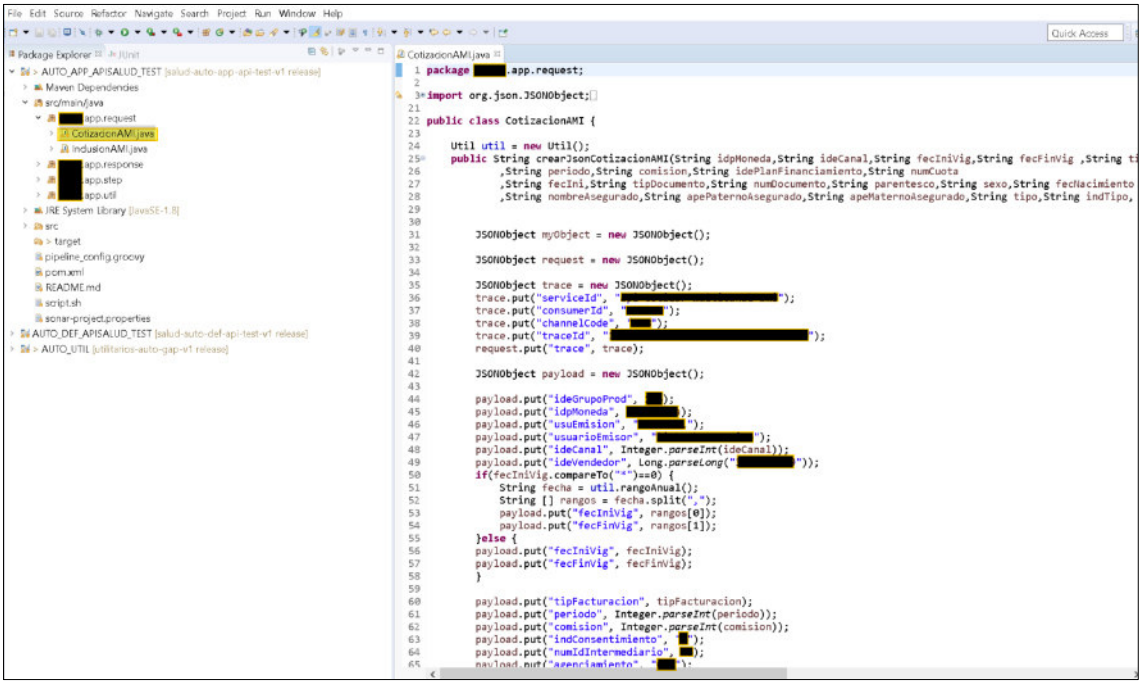
Figura 93: Conexión con las APIs y envío de request



Fuente: Código fuente del framework de automatización

- Una vez realizado la conexión a las APIs, crear los request de las peticiones (ver figura 94).

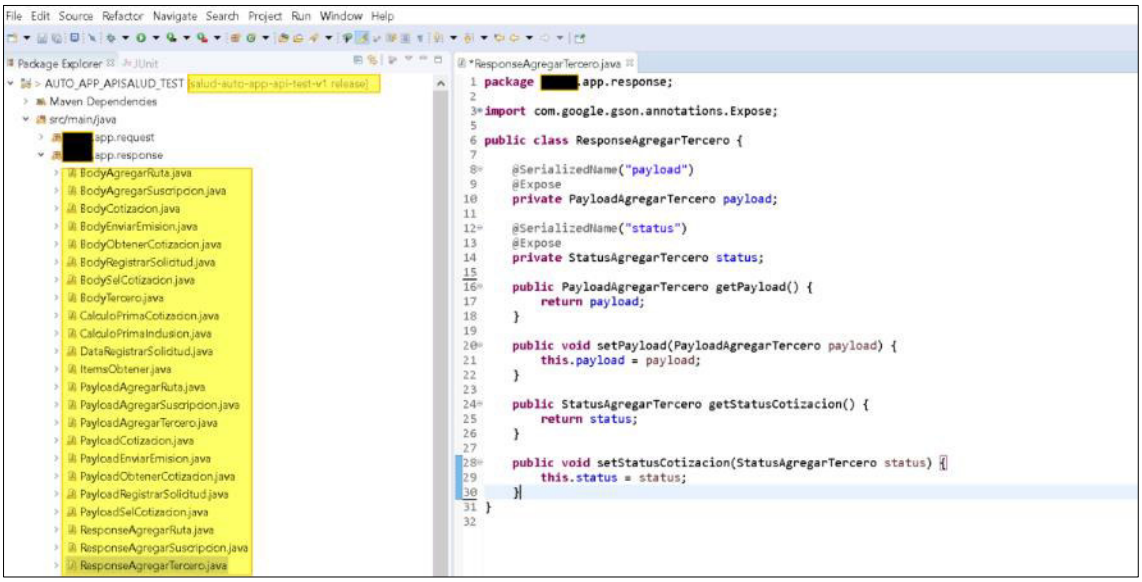
Figura 94: Generación de request de las peticiones



Fuente: Código fuente del framework de automatización

- Luego se debe crear objetos propios para guardar el cuerpo de la respuesta y posteriormente usarlo para las validaciones (ver figura 95).

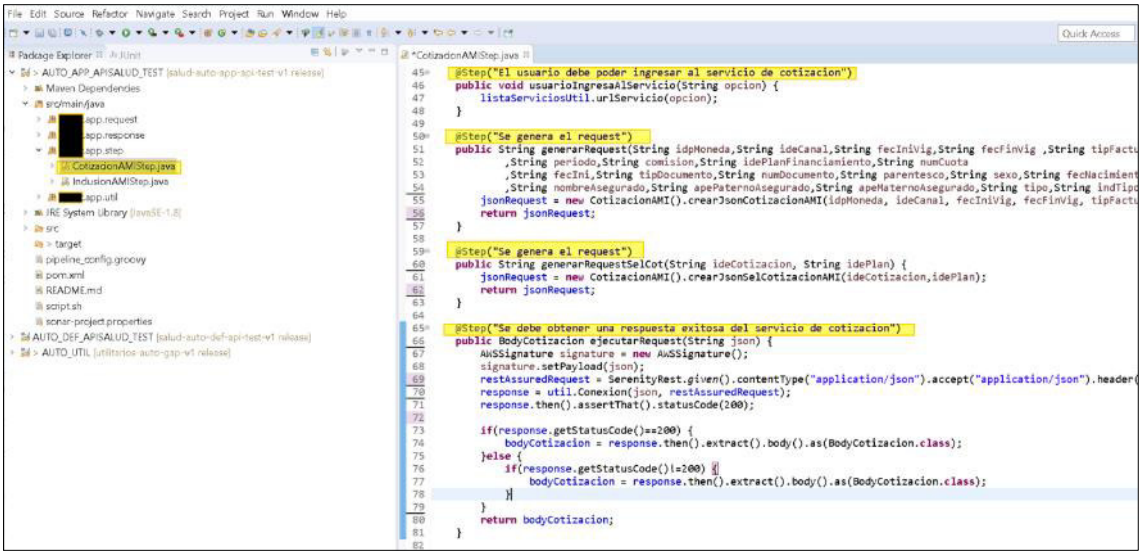
Figura 95: Objetos para guardar respuestas de las APIs



Fuente: Código fuente del framework de automatización

- Ahora se debe crear clases en el paquete Step donde se realizará los paso a paso o acciones que definen un conjunto de interacciones para realizar la ejecución de las pruebas y las validaciones, luego esta clase será instanciada por los definitions (ver figura 96).

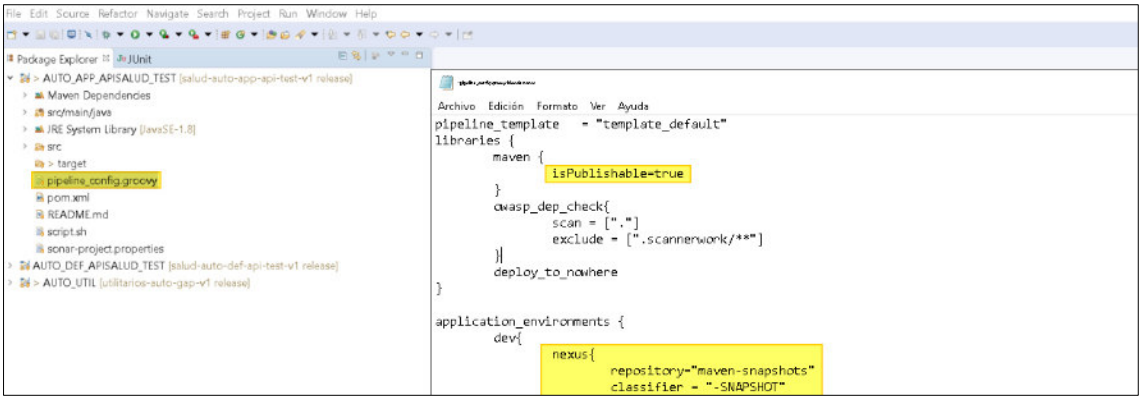
Figura 96: Clases Steps de una API



Fuente: Código fuente del framework de automatización

- Ahora se debe configurar el archivo pipeline_config.groovy, que va a permitir generar la librería de este proyecto y publicarlo en Nexus para ser consumido por el proyecto salud-auto-def-api-test (ver figura 97).

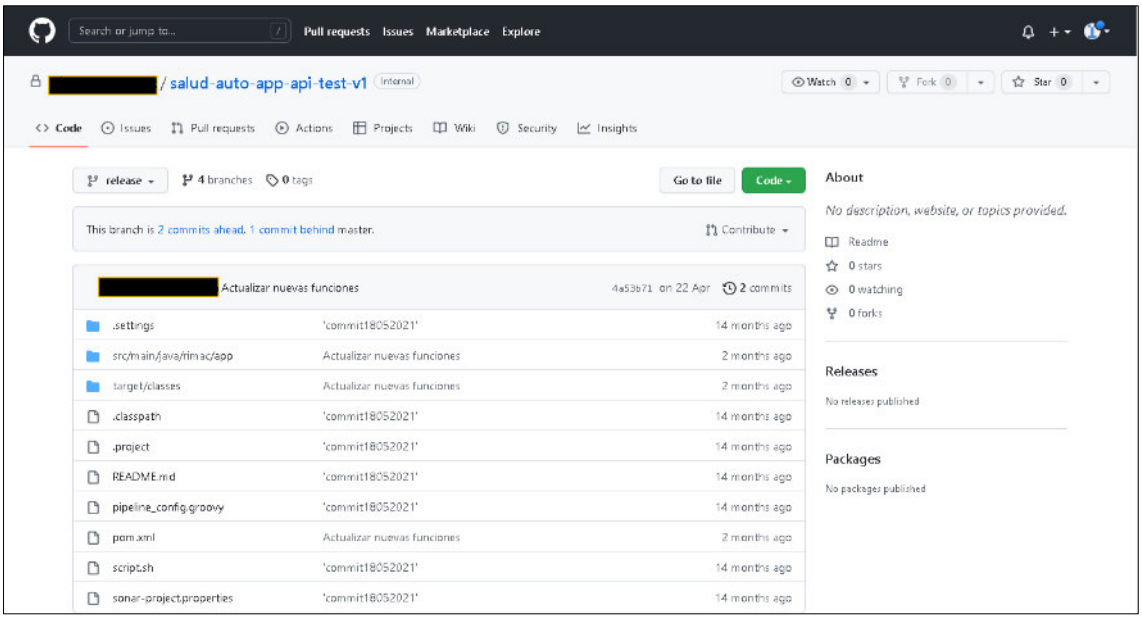
Figura 97: Configuración del archivo pipeline_config.groovy para APIs



Fuente: Código fuente del framework de automatización

- Luego se debe versionar el proyecto en GitHub (ver figura 98).

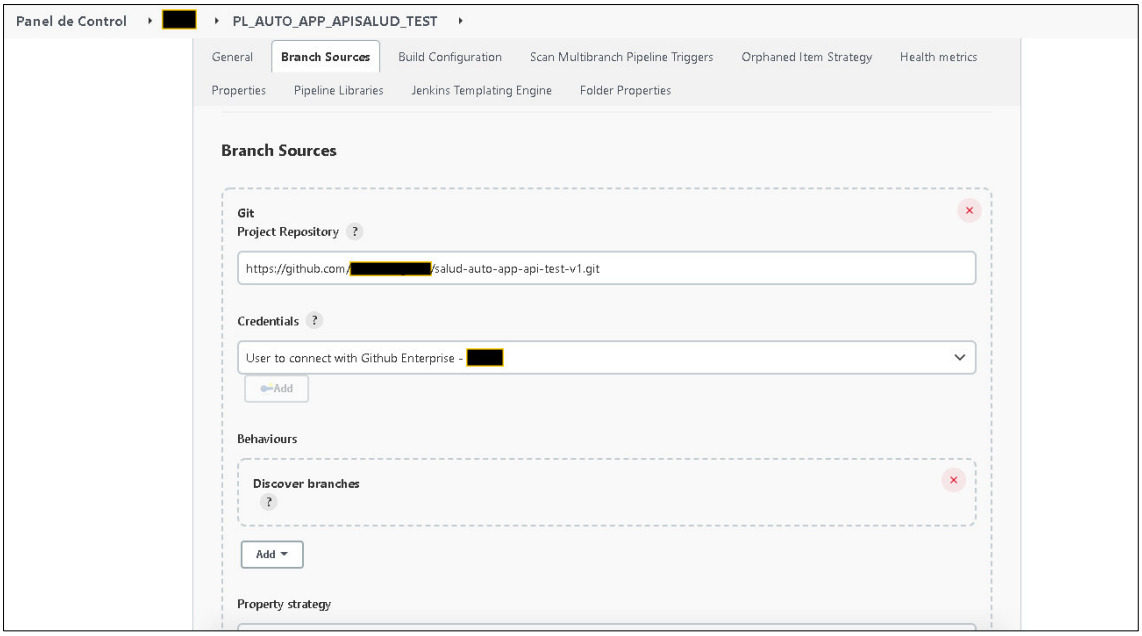
Figura 98: Versionamiento del proyecto salud-auto-app-api-test en GitHub



Fuente: Repositorio del proyecto de automatización

- Después de realizar el versionamiento ingresar a Jenkins y crear un pipeline multibranch y realizar las configuraciones para que consuma el proyecto desde GitHub y genere la librería y lo publique en Nexus, cada vez que se realice un cambio en el proyecto se generara una nueva versión de la librería (ver figura 99).

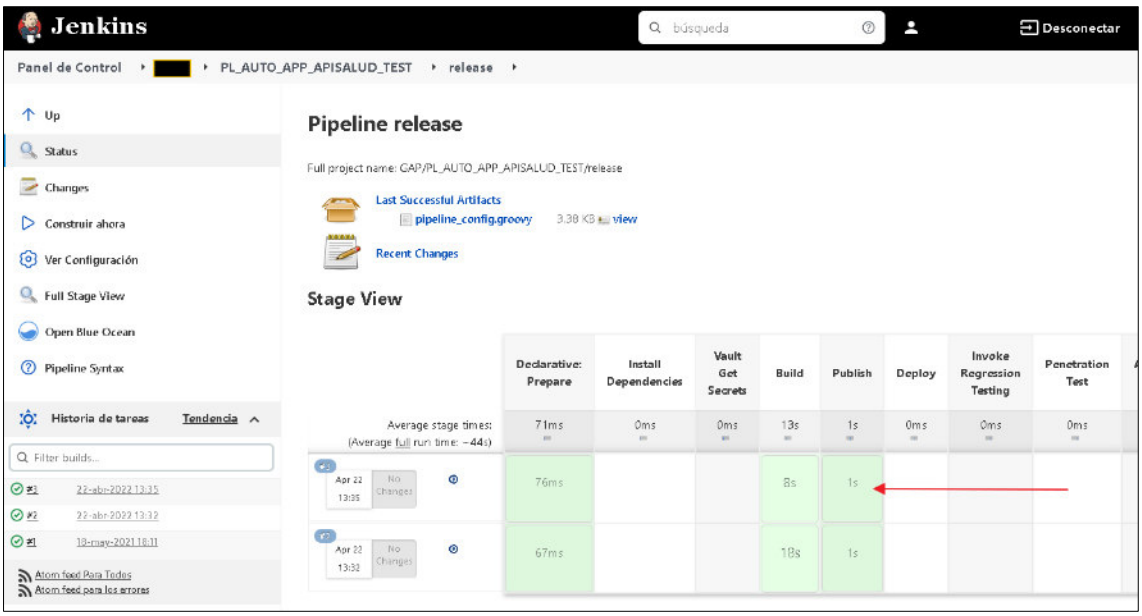
Figura 99: Configuración del pipeline multibranch para APIs



Fuente: Jenkins del proyecto de automatización

- Luego de configurar, se valida la ejecución del pipeline y la publicación de la librería del proyecto salud-auto-app-api-test en Nexus (ver figura 100).

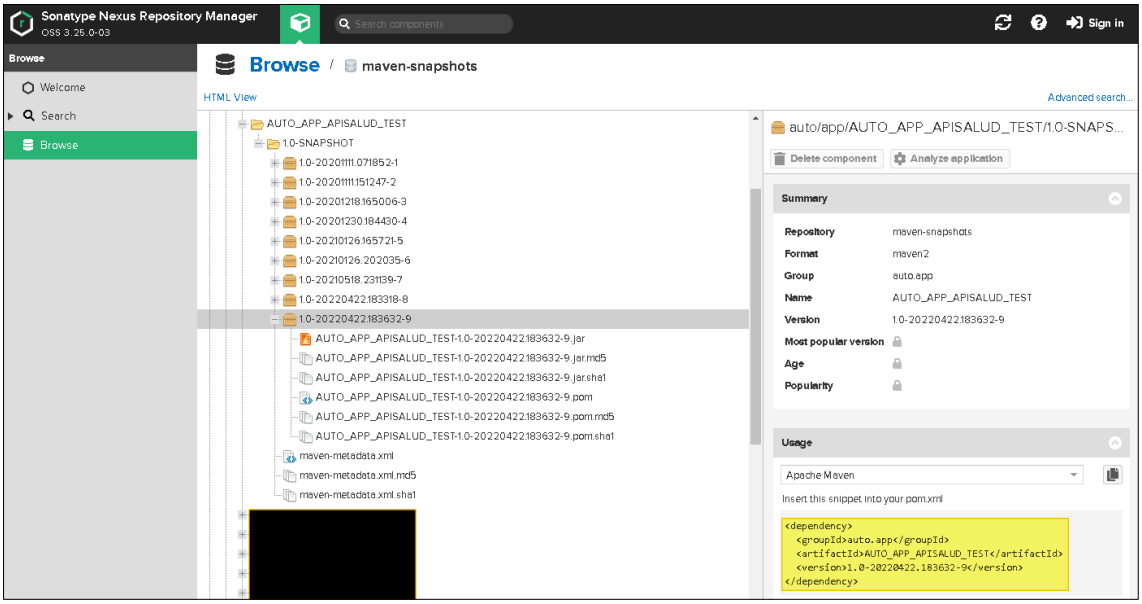
Figura 100: Ejecución pipeline del proyecto salud-auto-app-api-test y publicación en Nexus



Fuente: Jenkins del proyecto de automatización

- Luego ingresar a Nexus y revisar la publicación de la librería del proyecto salud-auto-app-api-test (ver figura 101).

Figura 101: Publicación de la librería salud-auto-app-api-test en Nexus

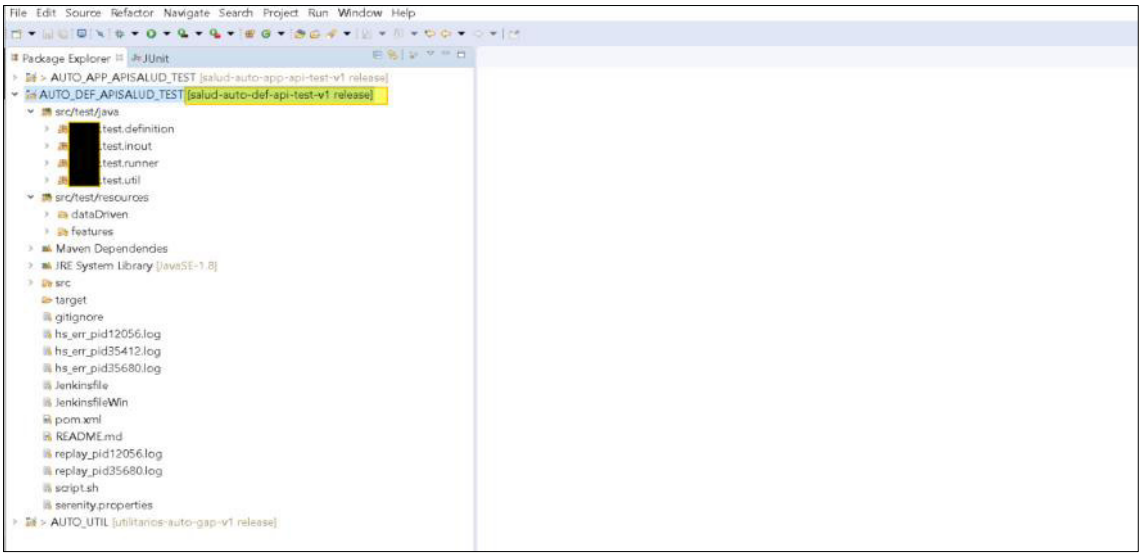


Fuente: Nexus del proyecto de automatización

Ahora se debe crear el proyecto salud-auto-def-api-test con la estructura de carpetas definida (ver figura 102).

- **Proyecto:** salud-auto-def-api-test

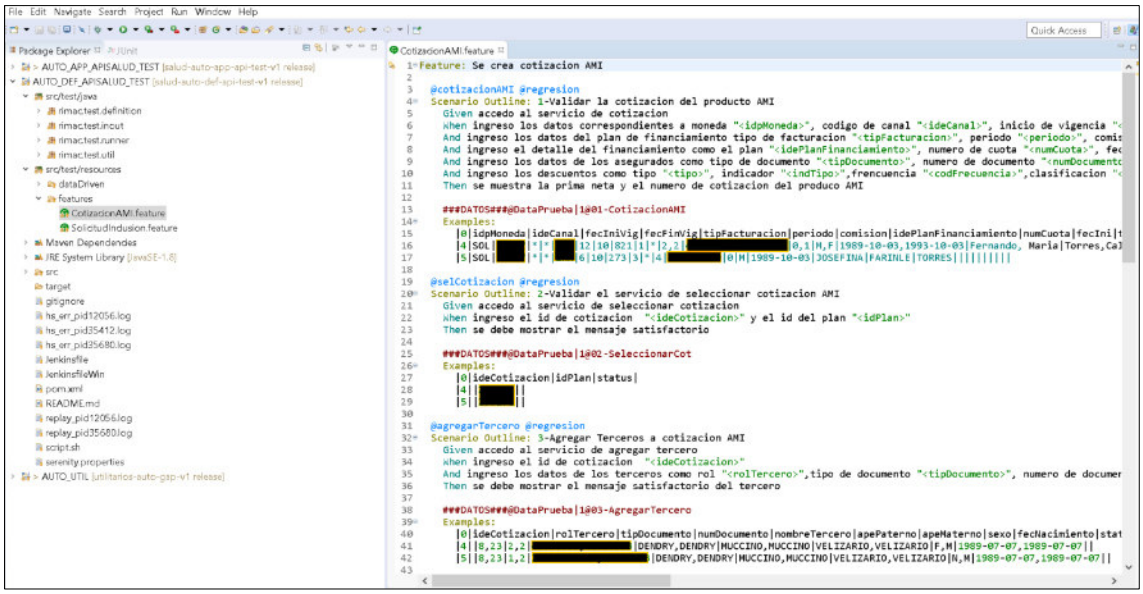
Figura 102: Estructura de carpetas del proyecto salud-auto-def-api-test



Fuente: Estructura de carpetas del proyecto de automatización

- Ahora se debe crear el feature y escribir los escenarios de pruebas en lenguaje Gherkin, estos pasos pueden ser reutilizables en los demás escenarios (ver figura 103).

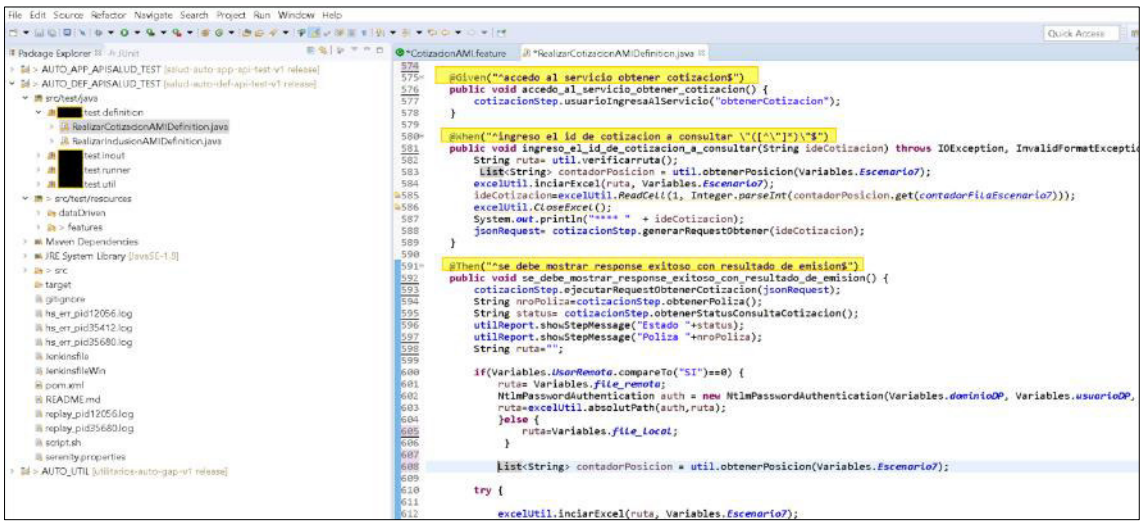
Figura 103: Creación de escenarios para APIs en lenguaje Gherkin



Fuente: Código fuente del framework de automatización

- Ahora se debe ejecutar el feature para generar los métodos que serán implementados en los definitions instanciando las clases Steps para realizar los paso a paso para las ejecuciones de las pruebas (ver figura 104).

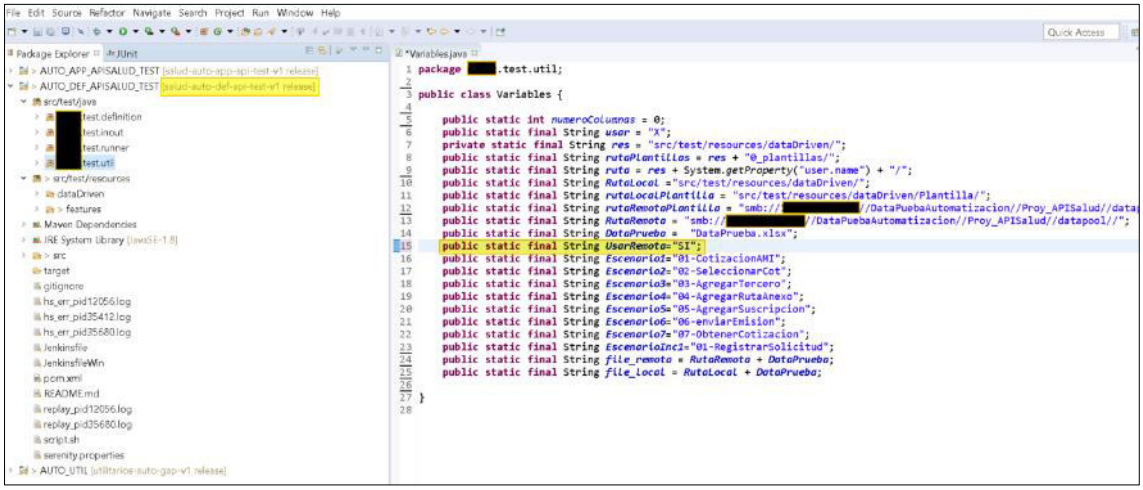
Figura 104: Implementación de los definitions en APIs



Fuente: Código fuente del framework de automatización

- Luego se debe configurar la clase variables para consumir los datos de prueba desde una ruta compartida para la ejecución desde Jenkins (ver figura 105).

Figura 105: Configuración para consumir los datos de pruebas desde una ruta compartida para APIs

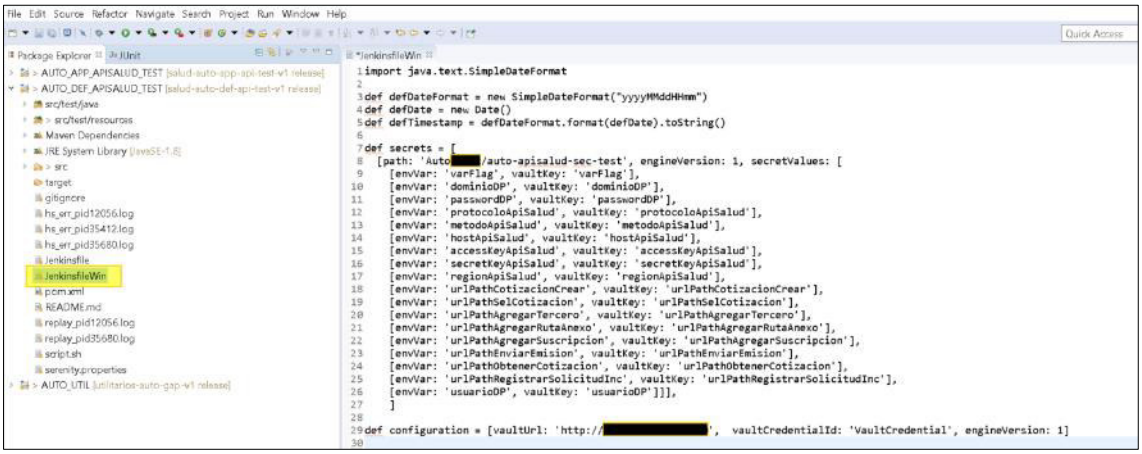


Fuente: Código fuente del framework de automatización

- Luego se debe configurar el archivo Jenkinsfile para la ejecución desde Jenkins.

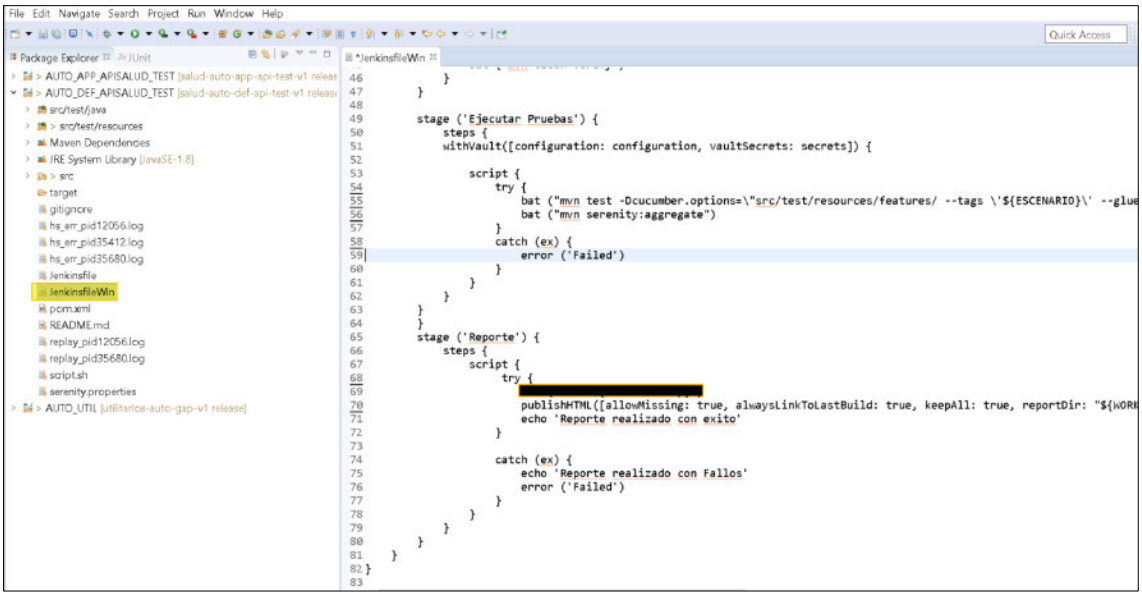
    - Donde se configura las variables para consumir las credenciales desde Vault (ver figura 106).
    - Donde se configura los Stage para la ejecución de las pruebas (ver figura 107).

Figura 106: Configuración Jenkinsfile para consumir las credenciales desde Vault



Fuente: Código fuente del framework de automatización

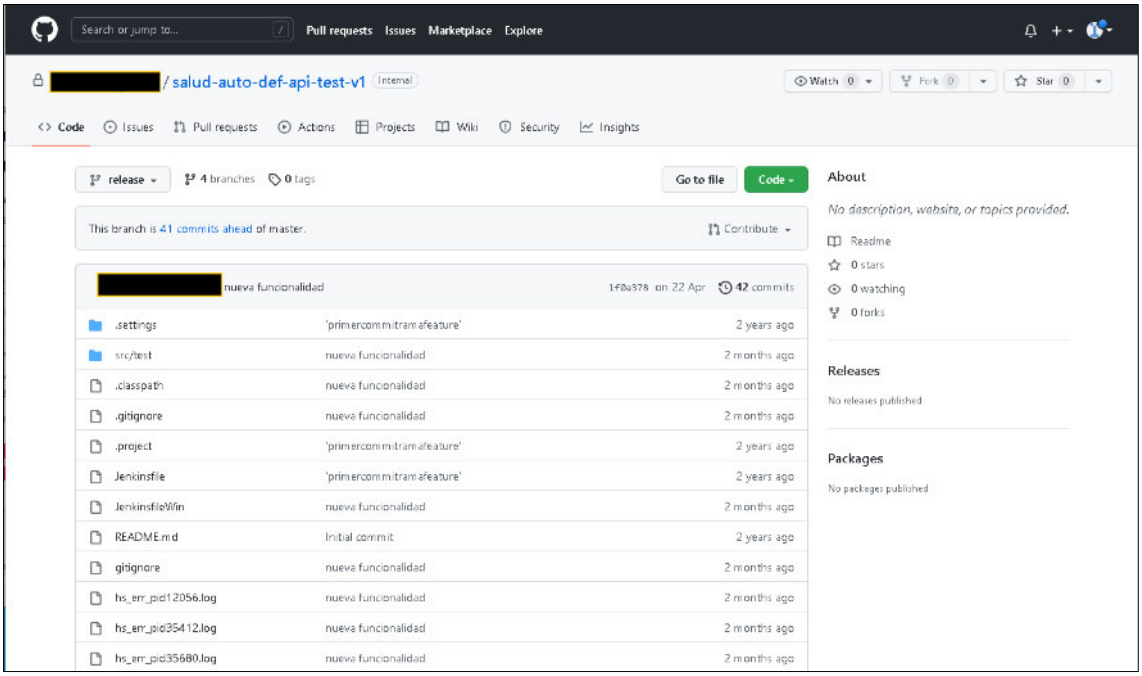
Figura 107: Configuración del archivo Jenkinsfile del proyecto de APIs



Fuente: Código fuente del framework de automatización

- Ahora se debe realizar el versionamiento del proyecto en GitHub (ver figura 108).

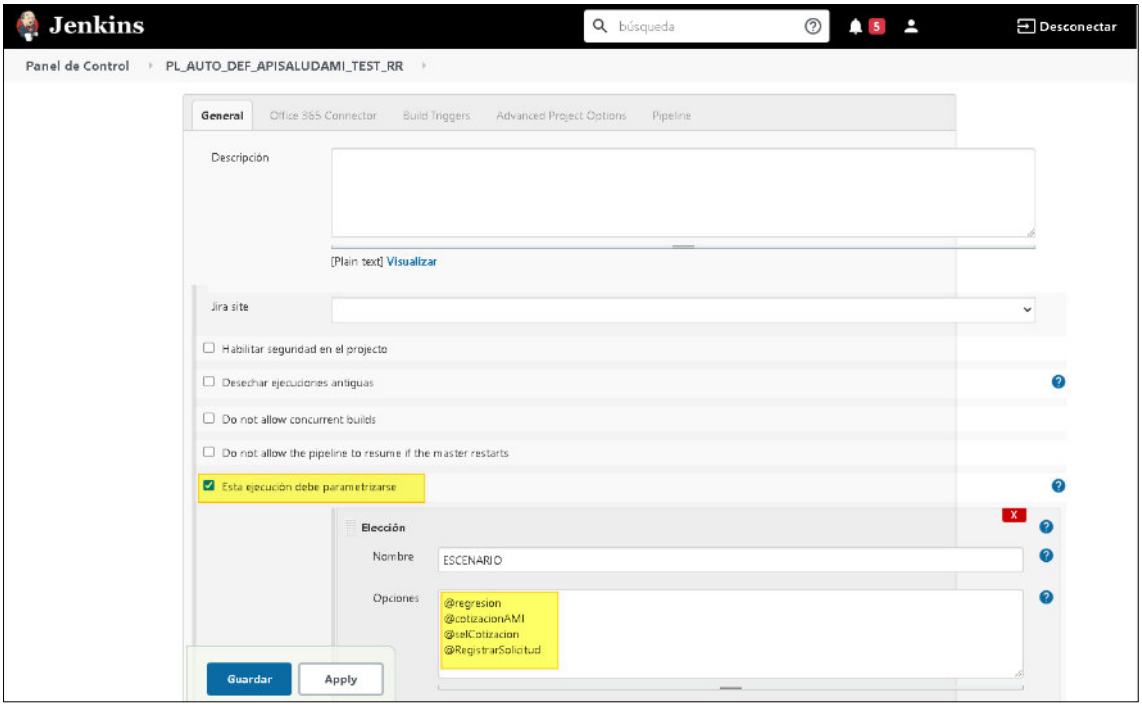
Figura 108: Versionamiento del proyecto salud-auto-def-api-test



Fuente: Repositorio del proyecto de automatización

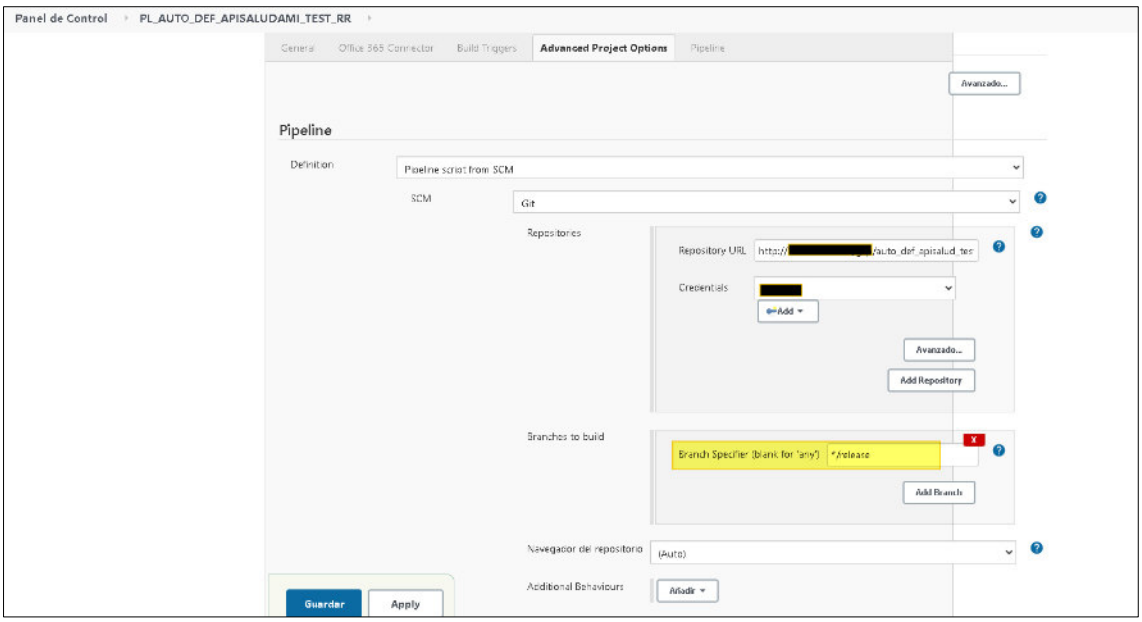
- Una vez versionado el proyecto, ingresar a Jenkins y crear un pipeline para realizar las ejecuciones de las pruebas automatizadas.
    - Configurar para ejecutar de forma parametrizable (ver figura 109).
    - Configurar la ruta y rama del proyecto desde GitHub (ver figura 110).
    - Configurar el archivo Jenkinsfile.

Figura 109: Configuración de escenarios parametrizables del proyecto APIs



Fuente: Jenkins del proyecto de automatización

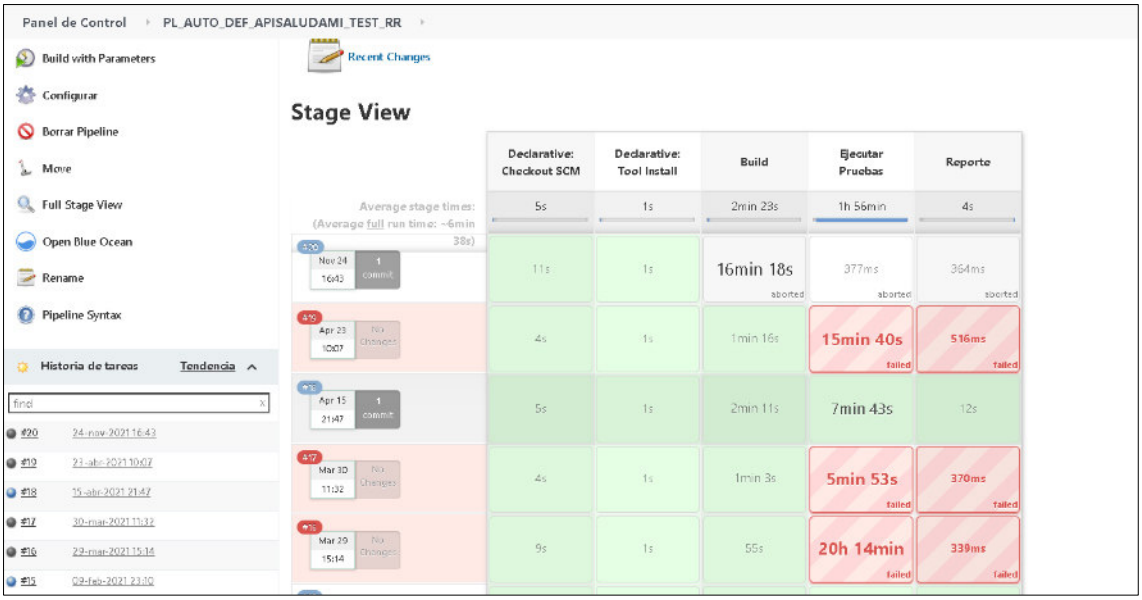
Figura 110: Configuración rutas y rama de GitHub del proyecto APIs



Fuente: Jenkins del proyecto de automatización

- Luego de terminar la configuración del pipeline, realizar las ejecuciones de los escenarios de pruebas automatizados (ver figura 111).

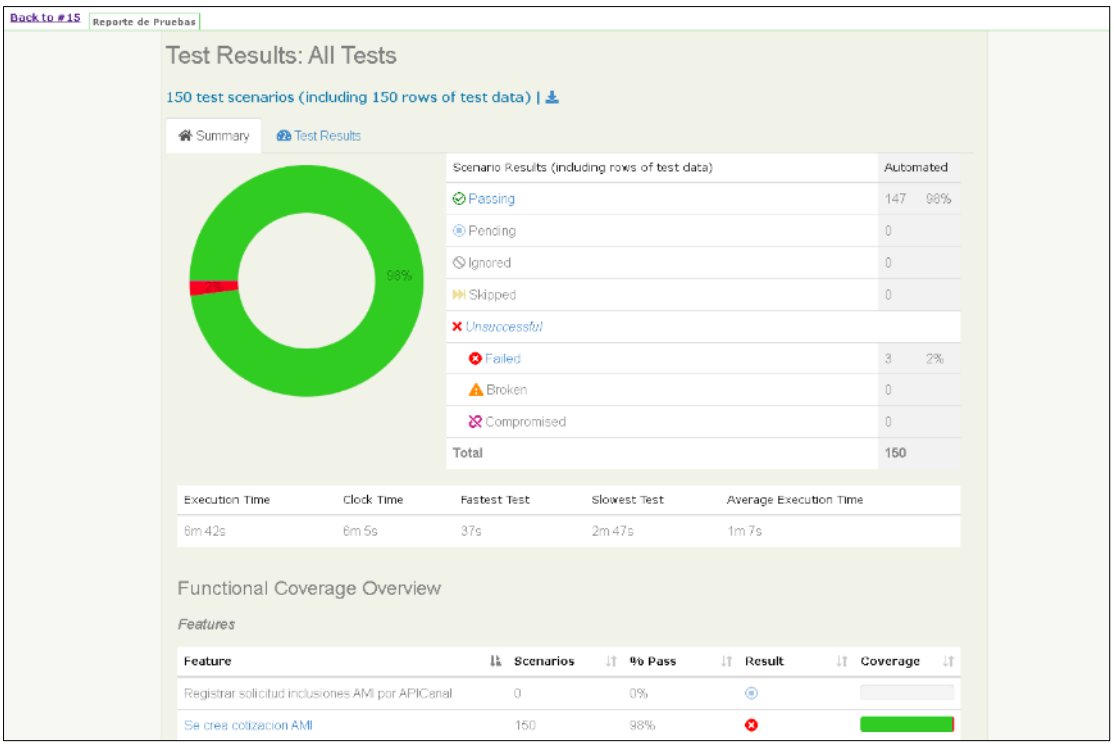
Figura 111: Ejecución de las pruebas automatizadas de APIs desde Jenkins



Fuente: Jenkins proyecto de automatización

- Una vez que se termine las ejecuciones de los escenarios de pruebas automatizados, se genera el reporte cuando la ejecución termino sin errores (ver figura 112).

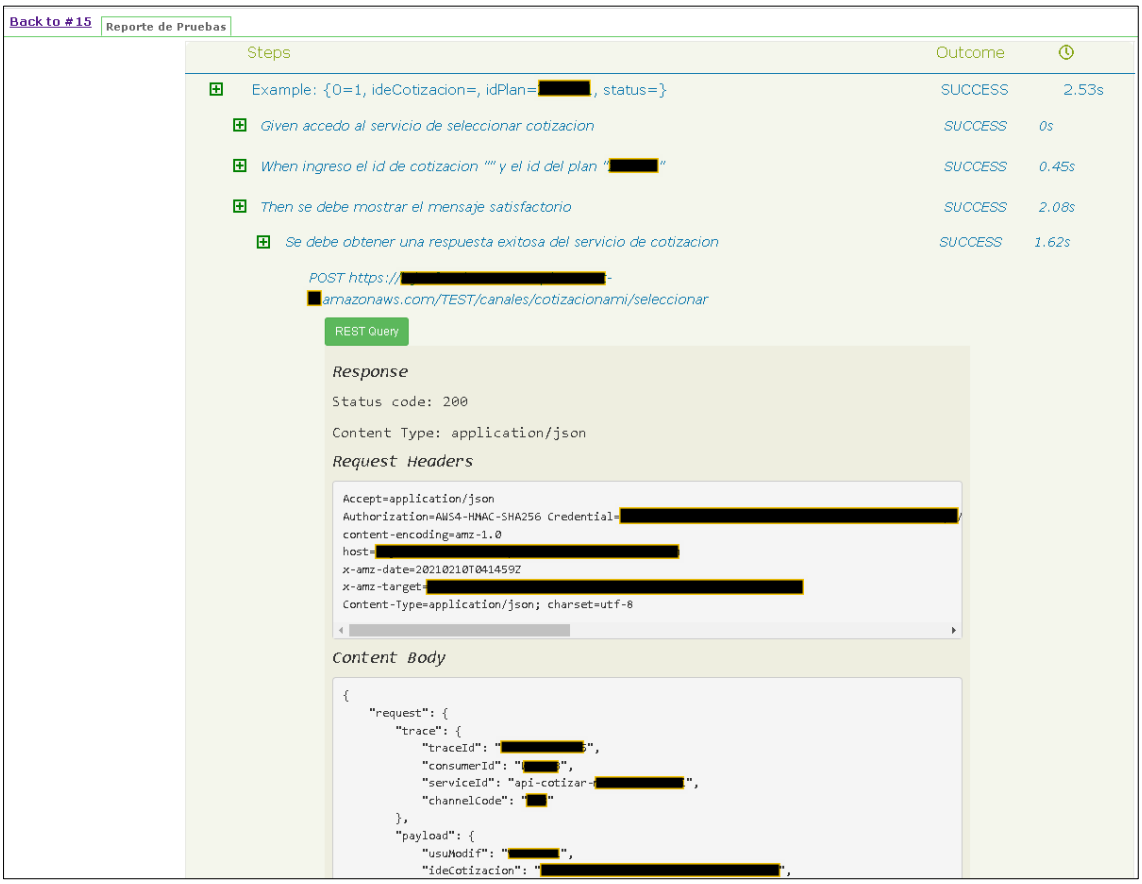
Figura 112: Reporte generado de una prueba exitosa de APIs



Fuente: Reporte generado del proyecto de automatización

En la figura 113, se muestra las evidencias generadas de la ejecución de pruebas automatizadas de APIs.

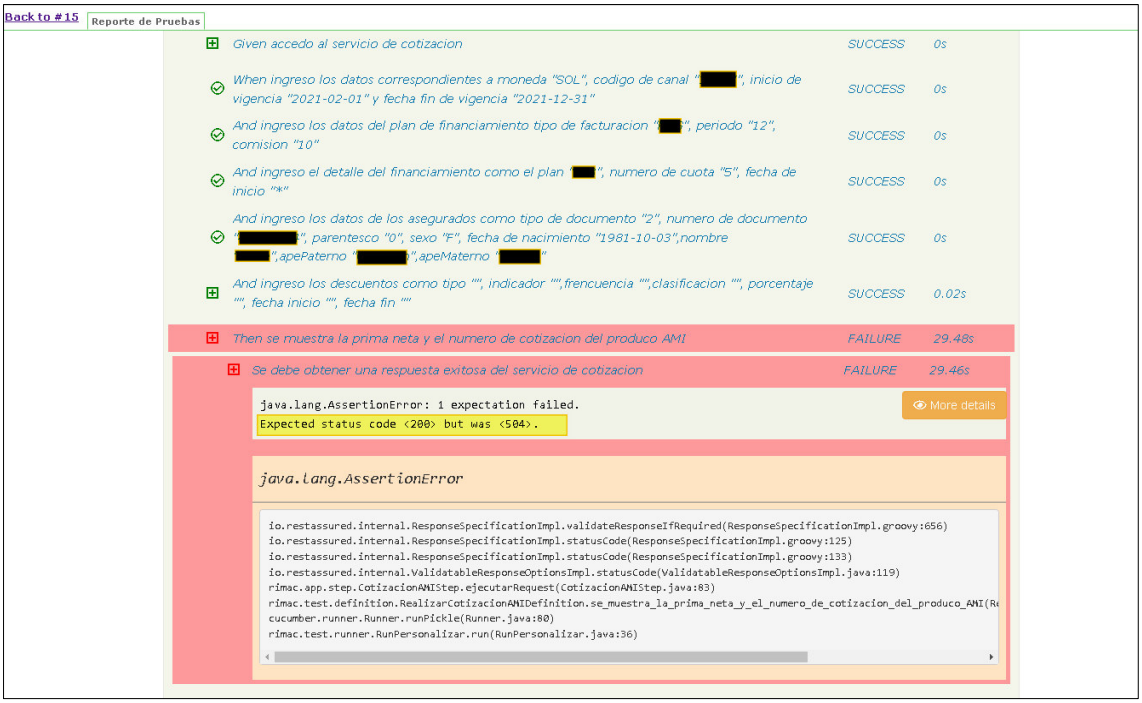
Figura 113: Visualización de las evidencias del reporte generado en APIs



Fuente: Reporte generado del proyecto de automatización

- También se genera el reporte cuando un caso ejecutado falla, donde se muestra el error y su detalle (ver figura 114).

Figura 114: Visualización del reporte cuando una prueba de APIs falla



Fuente: Reporte generado del proyecto de automatización

3.3. Evaluación

Se detallará la evaluación económica y los beneficios del proyecto.

3.3.1. Evaluación económica / evaluación costo – beneficio

En cuanto a los costos reales del proyecto, al ser una información de carácter confidencial, estos no pueden ser mostradas en este informe. Sin embargo, en la tabla 12 se detallan costos aproximados. En cuanto a la duración del proyecto, este fue de 6 meses.

Tabla 12: Costo estimado del proyecto

Personal requerido	Costo por mes (S/.)	# de meses	Costo total (S/.)
Analista de pruebas funcionales 1	4500	6	27000
Analista de pruebas funcionales 2	4500	6	27000
Ingeniero de automatización de pruebas	6000	6	36000
		Total	90000

Como se puede apreciar, la inversión total en capital humano ascendió a la suma de 90,000.00 soles.

Beneficios obtenidos para la organización:

- Debido a que las pruebas de regresión se realizan de forma automatizada ya no es necesario tener a los analistas funcionales dedicados a realizar estas pruebas ya que son ejecutadas a demanda o programado desde la herramienta Jenkins para acelerar todo el proceso de entrega donde nos genera el reporte con las evidencias de los paso a paso de las pruebas.
- Dado que la automatización simplifica las tareas rutinarias al realizar más ejecuciones de pruebas en menos tiempo, mitigando con ello los riesgos ya que los incidentes son reportados más rápido. En consecuencia, mejora la calidad del software mediante la detección temprana de defectos.
- Reducción del tiempo de elaboración de los reportes de las evidencias de pruebas ya que se generan de forma automática en cada ejecución.

- Las herramientas utilizadas para la realización del proyecto se podían obtener de forma gratuita, como son los softwares Eclipse, Maven, JDK, GIT, Selenium, Serenity, Cucumber. Para el caso de Jenkins, GitHub la compañía ya contaba con estos productos, por lo que no significo un gasto adicional y aportaron valor a la organización facilitando la mejor en el proceso de calidad.

CAPÍTULO IV – REFLEXIÓN CRÍTICA DE LA EXPERIENCIA

- La participación del autor en el proyecto fue con el cargo de Ingeniero de automatización de pruebas, la cual conllevo en realizar una serie de actividades para poder garantizar la calidad de los proyectos en cada Sprint, donde el autor realizó la implementación de automatización de los flujos para las pruebas de regresión de aplicaciones Web y APIs y participó en definir los lineamientos en el equipo de gobierno de automatización de pruebas.
- Este proyecto fue muy enriquecedor para el autor, ya que se usaron diversas herramientas, enfoques, patrones, buenas prácticas que le permitió consolidar y profundizar sus conocimientos asociados a la automatización de pruebas.
- Al autor le permitió brindar asesorías y capacitaciones sobre el framework y los lineamientos de automatización a diferentes equipos de la compañía para que empiecen aplicar en sus proyectos.
- Al tener implementado un proyecto utilitario con métodos genéricos para reutilizar en las automatizaciones Web y APIs permitió que otros equipos puedan realizar las automatizaciones mucho más rápido.
- Los analistas de pruebas funcionales pueden diseñar los escenarios de pruebas en lenguaje Gherkin ya que usa un lenguaje natural y no necesariamente deben saber de código.
- La automatización de pruebas requiere una inversión de tiempo y esfuerzo a corto plazo para conseguir resultados a largo plazo.
- La disponibilidad de ejecución de pruebas, ya que es posible ejecutar un gran número de escenarios de pruebas en un corto periodo de tiempo; donde las mismas pueden ser ejecutadas durante las 24 horas, los 7 días de la semana.

CAPÍTULO V – CONCLUSIONES Y RECOMENCACIONES

5.1. Conclusiones

- Se mejoró el proceso de pruebas en el área de calidad en un entorno de trabajo ágil al agregar como parte de las actividades las automatizaciones de pruebas de regresión ya que se logró aumentar la rapidez de la ejecución de las pruebas en comparación a las pruebas manuales permitiendo reducir costos, mitigar el riesgo, encontrar incidentes con anticipación.
- Con la implementación y ejecución de estas pruebas automatizadas de regresión periódicamente durante el proceso de desarrollo se logró garantizar la calidad esperada de las aplicaciones Web y APIs ya que son capaces de probar una y otra vez el software para así determinar el comportamiento de este y no se vea afectado por una actualización, cambio o desarrollo, todo esto en muy poco tiempo y con mayor exactitud.
- Se logró definir los lineamientos, procesos y arquitectura del framework de automatización.
- Se logró aplicar los patrones, enfoques y buenas prácticas al implementar el framework de automatización.
- Se logró obtener reportes generados automáticamente con las evidencias de los paso a paso de las pruebas con las ejecuciones desde la herramienta Jenkins.
- Se logró incrementar el alcance de la cobertura de pruebas de regresión, gracias a la automatización.
- Se logró diseñar los escenarios de pruebas usando el enfoque BDD (Desarrollo guiado por el comportamiento) y Gherkin ya que permite una mejor comunicación entre todas las partes interesadas de un proyecto, incluso por los que no tienen conocimientos técnicos.

5.2. Recomendaciones

- Realizar capacitaciones de los lineamientos y del framework de automatización de pruebas a los analistas de pruebas funcionales que no tengan conocimiento y/o experiencia previa en el tema para la realización de los mantenimientos de los scripts y ejecución de las pruebas desde la herramienta Jenkins.
- Realizar la integración del framework con la herramienta de administración de tareas de proyectos Jira para su ejecución.
- Evaluar y aplicar Machine Learning (reconocimiento de imágenes y técnicas de aprendizaje supervisado) para mejorar el mapeo de los elementos de las aplicaciones web.
- Realizar mayor cantidad de automatizaciones de escenarios de APIs ya que es un nivel más bajo, es decir que no interactúa directamente con la interfaz de usuario y nos va a permitir prevenir una mayor cantidad de bugs.
- Con la finalidad de realizar la automatización de pruebas de manera más eficiente, proponer como lineamiento al equipo de desarrollo que deben poner valores estáticos a las siguientes propiedades de los elementos web (id, name).

5.3. Fuentes de información

Crespo, A. (11 de Enero de 2018). *Automatización de pruebas*. Obtenido de Excentia.es.: https://www.excentia.es/automatizacion-de-pruebas

Dadgar, A. (2000). *HashiCorp Vault*. Obtenido de https://www.vaultproject.io/

Fagua Barrera, F. (30 de 11 de 2015). *Patrón Singleton.* Obtenido de https://repositorio.konradlorenz.edu.co/handle/001/3977

Furguson, J. (2014). *The Serenity BDD Book v2.4.48.* Obtenido de https://serenity-bdd.github.io/theserenitybook/latest/index.html

Haleby, J. (s.f.). *Rest - Assured Version 2.x*. Obtenido de https://rest-assured.io/

Jenkins. (s.f.). *Jenkins Continuous Integration*. Obtenido de https://www.jenkins.io/doc/

Maven, A. (2000). *The Apache Software Foundation.* Obtenido de https://maven.apache.org/

Nexus, R. (s.f.). *Nexus repository*. Obtenido de https://www.sonatype.com/products/nexus-repository

North, D. (2006). *Introducing BDD.* Obtenido de https://dannorth.net/introducing-bdd/

Oloruntoba, S. (19 de 02 de 2021). *SOLID.* Obtenido de https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design-es

POM. (07 de 08 de 2014). Obtenido de https://qanewsblog.com/2014/08/07/patrones-de-diseno-en-automatizacion-page-objects/#comments

Selenium. (s.f.). *The Selenium Browser Automation Project*. Obtenido de https://www.selenium.dev/documentation/

SmartBear, S. (2019). *Cucumber Guide*. Obtenido de cucumber.io/docs/guides/: https://cucumber.io/

SmartBear, S. (2019). *Gherkin Syntax*. Obtenido de cucumber.io/docs/gherkin/: https://cucumber.io/docs/gherkin/

Sutherland, K. S. (2020). *La Guía Definitiva de Scrum.* Obtenido de https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf

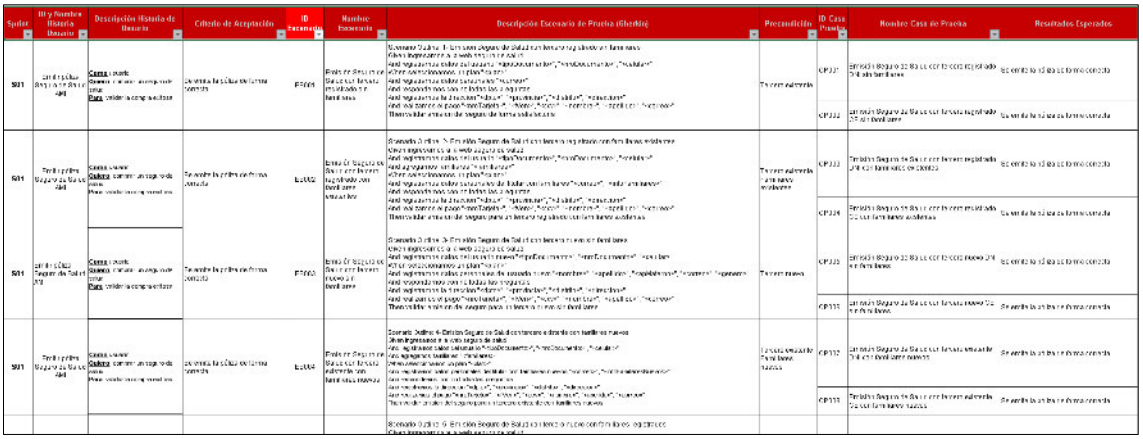
5.4. Glosario

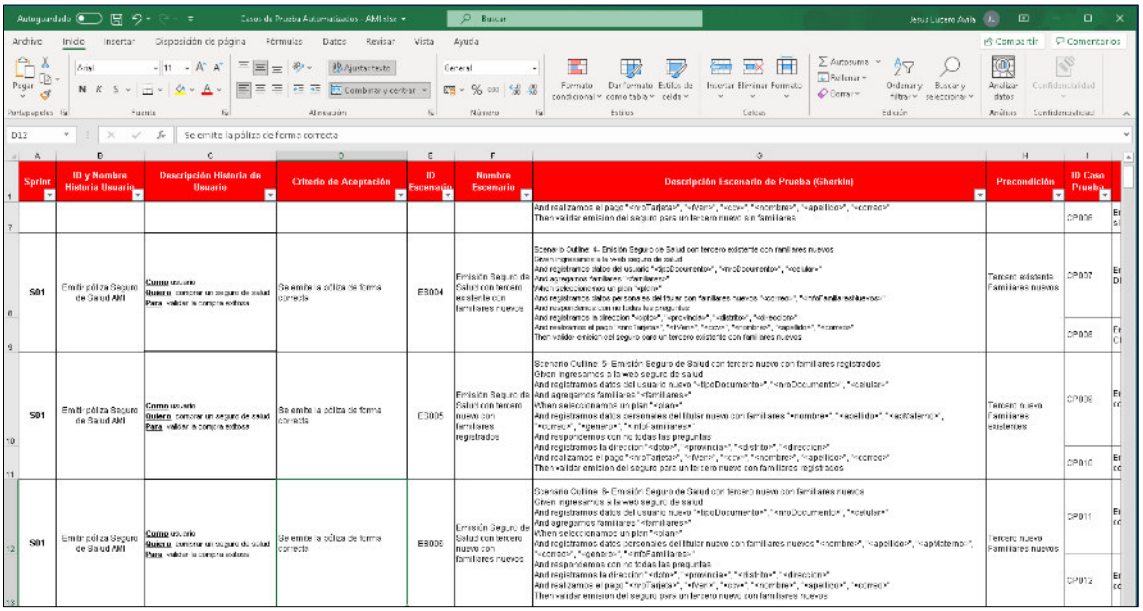
- **API:** Son un conjunto de comandos, funciones y protocolos que permiten la comunicación entre dos sistemas o plataformas diferentes.
- **Integración continua:** La integración continua es un proceso automatizado que consiste en hacer integraciones automáticas de proyectos continuamente.
- **Código abierto:** Este es un término utilizado para describir el desarrollo de software que está disponible gratuitamente para todos los desarrolladores.
- **Pipeline:** Un pipeline consta de múltiples Stages, cada una de las cuales se ejecutará en orden. Si un Stage falla, entonces el pipeline se considera fallida y los siguientes Stages no se iniciarán.
- **Jenkins:** Es una herramienta de integración continua de código abierto y actualmente uno de los más empleados para organizar una cadena de acciones para lograr el proceso de integración continua de forma automatizada.
- **Pruebas de regresión:** Son las pruebas que se realizan luego de realizar cambios en una aplicación para verificar que estos cambios no hayan impactado en otros componentes y sigan funcionando correctamente
- **Cobertura de pruebas:** Es una medida del total de pruebas que han sido realizadas en base a un determinado criterio.
- **Maven:** Es una herramienta de código abierto desarrollada en Java para la gestión de dependencias y construcción de proyectos de software en todo su ciclo de vida.
- **Caso de prueba:** Es un conjunto de pasos y acciones con resultados esperados basados en los requerimientos del proyecto.
- **Repositorio:** Es la ubicación lógica para el almacenamiento del código fuente de los proyectos de software, donde pueden ser creados de forma local o usar los servicios de la nube.
- **Pruebas automatizadas:** Son escenarios de pruebas que han sido implementados a través de alguna herramienta de automatización y se puede realizar las ejecuciones de forma programada o a demanda.
- **Eclipse:** Es una interfaz de desarrollo de software de código abierto.
- **Tercero:** Es una persona natural o jurídica.
- **Selenium:** Conjunto de librerías que permite automatizar aplicaciones web con distintos navegadores.

- **Cucumber:** Es una herramienta que soporta BDD, la cual nos va a permitir automatizar escenarios de pruebas que están hechos con BDD usando el lenguaje Gherkin.
- **DevOps:** Es un enfoque de desarrollo que trata de crear funcionalidades más rápidas para el negocio en entornos estables, disponibles y seguros.
- **BDD:** Es una técnica de desarrollo que se centra en el comportamiento el sistema y que utiliza el lenguaje Gherkin.
- **Historia de usuario:** Es una descripción de la funcionalidad o necesidad del negocio de lo que un usuario quiere hacer dentro de un producto de software.
- **Scrum Master:** Es la persona encargada de promover, apoyar y mantener a los miembros del equipo enfocados en los principios del marco de trabajo ágil.
- **Jira:** Esta es una herramienta ágil de gestión de proyectos utilizada para diferentes tipos de equipos de trabajo.
- **Request:** Es la petición de solicitud de información con una estructura determinada.
- **Response:** Es la respuesta a una petición de información con una estructura determinada.
- **Gherkin:** Define una estructura y sintaxis básica para la descripción de los escenarios de pruebas que pueden ser entendidas tanto por personas de negocio y técnicos.

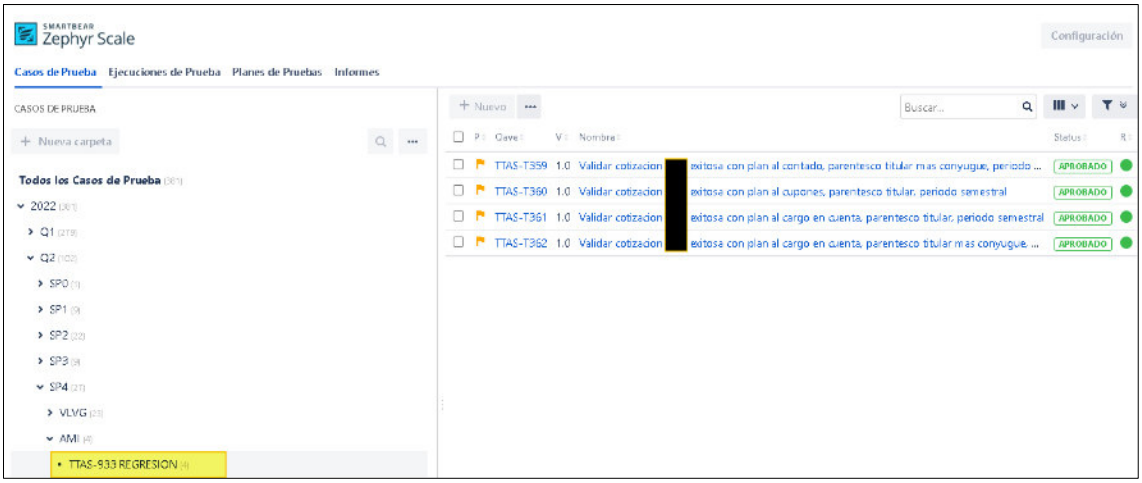
ANEXOS

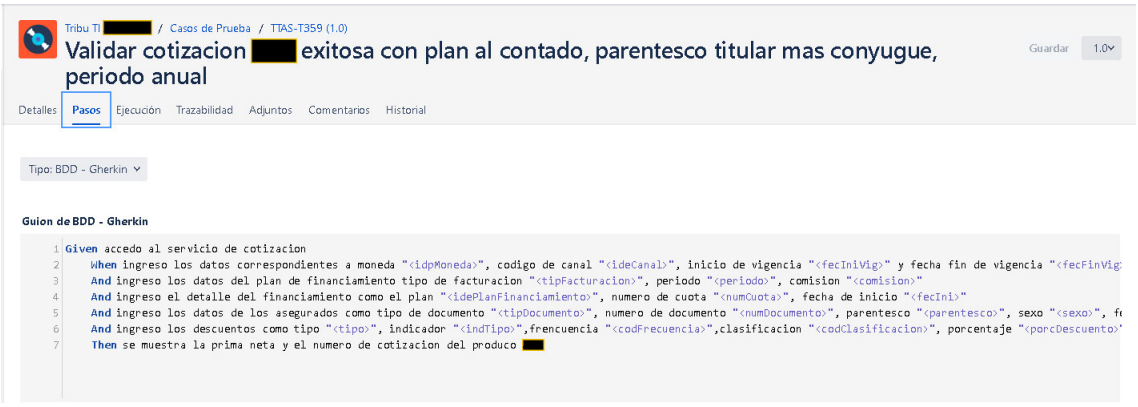
Anexo 1 A. Documento de escenarios de pruebas automatizadas web





Anexo 2 A. Documento de escenarios de pruebas automatizadas APIs



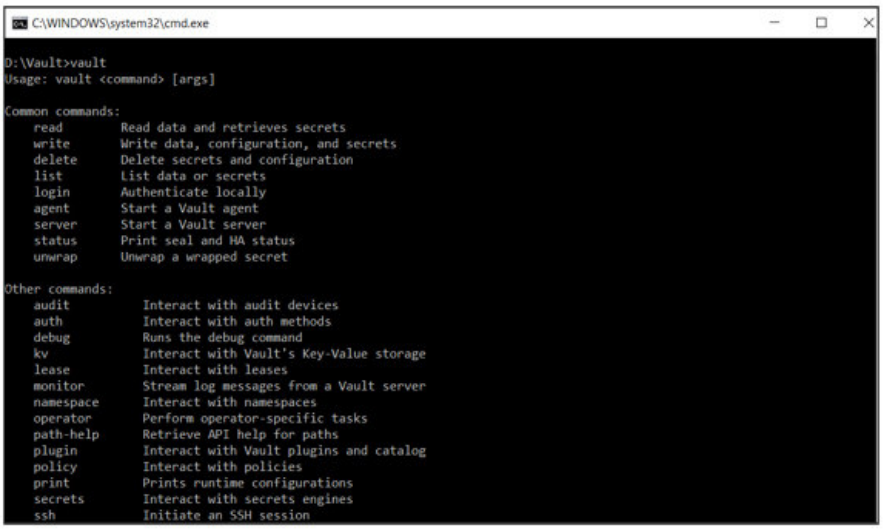


Anexo 3 A. Manual de instalación de la herramienta Vault

MANUAL DE INSTALACIÓN DE HASHICORP VAULT

1. **Descarga e instalación:**

   - Descargar Vault desde su página oficial: https://www.vaultproject.io/downloads.
   - El paquete (zip) debe ser descomprimido en una carpeta específica para ser tratado como un programa, ya que vault no posee una instalación del SO en ejecución.
   - Para confirmar simplemente debemos ingresar a la ruta en la cual hemos alojado el paquete e ingresar la sentencia "vault". Esto nos confirma que ya podemos ejecutar sentencias Vault solo en esta ruta.
   -



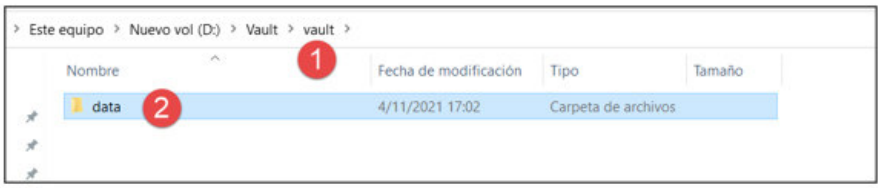
2. Configuración

Vault permite 2 tipos de configuraciones, modo desarrollador y modo producción.

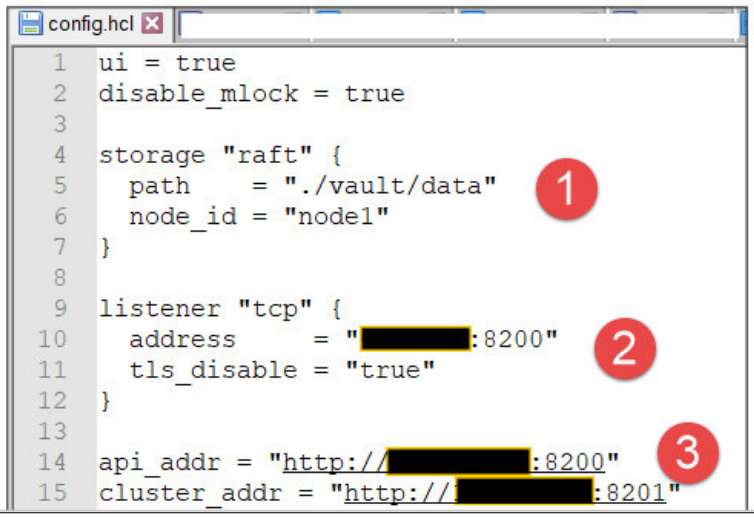
- La primera forma levanta el servicio de manera temporal, el objetivo de esta es probar cambios puntuales para luego ser pasados a un servidor productivo; al cerrar el ejecutor del servicio todas las configuraciones se pierden.

- La forma productiva requiere de un archivo de configuración para la permanencia de la información cada vez que levantamos el servicio, esta forma es la que vamos a considerar tanto para el trabajo local como para las ejecuciones con Jenkins.

- En la carpeta de instalación se deberá crear a la carpeta vault>data:



- Se deberá crear un archivo del tipo "config.hcl" donde se especificará lo siguiente:



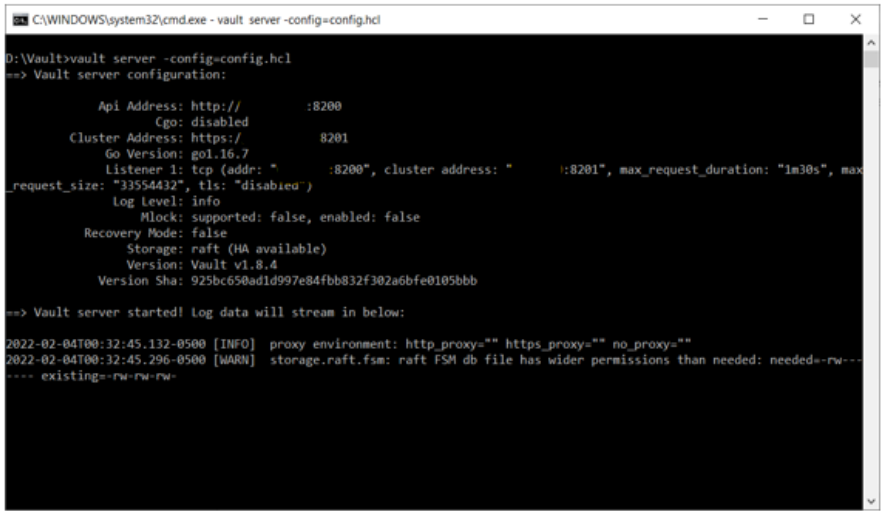
- 1: Ruta que creamos en el paso anterior para almacenar todas las configuraciones que realizaremos dentro de la aplicación. Es indispensable ya que cada vez que se inicia el servicio de Vault leerá de este Storage.

- 2: El puerto de escucha del servicio de Vault. De preferencia dejaremos desactivado el TLS de seguridad para no generar certificados.

- 3: La ruta y puerto donde se expondrá la aplicación web de Vault.

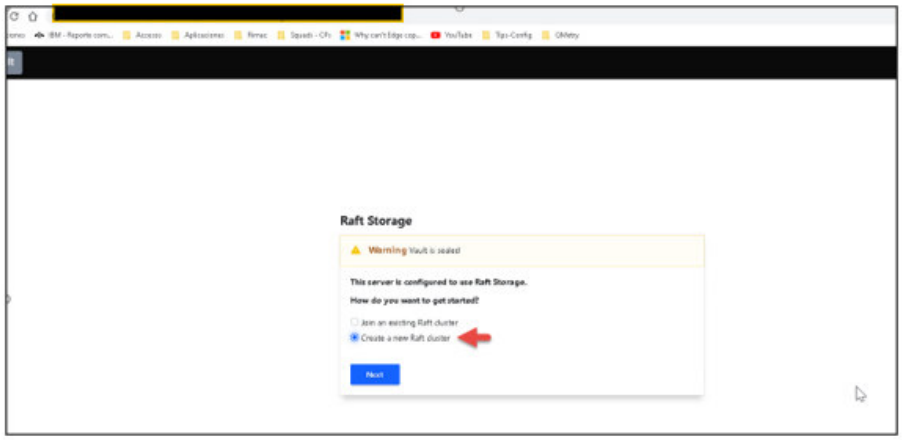
3. Iniciar servicio e ingresar a la aplicación

- Para iniciar el servicio se deberá ejecutar dentro de la ruta de instalación él comando "vault server -config=config.hcl"

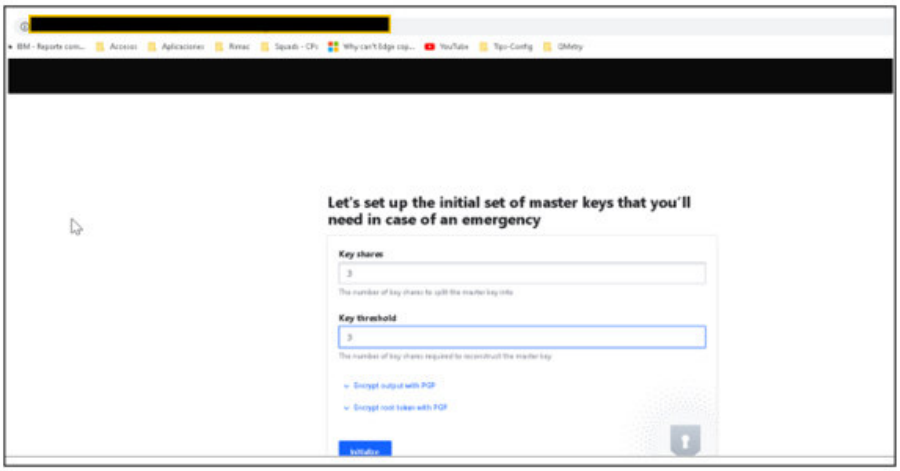


- Una vez iniciado el servicio ingresaremos a la ruta http://[redacted]:8200

- Al ingresar se mostrará la siguiente ventana y seleccionaremos la segunda opción.

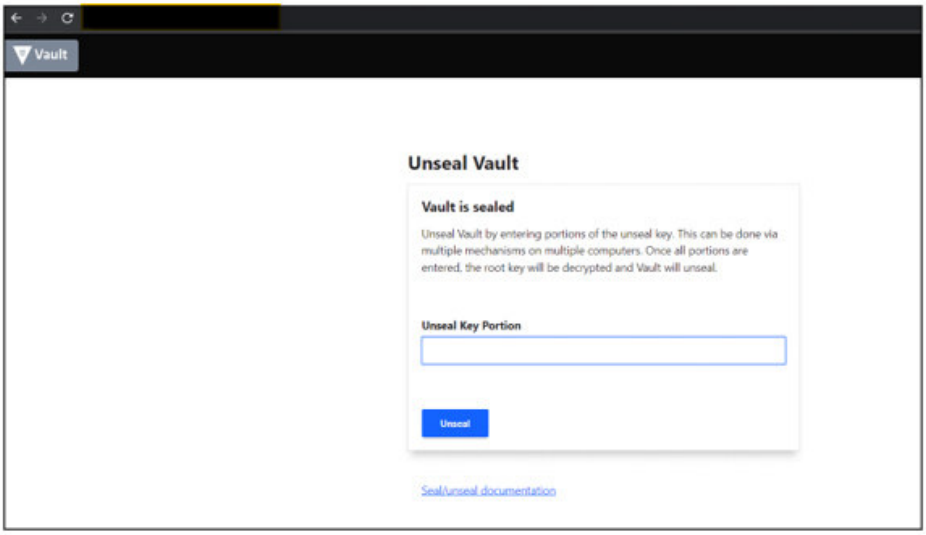


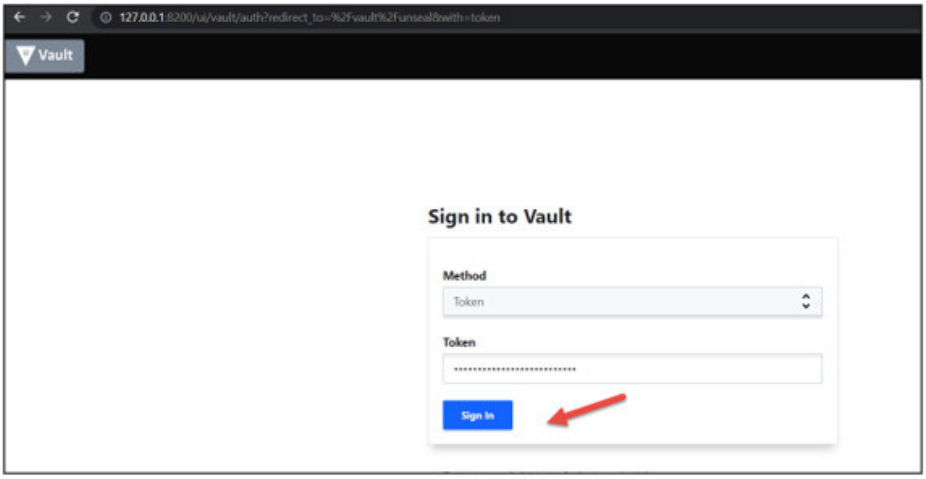
- Luego se mostrará esta ventana donde ingresaremos la cantidad de Key que se generarán para poder restaurar en caso perdamos las credenciales de administrador.



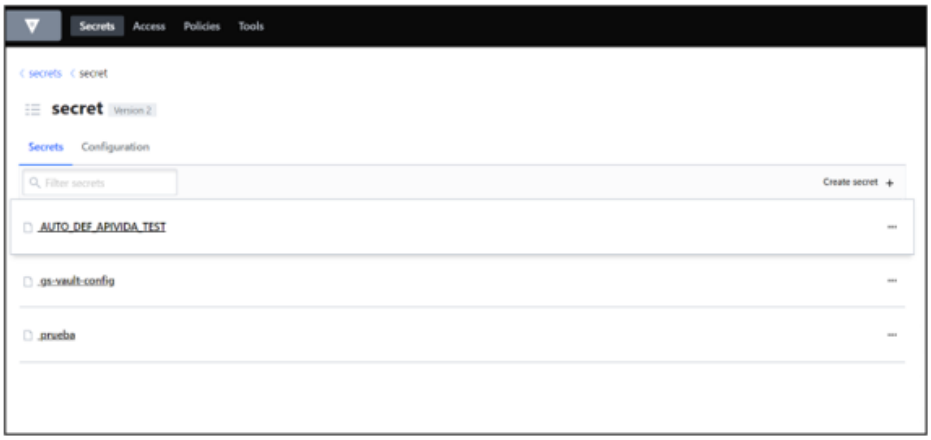
- Luego que se generen las Key, realizaremos las descargas de estas. Se creará un archivo "Json" con toda la información necesaria para los accesos. Este archivo deberá ser guardado en una ruta especifica.

- Para ingresar a Vault de modo administrador, nos pedirá el ingreso de los Key y el token del usuario root, todo esto se encuentra en el archivo "json" descargado.





- Luego de esto ya podemos crear y almacenar los secretos de nuestro proyecto.



- Al reiniciar el servicio se deberá ejecutar la sentencia antes indicada "vault server -config=config.hcl".

Anexo 4 A. Manual de ejecución de pruebas automatizadas en Jenkins

MANUAL DE EJECUCIÓN

1. Ingresar a Jenkins:

Jenkins es el servidor seleccionado como herramienta para el proceso de Integración Continua que nos permite la ejecución de pipelines.

Ruta: http://[redacted]:8080/login?from=%2F
Usuario: [redacted]
Contraseña: [redacted]

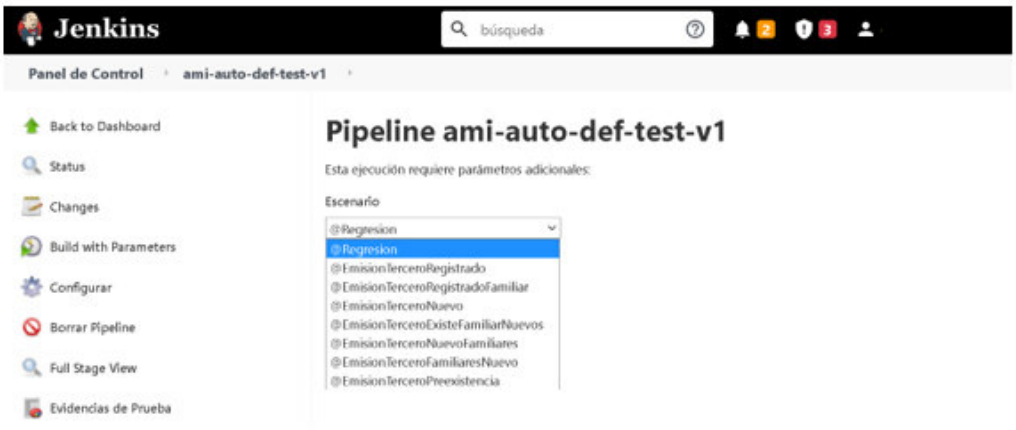
Pipeline para el proyecto:

Seleccionar el siguiente Proyecto: ami-auto-def-test

2. Ejecución de Escenarios:

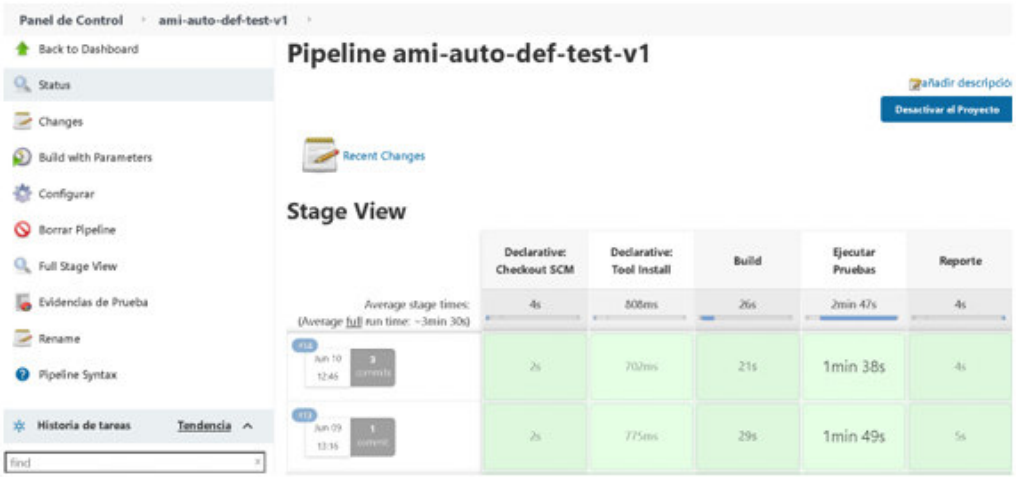
Dar Clic a la Opción "Build with Parameters"

Iniciamos con la ejecución de los Escenarios de prueba, en este caso seleccionamos "`@EmisionTerceroRegistrado`" y clic al Botón "Ejecución"



**Los Escenarios se encuentran ordenados, se debe respetar el orden de ejecución.*

Visualizamos que se está ejecutando el escenario con un número de Tarea



Validamos que termino su ejecución, ya que dejo de parpadear la Tarea, así como se muestra el tiempo de Ejecución para: Build, Ejecutar Pruebas y Reporte.

3. Visualizar evidencia de Ejecución

Luego de que termine la ejecución visualizamos el reporte generado por serenity.