**VERSIONES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Comentarios** | **Autor** |
| 15/06/2022 | 1.0 | Versión Inicial | Abraham Rivera R. |
| 16/06/2022 | 1.1 | Ejecución Cli | Flavio Romero |
| 06/07/2022 | 1.2 | Estrategia de Mantenimientos | Abraham Rivera R. |
| 26/09/2022 | 1.3 | Actualización de documento | Eduardo Arreátegui Li |

Índice

[1. Objetivo 4](#_Toc31795946)

[2. Términos y Definiciones 4](#_Toc31795947)

[3. Framework de Automatización 4](#_Toc31795948)

[3.1. Gobierno de TI 5](#_Toc31795949)

[3.2. Roles de Automatización 5](#_Toc31795950)

[3.3. Cobertura de Automatización 5](#_Toc31795951)

[3.4. Framework Técnico 6](#_Toc31795952)

[3.5. Capa de Soporte de TI 6](#_Toc31795953)

[3.6. Administración de Scripts 6](#_Toc31795954)

[3.7. Entrega Continua 8](#_Toc31795955)

[4. Gobierno de TI 8](#_Toc31795956)

[5. Framework Técnico 10](#_Toc31795957)

[5.1. Diseño del Framework 10](#_Toc31795958)

[5.2. Diseño de los componentes 11](#_Toc31795959)

[5.3. Estructura de los directorios 11](#_Toc31795960)

[5.3.1. Directorio raíz 12](#_Toc31795961)

[5.3.2. Directorio main 12](#_Toc31795962)

[5.3.3. Directorio Java 12](#_Toc31795963)

[5.3.4. Directorio resources 13](#_Toc31795964)

[5.4. Librerías 14](#_Toc31795965)

[6. Administración de Scripts 15](#_Toc31795966)

[6.1. Acerca del repositorio 15](#_Toc31795967)

[6.1.1. Datos del repositorio 15](#_Toc31795968)

[6.1.2. Características del repositorio 15](#_Toc31795969)

[6.2. Dinámica de trabajo 16](#_Toc31795970)

[7. Ejecución de CLI (Command Line Interface) 17](#_Toc31795971)

[8. Desarrollo de nuevos scripts de automatización 19](#_Toc31795973)

[8.1. Proceso 21](#_Toc31795974)

[8.2. Requerimientos de configuración y Herramientas 21](#_Toc31795975)

[8.3. Desarrollo de nuevos scripts 22](#_Toc31795976)

[8.3.1. Acceso al repositorio. 23](#_Toc31795977)

[9. Glosario 23](#_Toc31795984)

# Objetivo

El objetivo del documento es describir el framework de automatización propuesto para NIUBIZ, asimismo familiarizar al Automatizador con el marco de trabajo para la automatización de pruebas para las siguientes aplicaciones:

a. Pago Web

b. Pago App

c. Tokenización

# Términos y Definiciones

**AAPg:** Arquitectura de Automatización de Prueba Genérica

**SAP:** Sistema de Automatización de Pruebas

**SSP:** Sistema Sujeto a Prueba (PAGO WEB).

**GUI:** Interfaz Gráfica de Usuario

**Archivo “.feature”:**  Archivo que contiene el paso a paso del escenario de prueba. La descripción del paso a paso se desarrolla haciendo el uso de la herramienta Cucumber.

**Escenario de Prueba:** Para los casos de prueba del aplicativo Pago Web / Pago App / Tokenización, cada escenario de prueba dentro del feature representa a un caso de prueba.

**Aplicativo Conocido:** Aplicativo el cual tiene algunos casos de prueba ya automatizados.

**Aplicativo Nuevo:** Aplicativo el cual no tiene casos de prueba automatizados.

# Framework de Automatización

El framework propuesto para Niubiz se desarrolla para todo el ciclo de automatización de pruebas funcionales. Este framework contempla actualmente (5) cinco de los (7) siete componentes, dejando (2) dos para mejora continua:

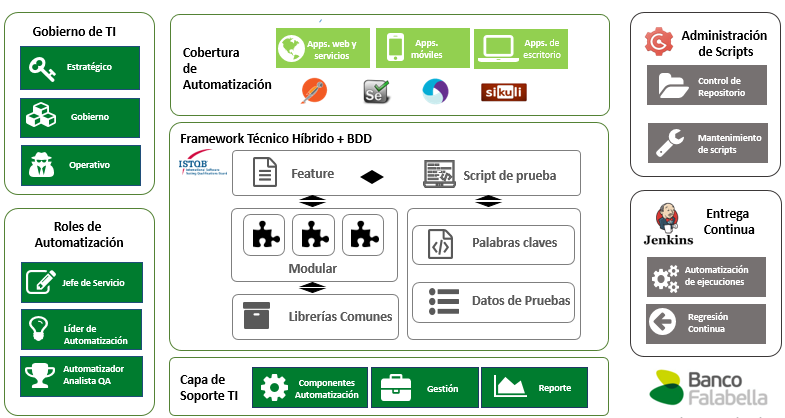


Figura N° 1 Framework de Automatización de NIUBIZ

## Gobierno de TI

El framework define la participación de la automatización de pruebas en cada una de las tres capas de gobierno de TI en las organizaciones: (a) A Nivel Estratégico, (b) De Gobierno, y (c) Operativo.

1. **Nivel Estratégico.** La automatización de pruebas de software es una estrategia operativa alineada a objetivos estratégicos de la gerencia de TI. El fin de este objetivo estratégico es la reducción de costos, asimismo, optimizar el modelo de calidad de software y validación de los sistemas.
2. **Nivel de Gobierno.** La automatización de prueba se encuentra bajo el gobierno de una estructura de gestión propuesta por NIUBIZ.
3. **Nivel Operativo.** Se cuenta con una (AAP) Arquitectura de Automatización de Pruebas estándar definida.

## Roles de Automatización

El framework de automatización plantea 3 roles que se manejan de forma colaborativa:

**- Jefe de servicio.** Es el encargado de realizar el rol de facilitador así mismo de monitorear el desarrollo eficiente de los proyectos, pilotos o servicios de automatización.

**- Líder de automatización.** Es el facilitador técnico y de estrategia, quien tendrá a cargo la centralización de las prácticas de automatización de soporte y de proyectos waterfall y/o ágiles, para así llevar una adecuada administración de los mismos; lo que permitirá identificar puntos de mejora continua, tanto en los procesos, así como también de impartir entrenamiento y guía a los miembros del equipo.

**- Automatizador.** Encargado de realizar las actividades operativas tales como administración y soporte de la infraestructura, desarrollo, mantenimiento y ejecución de los scripts de automatización, entre otros.

**-** **Analista QA.** Encargado de realizar las transferencias funcionales para futuros desarrollos en automatización, además de ejecutar los scripts automatizados.

## Cobertura de Automatización

El framework técnico en combinación con diferentes herramientas de automatización de pruebas, permite la cobertura de pruebas en aplicativos SERVICIOS / WEB:

Herramientas definidas

|  |  |
| --- | --- |
| Aplicaciones web | Selenium / WebDriver / JAVA |
| Servicios | Selenium / HttpClient / JAVA |

## Framework Técnico

El proceso de pruebas de automatización se basa en el marco técnico de trabajo híbrido para los proyectos en cascada, combinando hasta cuatro patrones de automatización, según aplique en cada caso: (1) Modular, (2) Liberías comunes, (3) Datos de Pruebas, y (4) Palabras claves. El beneficio de esta configuración es que aprovecha las funcionalidades de todos los tipos de marcos asociados. Así mismo, para proyectos ágiles se integra un patrón adicional, BDD, en el cual se contemplan los (0) features (escenarios de prueba) como punto de partida.

Este framework permite los siguientes beneficios:

* Mantenimiento eficiente debido al diseño modular y librerías de funciones comunes.
* Escalabilidad, presenta desacoplamiento, es fácil el adicionar componentes para mayor cobertura de los casos de pruebas automatizados.
* Flexibilidad, permite ejecutar diferentes casos de pruebas a partir de diferentes datos de entrada.
* Usabilidad, a través de palabras podemos lograr comportamientos distintos sin tener que ingresar al código.

Adicionalmente, para el caso de proyectos ágiles se añadirá BDD, lo caul permite sumar a los beneficios lo siguiente:

* Fácil entendimiento, permite incluir en la automatización el escenario de manera escrita (lenguaje Gherkin) lo que permite que cualquier rol entienda el comportamiento que tendrá el escenario automatizado al ejecutarse.

## Capa de Soporte de TI

El framework plantea los siguientes servicios como soporte de TI base:

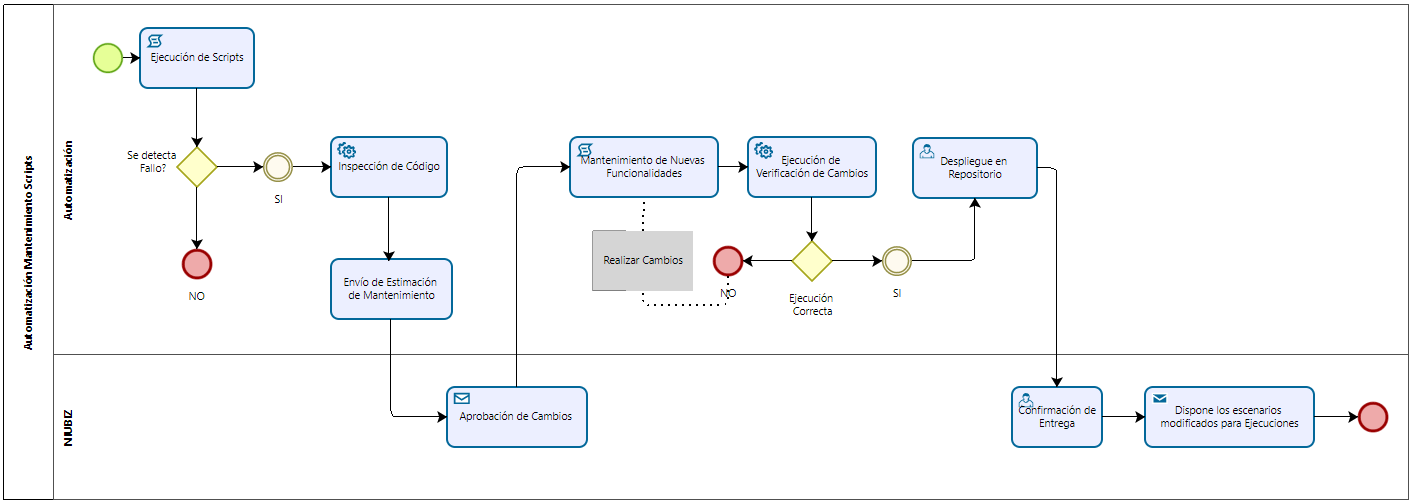
* Las herramientas correspondientes a la arquitectura de la automatización reciben soporte técnico y mantenimiento oportuno.
* Gestión de pruebas automatizadas lleva un control del inventario de los casos de pruebas automatizados y el inventario de las evidencias generadas.
* Los reporte e indicadores que son generados al finalizar las ejecuciones, permiten llevar seguimiento del desarrollo y mantenimiento de scripts.

## Administración de Scripts

El framework brinda soporte de administración técnico de los scripts de automatización con el Control de repositorio y Mantenimiento de scripts.

**Estrategia de Mantenimiento:**

* Para asegurar un mantenimiento correcto, el FW Técnico debe considerar lo siguiente:
  + Debe ser **MODULAR**
  + Debe ser **ESCALABLE**
  + Debe ser **COMPRENSIBLE** (Documentado)
  + Debe ser **CONFIABLE**
  + Debe ser **VERIFICABLE**
* **Tipos de Mantenimiento:**
  + **Preventivo**: se realizan cambios para mejorar la operación.
  + **Correctivo**: cambios para corregir fallos en el Sistema de Automatización de Pruebas ante algún cambio del SSP.
  + **Perfectivo**: se establecen cambios para optimizar.
  + **Adaptativo**: se hacen cambios para admitir nuevas aplicaciones



1. Un mantenimiento correctivo se detecta al ejecutar un script y este falle por causas de algún cambio en el SSP.
2. Se realiza una inspección de código para detectar el módulo / clase / objeto(s) afectado(s).
3. Se prepara y envía la estimación con el esfuerzo por realizar el mantenimiento.
4. El cliente aprueba el esfuerzo y se procede con los cambios.
5. El equipo de automatización realiza los cambios hasta tener una nueva versión que ejecute sin fallos.
6. Se actualiza el código en el repositorio centralizado.
7. Se realiza una prueba para confirmar los cambios desde el repositorio central.
8. Se dispone los scripts modificados para su ejecución.

## Entrega Continua

El framework está preparado para integrarse al proceso de entrega continua que posea NIUBIZ, tanto para la Ejecución de automatizada y Regresión continua.

# Gobierno de TI

La estrategia de automatización que plantea el gobierno de TI es la siguiente:

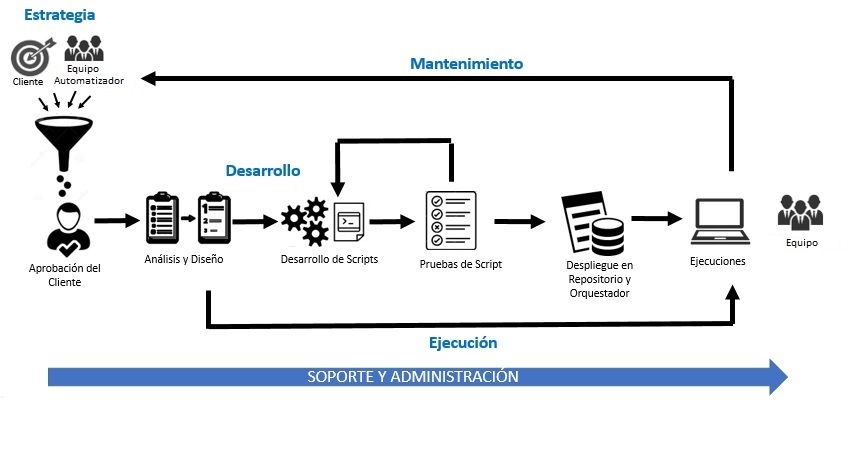


Figura N° 2 Estrategia - Gobierno TI

1. **Estrategia:**

Los requerimientos de automatización serán identificados por NIUBIZ (Equipo de Calidad). Los Analistas de QA transfieren información acerca de los casos de prueba o flujos que se proponen automatizar. Esta información pasa por un análisis de factibilidad, el cual, será descrito más adelante. El análisis de factibilidad asigna una clasificación y un puntaje a los flujos o casos de prueba, permitiendo determinar los flujos óptimos a ser automatizados.

Cuando se tenga los flujos ya identificados se realiza una Estimación Alto nivel, la cual permite la elaboración del Plan de Automatización.

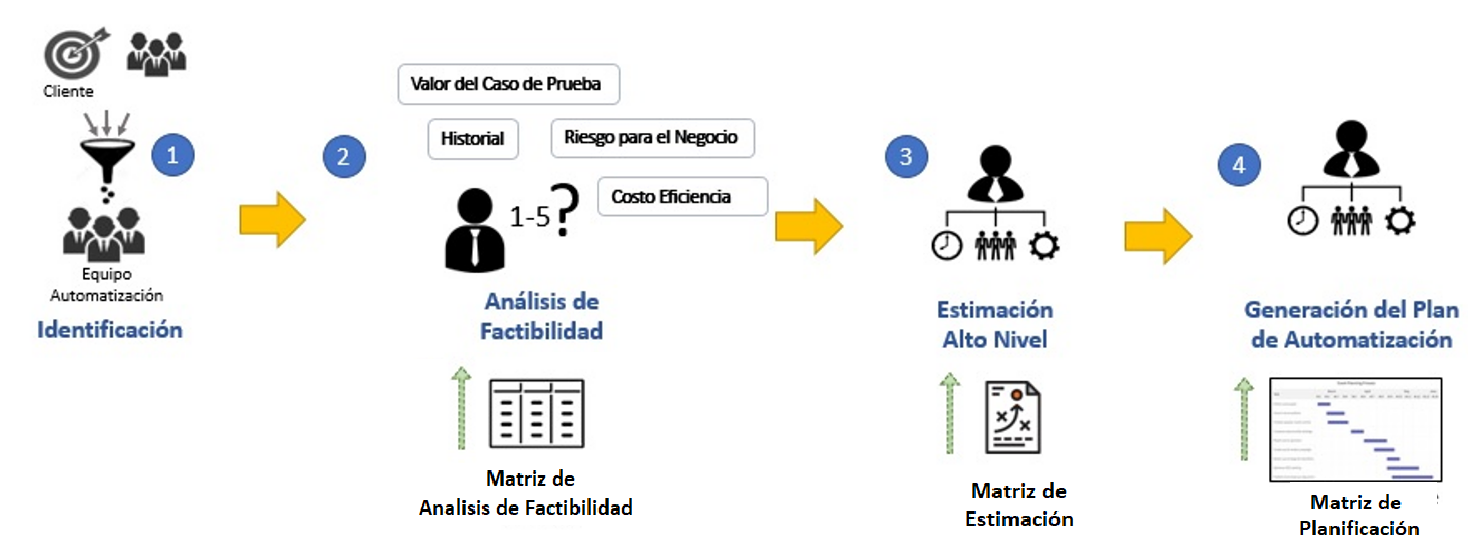


Figura N° 3 Descripción de la estrategia

Cuando el plan de automatización es aprobado por NIUBIZ, se da inicio a la etapa de desarrollo.

Análisis de Factibilidad:

Las actividades a realizar dentro de esta etapa son las siguientes:

1. **Actividad: Análisis Nivel 1**

Objetivo:

* Identificar los candidatos que sean viables funcionalmente/técnicamente.
* Asignar un nivel de complejidad
* Asignar puntuación a los candidatos viables, en base a los siguientes criterios: Valor de Caso de Prueba, Riesgo de Negocio, Costo-Eficiencia e Historial.

Herramientas usadas:

* Matriz de Análisis Factibilidad
* Reuniones con los Analistas QA asignados / personas que tengan el know-how del SSP a ser automatizado.
* Al finalizar el análisis se procede a realizar la estimación (Matriz de Estimación Alto Nivel).

1. **Desarrollo**

Es una etapa cíclica que consiste en desarrollo y pruebas. Se realiza el desarrollo de los flujos identificadas y posteriormente se realiza las pruebas de los mismos.

Cuando los scripts ya se encuentren verificados, se realiza el despliegue en el repositorio.

Finalmente, los scripts estarán disponibles para las ejecuciones programadas o a demanda.

1. **Mantenimiento**

Los scripts que necesiten un mantenimiento, serán identificados por NIUBIZ o por el equipo de TSOFT. Luego se hará de conocimiento al cliente para su aprobación y priorización dentro del backlog del equipo.

1. **Soporte y Administración**

Durante todo el proceso de atención de automatización se cuenta con el soporte por el equipo de TSOFT.

# Framework Técnico

## Diseño del Framework

El framework propuesto para la automatización de pruebas, es el siguiente:

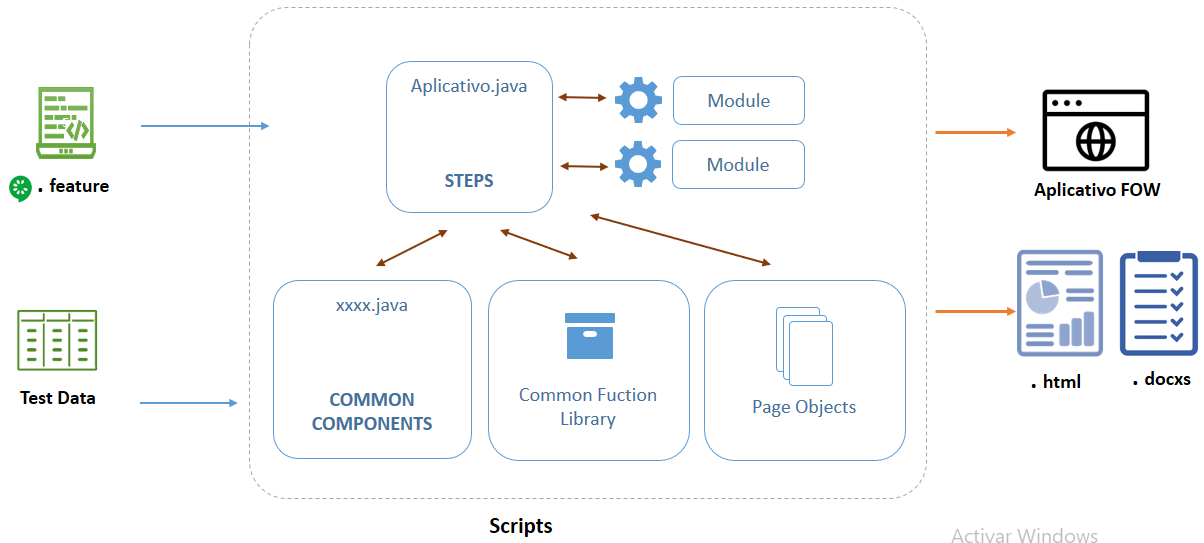


Figura N° 4 Framework Técnico de Automatización de Pruebas

El framework se basa en el patrón de diseño **Page Object Model o POM**, pues permite mantener por separado los localizadores de los elementos Web, de los métodos de las pruebas. Adicional a ello, se ha implementado una clase Base (véase Figura N° 5), la cual contiene los métodos principales para la manipulación de los elementos Web con **Selenium Web**, como hacer un clic o escribir un texto*.* Así mismo, una de las librerías usadas en esta arquitectura es Cucumber, la cual, permite escribir los escenarios de pruebas con un lenguaje natural los cuales, serán almacenados en los archivos con la extensión **“.feature”** permitiendo a su vez la configuración y el desmontaje del entorno antes y después de cada escenario, mediante el desarrollo de **Hooks**.

**Steps** es el paquete que contiene las clases que implementan los pasos detallados en el archivo “.feature”. Los métodos de estas clases son las que interactúan con el navegador y hacen uso de otros métodos para la generación de reportes.

La generación de reportes se realiza con las librerías Extend Report , para los reportes de html y Word respectivamente.

**Interacción de componentes del framework en la ejecución de un escenario**

Los puntos de entrada para la ejecución automática de pruebas, son los archivos feature y Excel. Todos los escenarios de prueba listos para automatizar se encuentran en los archivos feature y los datos de entrada para los escenarios de prueba se encuentran en el Excel.

Al ejecutar el proyecto se envía una lista de parámetros, como la url, usuario, ambiente, tag, etc. Uno de los parámetros que indica que casos de prueba se ejecutarán, es el tag. El cual debe ser previamente configurado en el archivo “feature”.

Para la ejecución de un escenario de prueba se lee los datos de entrada del Excel. Luego se ejecuta cada paso detallado en el feature, el cual tiene su implementación dentro del paquete Steps. Cuando un paso indica una acción con un elemento web, se ubica en el Page Object el localizador del elemento web al que se desea acceder y mediante el método de manipulación del objeto de la clase BaseClass, se efectúa la acción. Al ejecutar los pasos se van generando evidencias en Word y html.

Dependiendo de los hooks implementados, existirán algunos métodos que se ejecuten antes y/o después de los escenarios de prueba.

## Diseño de los componentes

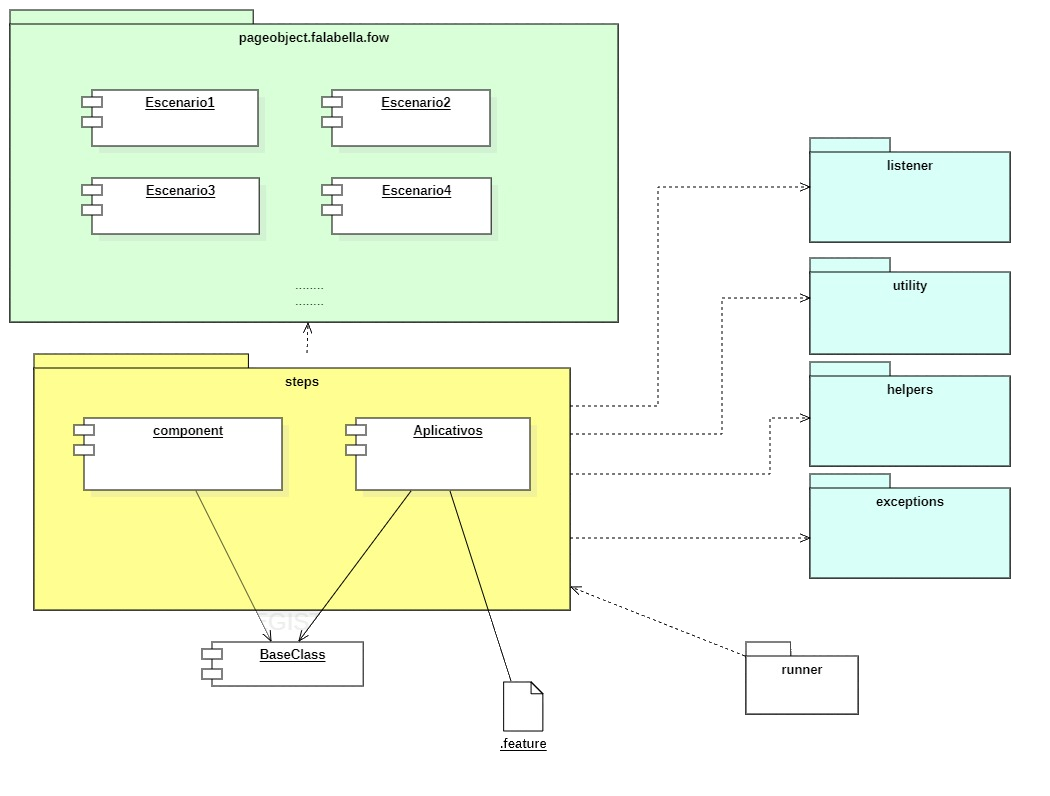


Figura N° 5 Diseño de los componentes

## Estructura de los directorios

El framework ha sido desarrollado en lenguaje JAVA, bajo la estructura de directorios de MAVEN. Esto significa que las clases o recursos deben ir en la carpeta indicada por MAVEN, para que luego puedan ser procesados correctamente.

Datos del proyecto:

**GroupId:**  com.tsoft.bot.frontend

**ArtifactId:**  test-automation-frontend

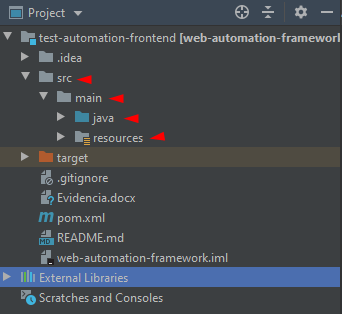


Figura N° 6 Estructura MAVEN del framework

### Directorio raíz

**Directorio src:** Directorio en el cual se almacenan todos los recursos del proyecto (directorio main).

### Directorio main

Directorio en el cual se almacenan las clases (directorio java) y recursos del proyecto (directorio resources).

1. **Directorio java:** Directorio que almacena las fuentes del proyecto.
2. **Directorio resources:** Directorio que almacena todos los recursos que se necesite en el proyecto.

### Directorio Java

Dentro de este directorio Java se encuentran los siguientes directorios:

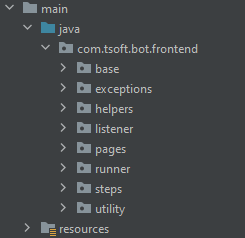


Figura N° 7 Directorios de las fuentes del proyecto

1. **Directorio exceptions:** Contiene las excepciones creadas para el proyecto.
2. **Directorio helpers:** Contiene la clase Hook de cucumber (acciones que se ejecutarán antes o después de los escenarios, etc.)
3. **Directorio listener:** Contiene la Clase Listener, para la manipulación del Reporte (Extend Report).
4. **Directorio pageobject:** Contiene los Page Object de los escenarios de prueba, ordenados según el aplicativo y escenario.

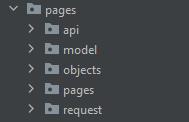


Figura N° 8 Directorios de los page objects del proyecto

1. **Directorio runner:** Contiene la clase principal para ejecutar el proyecto. Run Test.java y las configuraciones de cucumber.
2. **Directorio steps:** Contiene las clases que implementan los pasos detallados en los archivos feature.

La jerarquía de las carpetas sigue el siguiente orden, los pasos se encuentran dentro de una clase java con el nombre del aplicativo. Si hay acciones comunes que son compartidas entre diferentes pasos, pero el mismo aplicativo, se colocan dentro de una carpeta con el nombre del aplicativo, en “Component”; si las acciones son comunes y compartidas entre diferentes aplicativos, se colocan dentro de “Common”, en “Component”.



Figura N° 9 Directorios de los steps del proyecto

1. **Directorio utility:** Contiene las clases que son utilitarios para el proyecto, como el manejo del Word, Sleeper , conexión a Base de datos o las conexiones con los servicios SOAP.

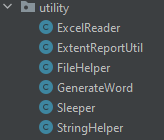


Figura N° 10 Directorio utiliy del proyecto

### Directorio resources

Dentro de este directorio Java se encuentran los siguientes directorios:

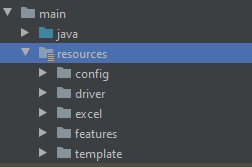


Figura N° 11 Directorios de recursos del proyecto

1. **Directorio config:** Contiene los archivos xml de configuración del proyecto.

* Extent-config.xml: datos de configuración del reporte html.
* Testng.xml: datos de configuración para la ejecución del proyecto.



Figura N° 12 Lista de archivos dentro del directorio config

1. **Directorio driver:** Contiene los drivers de los navegadores.

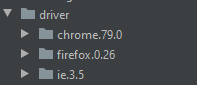


Figura N° 13 Estructura del directorio de drivers

1. **Directorio excel:** Contiene el Excel con los datos de entrada para los casos de prueba. De acuerdo al aplicativo y la organización de los datos de entrada. Se crea una carpeta con el nombre del aplicativo y dentro se crea un Excel por cada funcionalidad. Dentro de cada Excel, se crea una Hoja por cada Escenario.

****

Figura N° 14 Estructura de los datos de entrada - EXCEL

1. **Directorio features:** Contiene los archivos features. Los archivos features son separados por aplicativo.



Figura N° 15 Directorio de features

1. **Directorio template:** Contiene las plantillas para el reporte en word.

## Librerías

El proyecto al estar desarrollado en una estructura MAVEN, sus dependencias se encuentran dentro del archivo POM.XML. Estas dependencias son descargadas al momento de instalar el proyecto.

Las librerías usadas son:

|  |  |  |  |
| --- | --- | --- | --- |
|  | GroupId | ArtifactId | Versión |
| JSON | org.json | json | 20180813 |
| Reporte HTML | com.aventstack | extentreports | 4.0.9 |
| CUCUMBER | net.masterthought | cucumber-reporting | 4.1.0 |
| info.cukes | cucumber-java | 1.2.5 |
| info.cukes | cucumber-testng | 1.2.5 |
| info.cukes | cucumber-junit | 1.2.5 |
| Manipulación de EXCEL | org.apache.poi | poi | ${apache-poi-version} |
| org.apache.poi | poi-ooxml | ${apache-poi-version} |
| org.apache.poi | poi-ooxml-schemas | ${apache-poi-version} |
| org.apache.poi | ooxml-schemas | 1.3 |
| SELENIUM | org.seleniumhq.selenium | selenium-java | 3.141.59 |
| org.seleniumhq.selenium | selenium-server | 3.141.59 |
| org.seleniumhq.selenium | selenium-api | 3.141.59 |
| org.seleniumhq.selenium | selenium-support | 3.141.59 |
| JUNIT | junit | junit | 4.12 |
| TESTNG | org.testng | testng | 6.10 |
| Librería JAVA | com.google.guava | guava | 22.0 |

# Administración de Scripts

## Acerca del repositorio

### Datos del repositorio

Nombre Repositorio: framework-selenium

Acceso al repositorio: https://bitbucket.org/VisaNet\_TI/framework-selenium/src/master/

### Características del repositorio

Existe solo un repositorio en donde almacena un proyecto java con la estructura maven. Este proyecto incluye escenarios para los diferentes aplicativos de Niubiz.

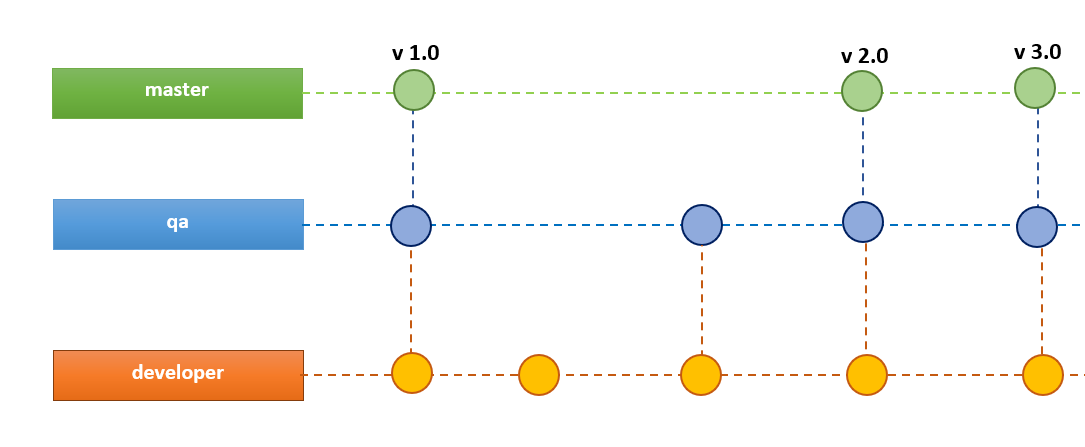
1. **Accesos:**

*Escritura y Lectura:*  Desarrolladores de test

1. **Estrategia de Ramas:**

**Repositorio**: framework-selenium

* **master**: Branch contenedor de las versiones finales de los flujos automatizados.
* **qa:** Branch contenedor de las versiones verificadas por el equipo de calidad.
* **developer**: Branch contenedor de las versiones desarrollados por el equipo de automatización.



## Dinámica de trabajo

Para el adecuado control del repositorio y el mantenimiento de los scripts, se ha propuesto la siguiente dinámica de trabajo.

1. Al llegar un nuevo proyecto, solicitud a demanda de automatización de prueba de un aplicativo o mantenimiento de scripts, se deberá crear una rama con el nombre del proyecto o aplicativo.
2. Los automatizadores descargan la versión del master a su local.
3. Los automatizadores desarrollan los nuevos scripts en la rama developer. Realizan las pruebas correspondientes y realizan un push dentro de la misma rama.
4. Se realiza una revisión de pares al código que se quiere subir a la rama qa.
5. Cuando se aprueba el desarrollo se sube el código a la rama qa.
6. Se aprueba el pull request para que la rama master tenga la versión actualizada.

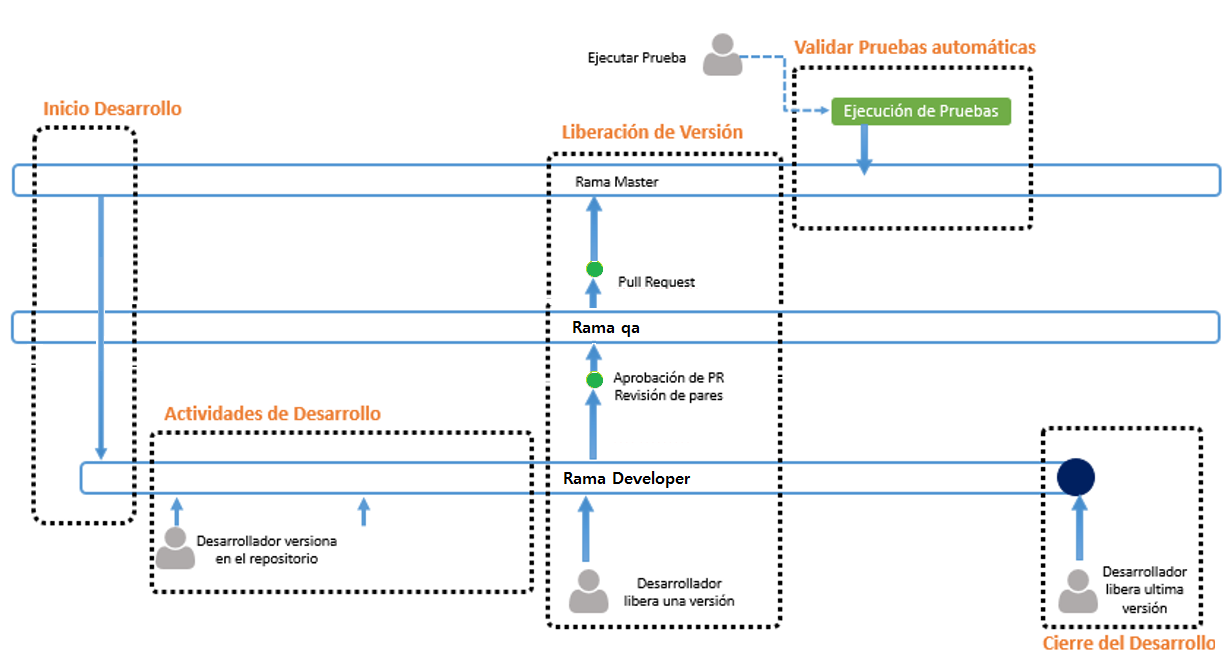


Figura N° 16 Dinámica de Trabajo para interactuar con el repositorio

# Ejecución de CLI (Command Line Interface):

## Arquitectura

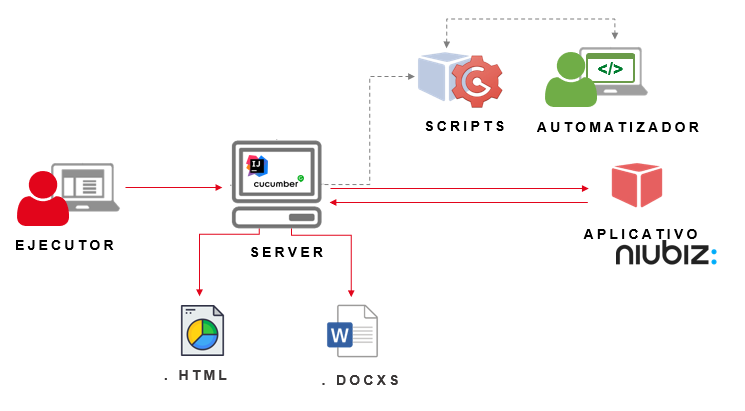


Figura N° 17 Arquitectura

## Configuración del servidor SSH

Configuración de la herramienta freeSSHd

**Sección SSH:** Configurar el adaptador de red, puerto y como opcional el mensaje de bienvenida

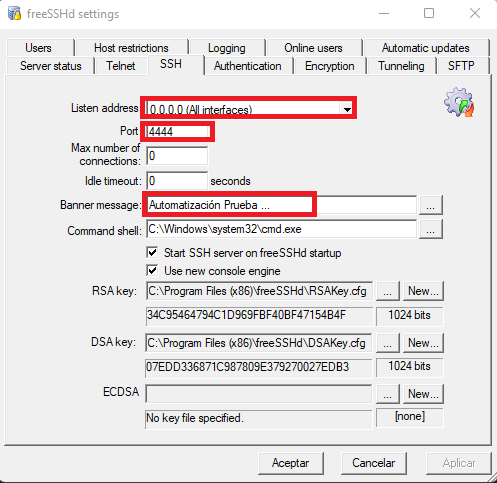


Figura N° 18 Configuración del servidor SSH - Sección SSH

**Sección Authentication:** Seleccionar la authentication Required y Allowed

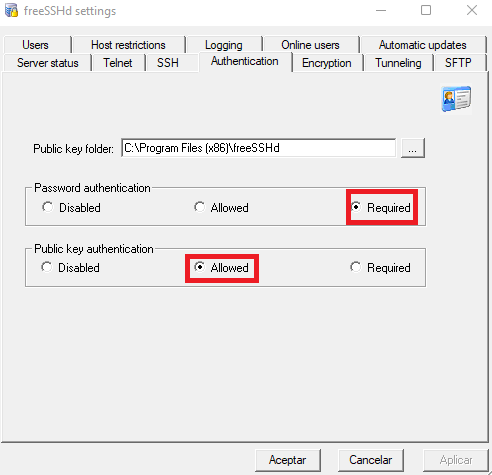


Figura N° 19 Configuración del servidor SSH - Sección Authentication

**Sección Encryption:** Seleccionar la encriptación AES256

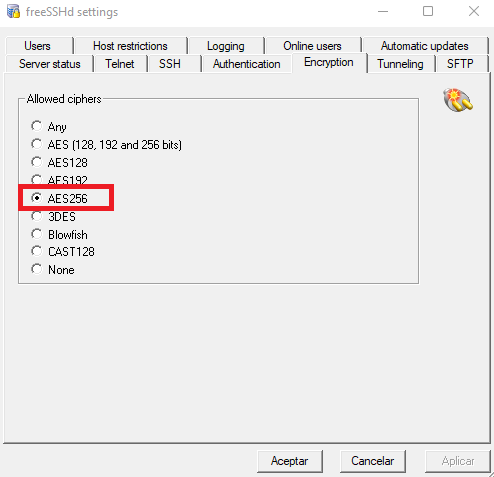


Figura N° 20 Configuración del servidor SSH - Sección Encryption

**Sección Users:** Al agregar el usuario seleccionar las opciones Shell y SFTP.

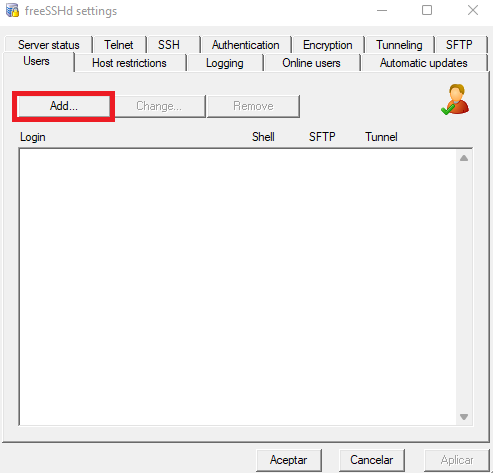


Figura N° 21 Configuración del servidor SSH - Sección Users

**Sección Server Status:** Iniciar la el servicio de SSH



Figura N° 22 Configuración del servidor SSH - Sección Server status

## Ejecución del proyecto desde Putty

Ingresar el IP Address del servidor y puerto configurado en freeSSH.

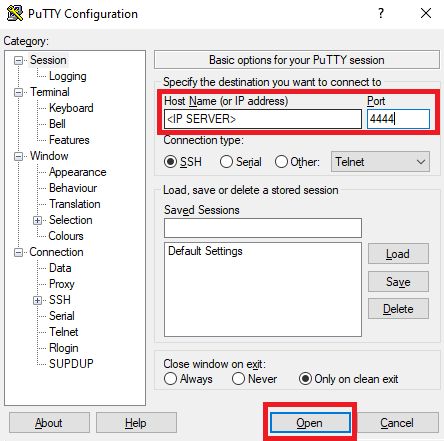


Figura N° 23 Putty Configuration

Iniciar sesión con las credenciales creadas en freeSSHd

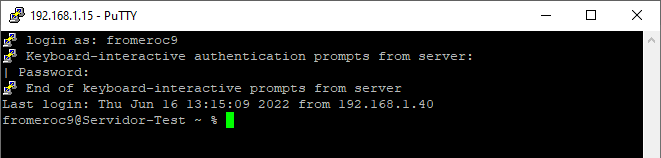


Figura N° 24 Inicio de sesión con Putty al servidor remoto

Movernos al directorio donde está el proyecto y ejecutar el comando \*mvn test

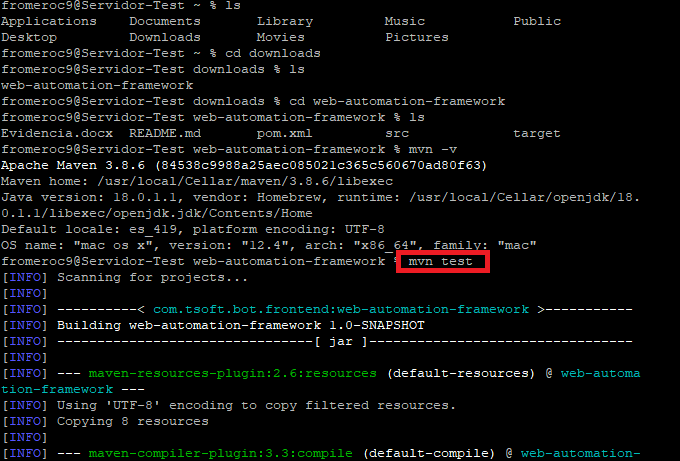


Figura N° 25 Ejecución del proyecto

# Desarrollo de nuevos scripts de automatización

## Proceso

Se sugiere el siguiente proceso a seguir cuando se tienen que automatizar nuevos casos de prueba:



Figura N° 26 Proceso a seguir para automatizar nuevos casos

La salida de los sub-procesos de preparación y análisis de desarrollo generan una estimación, la cual es una entrada para la realización del plan de automatización.

Cuando el plan de automatización es aprobado por NIUBIZ, se continúa con las etapas de implementación, customización e integración.

## Requerimientos de configuración y Herramientas

1. JAVA: Se necesita tener instalado el JDK, de preferencia la versión 8.

Comando para verificar la instalación de java:

\* java –versión

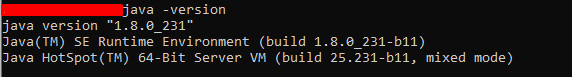


Figura N° 27 Verificar la instalación de JAVA

1. MAVEN

Descargar maven, de preferencia la versión 3.6. Descomprimir y copiar la carpeta en el Disco C:\Program Files

Comando para verificar la instalación de maven:

\* mvn –versión

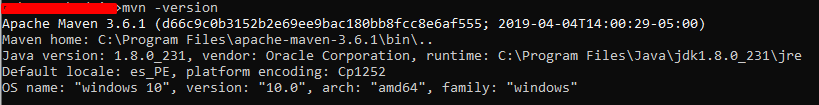


Figura N° 28 Verificar la instalación de MAVEN

1. VARIABLES DE ENTORNO

Configurar las variables de entorno: **JAVA\_HOME** y **MAVEN\_HOME**

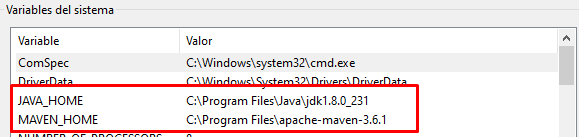


Figura N° 29 Variables de entorno del sistema configuradas (JAVA y MAVEN)

1. IDE: Instalación del IDE IntelliJ IDEA.



Figura N° 30 IDE IntelliJ IDEA

1. GIT

Para poder acceder a la última versión del proyecto, deberá contar con git instalado en su PC. Verificar la instalación de git con el siguiente comando:

\* git --version



Figura N° 31 Verificar la instalación de git

## Desarrollo de nuevos scripts

### Acceso al repositorio.

Los pasos a seguir son los siguiente:

**Nota:** Si no cuenta con usuario en GoGs, solicitar al administrador del GoGs, la creación de un usuario, así mismo, se añada el nuevo usuario a la organización y permita el acceso al repositorio del proyecto.

1. Para descargar el proyecto hay 2 formas:
2. Clonando el repositorio:

git clone <http://172.16.10.91:3000/NIUBIZ/test-automation.git>

Realizando un pull al Proyecto

git remote add origin <http://172.16.10.91:3000/NIUBIZ/test-automation.git>

git pull origin master

1. Importar el Proyecto en el IDE de su preferencia.

### Creación de los features

Para el desarrollo de los escenarios se debe conocer las sintaxis del lenguaje Gherkins. Se sugiere revisar el siguiente link: <https://cucumber.io/docs/gherkin/reference/>

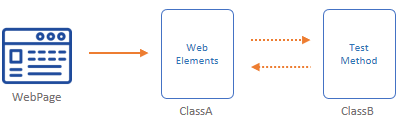
Descripción de un escenario de prueba:



Figura N° 32 Descripción de un caso de prueba

* 1. **Palabras clave:** Feature, Scenario Outline, Given, When, Then y Examples.
  2. **Tag:** Etiquetas que permiten identificar a los escenarios de prueba al momento de ejecución. Cada escenario debe tener mínimo 2 tags, uno de regresión y otro que indique el nombre del escenario.
  3. **Nombre del feature:** Descripción de alto nivel de los escenarios relacionados.
  4. **Nombre Escenario:** Nombre del caso de prueba o flujo que se automatizará. Un escenario puede incluir la validación de varios casos de prueba.
  5. **Filas:** Cada fila y número indican la cantidad de filas que se leerán desde el Excel.

# Glosario

* **POM – Page Object Model**: Es un patrón de diseño permite mantener las pruebas y los localizadores de elementos web por separado. Es decir, este modelo indica que, para cada página web en la aplicación, debería existir una clase de Page correspondiente, la cual contiene objetos de los elementos web de la interfaz de usuario.
* 
* Figura N° 33Patrón de Diseño POM
* **BDD - Behavior Driven Development**: es una estrategia de desarrollo que plantea definir un lenguaje común para el negocio y para los técnicos, y utilizar eso como parte inicial del desarrollo y el testing.
* **Cucumber:** Es una herramienta que permite el desarrollo dirigido por el comportamiento (BDD). Ofrece una forma de escribir pruebas en alto nivel, permitiendo que cualquiera pueda comprenderlas, independientemente de sus conocimientos técnicos. El lenguaje que se usa es Gherkins y los archivos en los cuales se describen los pasos del caso de prueba tienen la extensión **“.feature”**.
* **Gherkin:** es un lenguaje comprensible por humanos y por ordenadores, con el que vamos a describir las funcionalidades, definiendo el comportamiento del software, sin entrar en su implementación. Se trata de un lenguaje fácil de leer, fácil de entender y fácil de escribir.
* **Extent Reports:** es un informe HTML personalizable que se puede integrar en Selenium WebDriver utilizando los marcos JUnit y TestNG.
* **Apache POI:** es una API que permite crear, modificar y mostrar archivos de MS-Office utilizando programas Java.
* **Hook:** son bloques de código que pueden ejecutarse en varios puntos del ciclo de ejecución de Cucumber. Por lo general, se utilizan para la configuración y desmontaje del entorno antes y después de cada escenario.

|  |  |
| --- | --- |
| Autor del Documento: | Abraham Rivera Rivadeneyra |
| Fecha de Creación: | 15/06/2022 |