

**Prueba de Concepto**

# Visión General

* 1. **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:

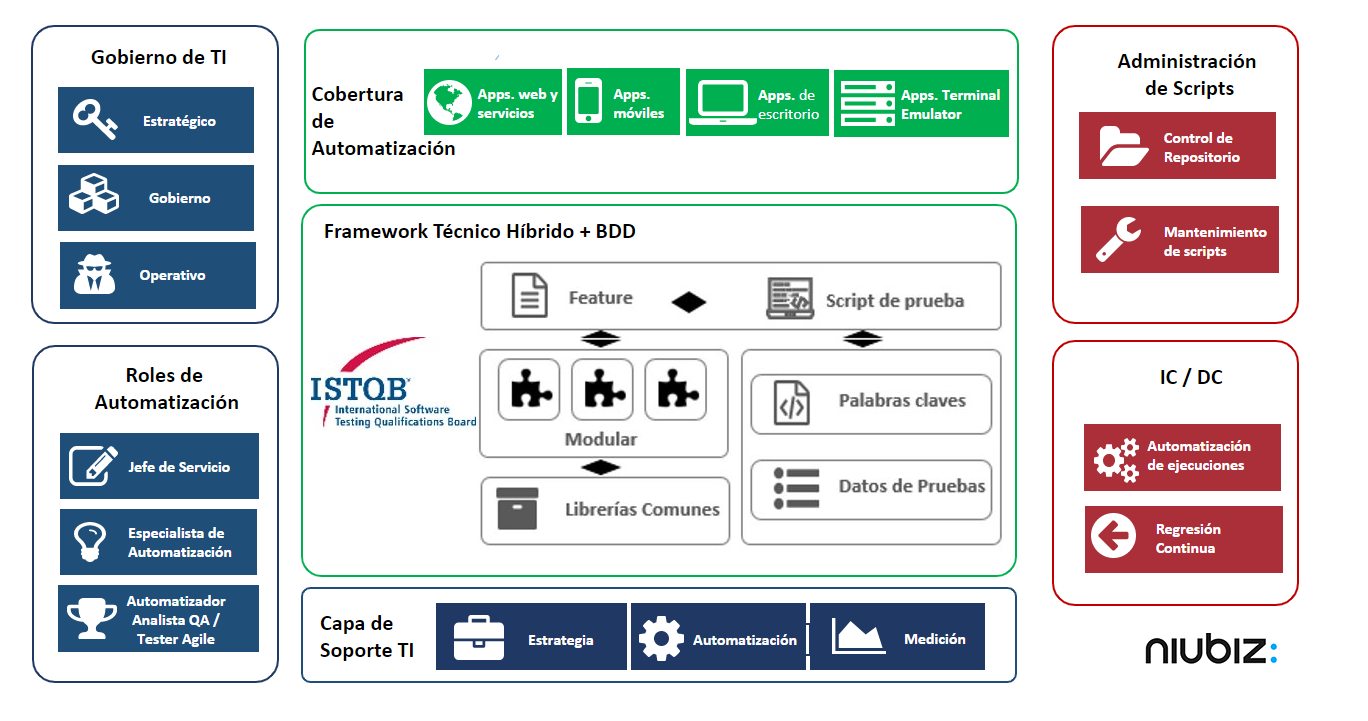
1. Vende Más
   1. **Introducción**

Este proyecto abarca a la utilización del Framework Selenium + Appium para las pruebas de testing, creando producto útil y usable para desarrollar diferentes tipos de pruebas.

Un framework de automatización nos permite tener un conjunto de pautas o reglas para la generación de casos de prueba y estandarizar la ejecución de pruebas UI.

El propósito de crear esta la estructura de trabajo es apoyar a los QA cuando empiecen a automatizar, permitiendo las creaciones de scripts para brindar soluciones.

* 1. **Framework Técnico**



1. Uso de MAVEN para las dependencias y librerías.
2. Uso de PAGE OBJECT MODEL, para la reutilización de componentes.
3. Uso de CUCUMBER (Gherkin) para la capa usuario y entender el proceso de automatización.
4. Uso del lenguaje de programación JAVA para la construcción de los scripts.
5. Uso de KEYWORD DRIVEN, para la asignación de diversas acciones dentro de la ejecución.
6. Uso de EXCEL para el manejo de los datos a nivel usuario ejecutor
   1. **Tipos de Framework**

**Data Driven:** Son aquellos en que los datos requeridos para la ejecución de las pruebas​.

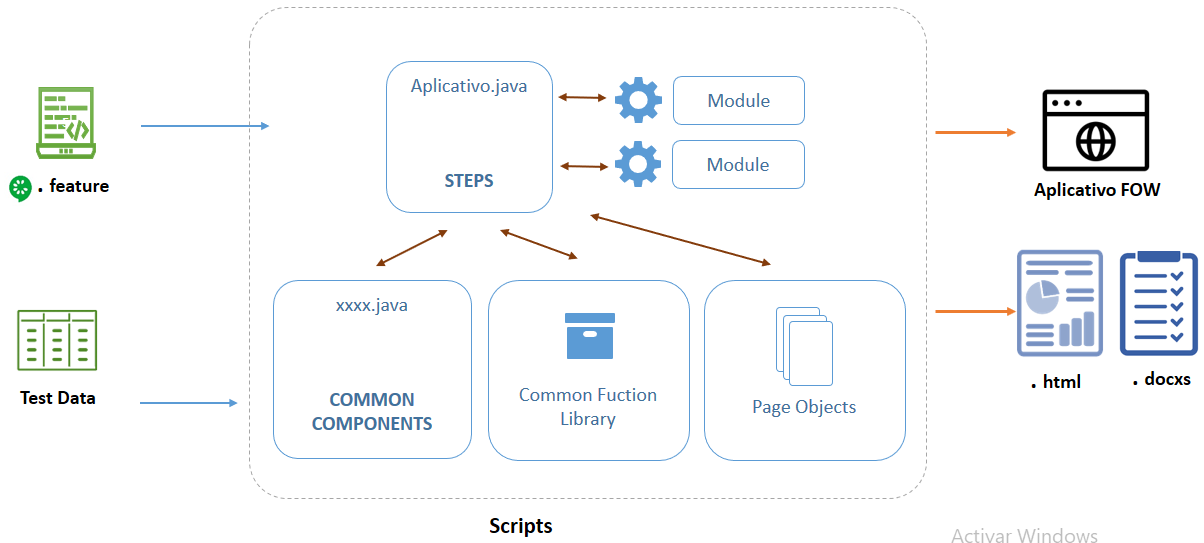
**BDD:** Utilizan lenguaje natural, no técnico, para la especificación de las pruebas. ​

**Keyword driven:** Identifican una serie de palabras clave y luego se asigna cada una de éstas a una serie de acciones​.

**Modular:** Esta separación nos permite una mejor mantenibilidad y actualización de las pruebas. ​

**Híbrido:** Combinación de diversos frameworks para optimización.

* 1. **Diseño de Framework**



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, 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.

* 1. **Temas Investigados**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cucumber | BDD | Modular | TestNG | Page Object Model | Appium driver |
| Locator | Data Driven Testing | Keyword drive | Plugins | Extent Reports |  |

* 1. **Requisitos Previos**

1. Descargar, instalar y configurar JDK: <https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html>
2. Descargar y configurar Maven.

<https://maven.apache.org/download.cgi>

1. Descargar Android SDK

<https://developer.android.com/studio>

1. Descargar e instalar Node JS

<https://nodejs.org/es/download/>

1. Descargar e instalar Appium Desktop

<https://appium.io/downloads.html>

# Documentación

* 1. **Términos y Definiciones**

1. **AAPg:** Arquitectura de Automatización de Prueba Genérica
2. **SAP:** Sistema de Automatización de Pruebas
3. **SSP:** Sistema Sujeto a Prueba.
4. **GUI:** Interfaz Gráfica de Usuario
5. **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.
6. **Escenario de Prueba:** Para los casos de prueba del aplicativo Vende Mas, cada escenario de prueba dentro del feature representa a un caso de prueba.
7. **Aplicativo Conocido:** Aplicativo el cual tiene algunos casos de prueba ya automatizados.
8. **Aplicativo Nuevo:** Aplicativo el cual no tiene casos de prueba automatizados.
   1. **Gobiernos 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: (1) A Nivel Estratégico, (2) De Gobierno, y (3) 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.
   1. **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.
  1. **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 Movil | Selenium / Appium / JAVA |

* 1. **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) Librerías 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 cual permite sumar a los beneficios lo siguiente:

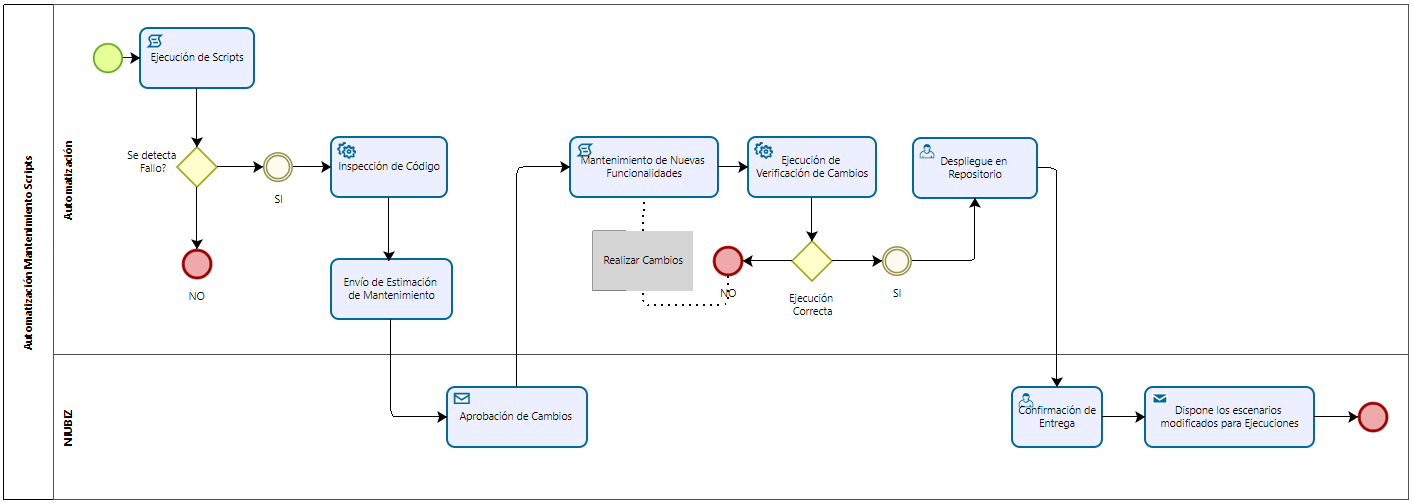
* + - Fácil entendimiento, permite incluir en la automatización el escenario de manera escrita (lenguaje Gherkin) lo que permite que cualquier rol entienda el comportamiento que tendrá el escenario automatizado al ejecutarse.
  1. **Capa de Soporte de TI**

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.
  1. **Administración de Script**

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

1. **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
2. **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
3. **Estrategia de Mantenimiento:**



1. Como mantenimiento preventivo cada quince días, se realizará una ejecución de los scripts automatizados y verificar si esta falla 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 realiza la recopilación de evidencias para preparar y enviar 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.
   1. **Acerca del repositorio**

#### Datos del repositorio:

#### Nombre Repositorio: framework-appium

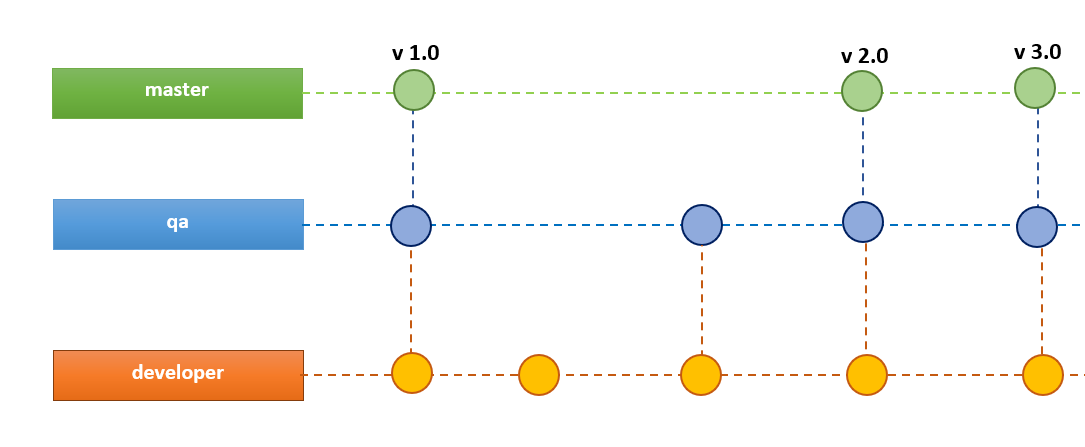
#### Características del repositorio

#### 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
2. Estrategia de Ramas:

**Repositorio**: framework-appium

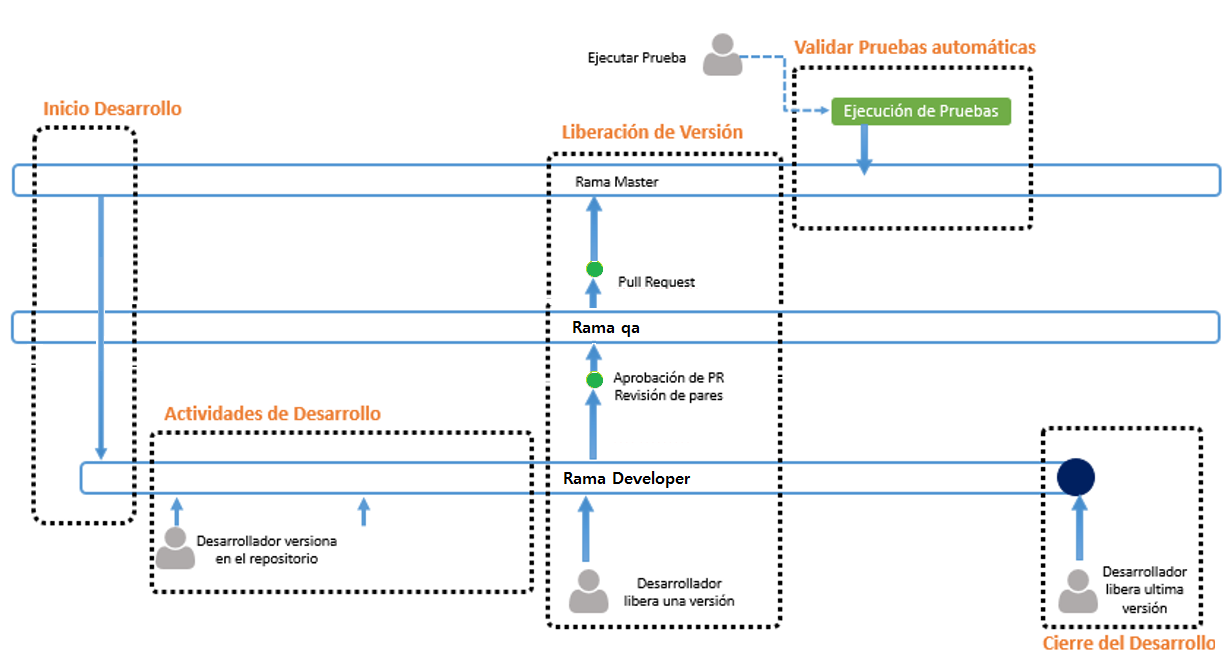
* **master**: Branch contenedor de las versiones finales de los flujos automatizados.
* **qa**: Branch contenedor de las versiones verificadas por el equipo de calidad.
* **developer**: Branch contenedor de las versiones desarrollados por el equipo de automatización.



* 1. **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.

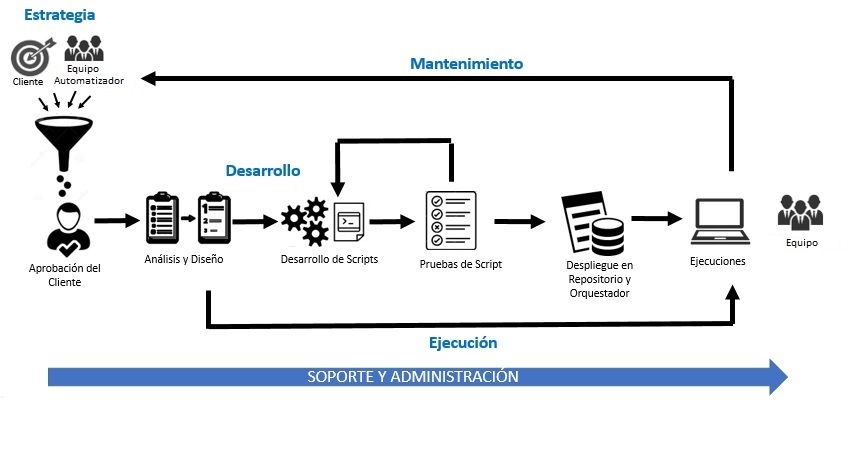


* 1. **Entrega Continua**

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

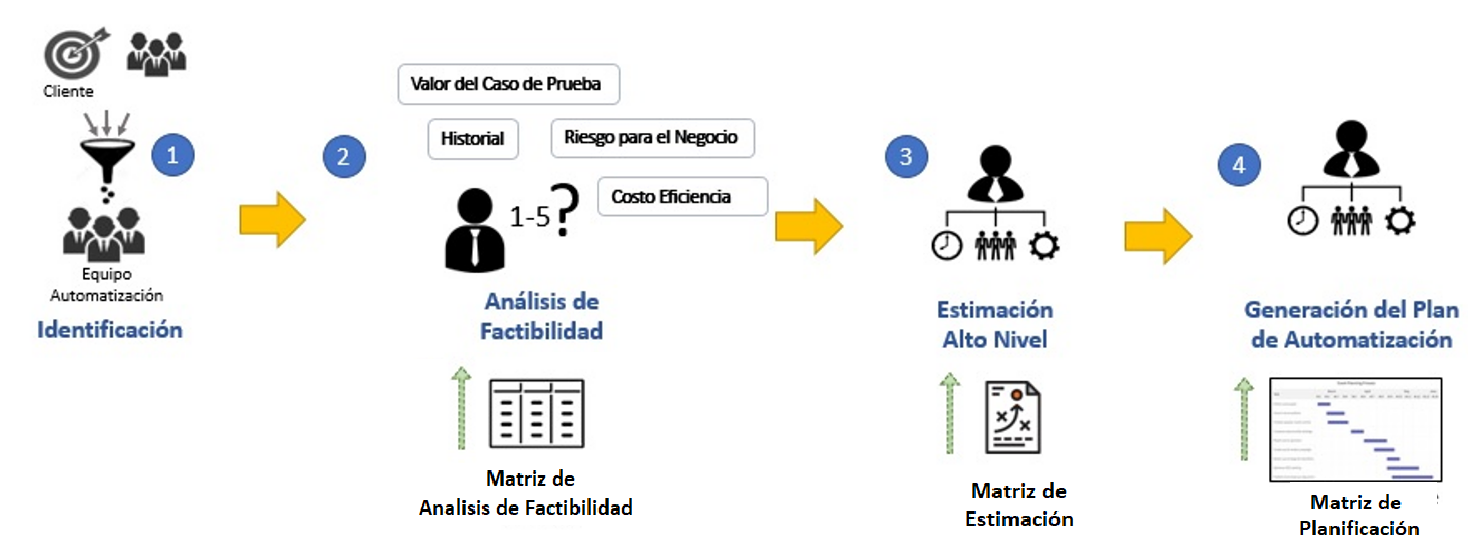
La estrategia de automatización que plantea el gobierno de TI es la siguiente:



* 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.



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**

**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 automatización. Luego se hará de conocimiento al cliente para su aprobación y priorización dentro del backlog del equipo.

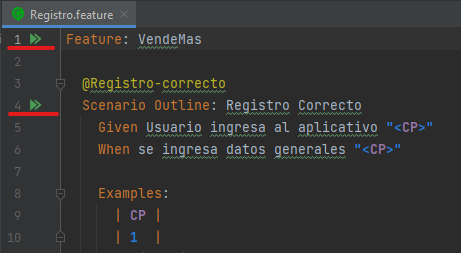
* 1. **Soporte y Administración**

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

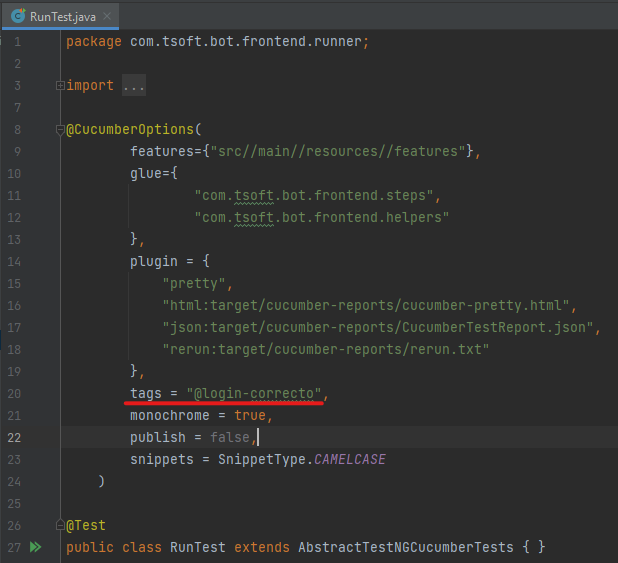
# Empezar Proceso

* 1. **Ejecución GUI**

Podemos ejecutar los diferentes escenarios por separado o inicializar todo el. feature para su ejecución.

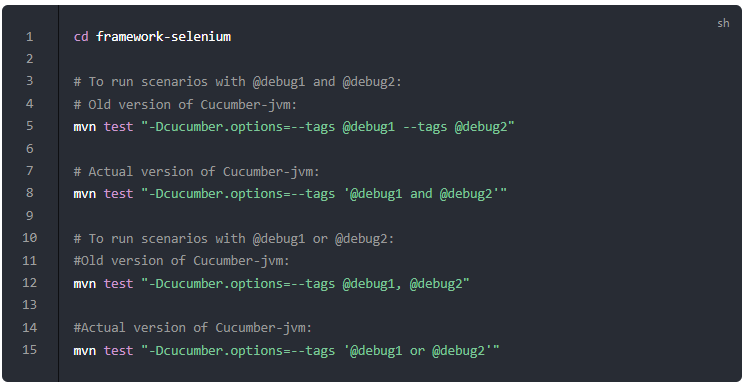


Podemos ejecutar desde la clase RunTest especificando el tag correspondiente de cada feature.



* 1. **Ejecución CLI**

Similar que ejecutar appium por GUI, pero con comando en la terminal favorita.



* 1. **Estructura de Proyecto**

Estructura de carpeta del framework trabajado.

**Global**

|  |  |
| --- | --- |
| **Carpeta** | **Descripción** |
| src/main | Carpeta principal del proyecto |
| target | Carpeta donde se genera los reportes html, documento de word |

**Java/Fronted**

|  |  |
| --- | --- |
| **Carpeta** | **Descripción** |
| base | Especializado para guardar acciones, el automatizador puede crear nuevas acciones de la API de Selenium |
| collections | Carpeta especializada para las pruebas de API |
| helpers | Especializado para la configuración del driver browser a utilizar en las pruebas |
| listener | Almacena las configuraciones antes, durante y después de la ejecución de las pruebas |
| pages | Especializado para contener funcionalidad de los steps a desarrollar |
| steps | Especializado para almacenar los pasos a desarrollar del feature a nivel funcional |
| runner | Especializado para la configuración de los casos a ejecutar |
| utily | Almacena diferentes herramientas necesarias que el framework necesita como lectura de excel, creación de reportes, etc |

**Java/Resource**

|  |  |
| --- | --- |
| **Carpeta** | **Descripción** |
| config | Configuraciones especiales del testng y extent report |
| excel | Especializado para almacer los archivos excel |
| feature | Especializado para la creación de .feature (escenarios en Gherkin) |
| template | Configuraciones de plantilla para la generación del Word |
| Log4j2.properties | Configuraciones de log4j |

**Target**

|  |  |
| --- | --- |
| **Carpeta** | **Descripción** |
| generated-sources | Genera diferentes anotaciones |
| resultado | Almacena las capturas de pantalla y genera el reporte html |

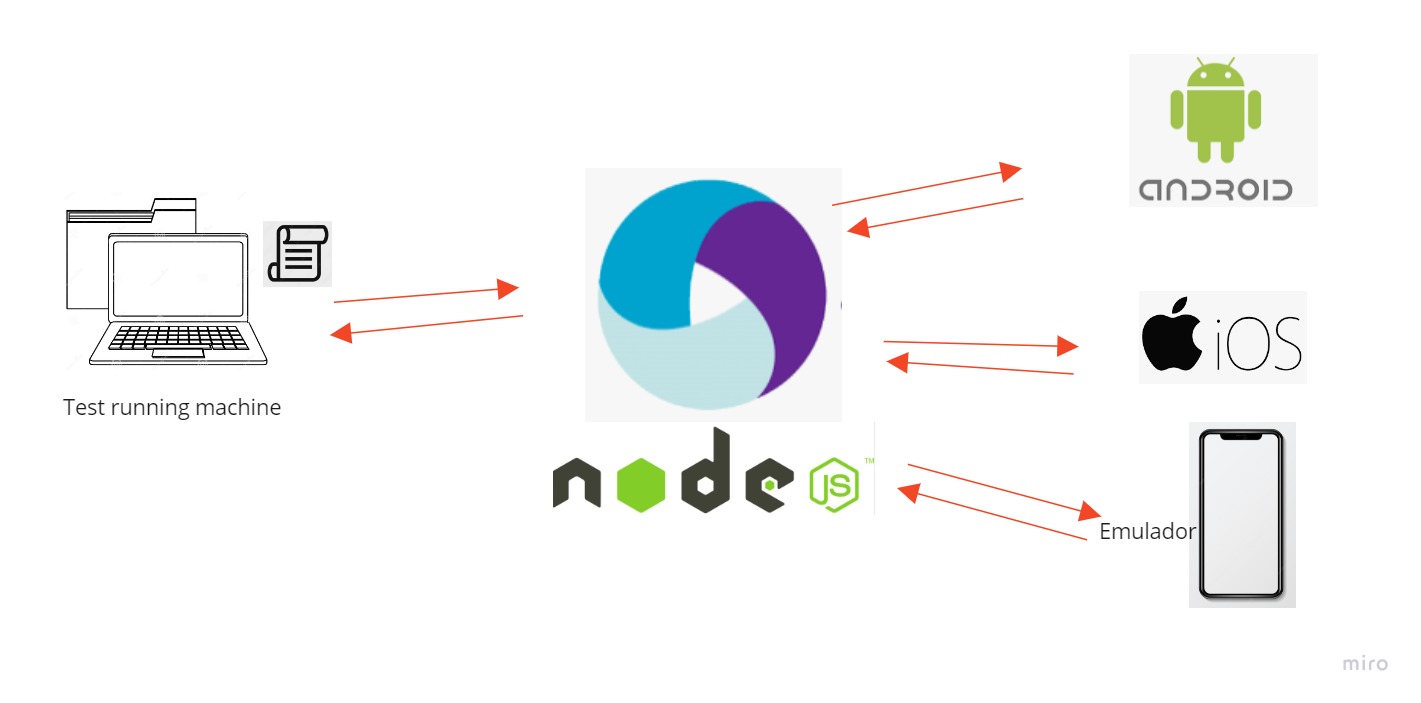
* 1. **Plugins**

|  |  |
| --- | --- |
| **Dependencias** | **Descripción** |
| JSON | JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos, y lo usamos más comúnmente para la comunicación cliente-servidor. |
| Extent Reports | Informes de análisis en tiempo real para que pueda ver sus pruebas de una manera totalmente diferente. |
| Cucumber | Defina los archivos de características con todas sus partes interesadas mediante el desarrollo basado en el comportamiento (BDD) |
| Apache POI | Apache POI es crear y mantener API Java para manipular varios formatos de archivo basados en los estándares Office Open XML (OOXML) y el formato OLE 2 Compound Document de Microsoft (OLE2). |
| Selenium | Si desea crear suites y pruebas de automatización de regresión robustas y basadas en navegador, escalar y distribuir scripts en muchos entornos |
| TestNG | TestNG es un marco de pruebas inspirado en JUnit y NUnit pero que introduce algunas funcionalidades nuevas que lo hacen más potente y fácil de usar |

# Prueba Móvil

Appium es de código abierto para automatizar aplicaciones móviles nativas, web, híbridas en plataformas móviles IOS, Android y Windows.

Appium emplea el WebDriver de Selenium, que es un entorno de pruebas para aplicaciones web y que especifica un protocolo cliente-servidor conocido como JSON Wire Protocol. Esto permite al cliente el uso de marcos de pruebas escritos en cualquier lenguaje, enviando las solicitudes HTTP apropiadas al servidor.



Appium permite comprobar automáticamente las propiedades y el comportamiento de los distintos elementos (botones, cajas de texto, etc) que componen la interfaz gráfica de una aplicación. Esto es posible mediante el uso de herramientas que inspeccionan los elementos de la aplicación como, por ejemplo, UIAutomatorViewer o el propio inspector de Appium, y a través de métodos de Selenium como FindElement, By, etc.

* 1. **Creación de Feature**

**Repositorio\_principal:** <https://bitbucket.org/VisaNet_TI/framework-appium/src/developer/>

Ubicación: src/main/resources/features

Feature: VendeMas  
  
 @login-correcto  
 Scenario Outline: Login Correcto  
 Given Usuario ingresa login del aplicativo **"<CP>"** When para acceder en vendemas **"<CP>"** Examples:  
 | **CP** |  
 | **1** |  
 | **2** |  
 | **3** |

Feature: VendeMas  
  
 @Registro-correcto  
 Scenario Outline: Registro Correcto  
 Given Usuario ingresa al aplicativo **"<CP>"** When se ingresa datos generales **"<CP>"** Examples:  
 | **CP** |  
 | **1** |  
 | **2** |  
 | **3** |

* 1. **Armar los steps**

Ubicación: src/main/java/fronted/steps

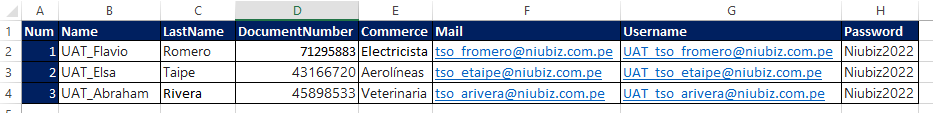
package com.bot.frontend.steps;  
  
import com.bot.frontend.pages.components.LoginPage;  
import com.bot.frontend.pages.components.MainPage;  
import io.appium.java\_client.AppiumDriver;  
import io.appium.java\_client.MobileElement;  
import io.cucumber.java.en.Given;  
import io.cucumber.java.en.When;  
  
public class LoginStep {  
  
 @Given("Usuario ingresa login del aplicativo {string}")  
 public void usuarioIngresaLoginDelAplicativo(String cp) throws Throwable {  
   
 }  
  
 @When("para acceder en vendemas {string}")  
 public void paraAccederEnVendemas(String cp) throws Throwable {  
   
 }  
}

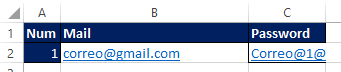
package com.bot.frontend.steps;  
  
import com.bot.frontend.pages.components.LoginPage;  
import com.bot.frontend.pages.components.MainPage;  
import com.bot.frontend.pages.components.RegistroPage;  
import io.appium.java\_client.AppiumDriver;  
import io.appium.java\_client.MobileElement;  
import io.cucumber.java.en.Given;  
import io.cucumber.java.en.When;  
  
public class RegistroStep {  
  
 private AppiumDriver<MobileElement> driver;  
 private RegistroPage lp = new RegistroPage(driver);  
 //private MainPage mp = new MainPage(driver);  
  
  
 @Given("Usuario ingresa al aplicativo {string}")  
 public void usuarioIngresaAlAplicativo(String cp) throws Throwable {  
 lp.RegistroLogin(cp);  
  
 }  
  
 @When("se ingresa datos generales {string}")  
 public void seIngresaDatosGenerales(String cp) throws Throwable {  
 lp.validarRegistro(cp);  
  
 }  
}

* 1. **Armar la data de prueba**

Ubicación: src/main/resources/excel

Como vamos a probar el inicio de sesión del aplicativo necesito los siguientes datos

****

****

* 1. **Armar los Objects Page**

En este ejemplo debemos armar los object del sitio web y de la data en Excel

**Excel Objects**

Ubicación: src/main/java/fronted/pages/objects

Se debe indicar que tipo de valor vas a recibir o estar declarando en el excel.

public class UtilsExcel {  
  
 public String EXCEL\_DOC = "excel/data.xlsx";  
 public String PAGE\_SHEET1 = "LOGIN";  
 public String PAGE\_SHEET2 = "REGISTRO";  
}

public class LoginExcel {  
  
 public String USERNAME = "Username";  
 public String PASSWORD = "Password";  
   
}

public class RegistroExcel {  
  
 public String PASSWORD = "Password";  
 public String MAIL = "Mail";  
}

**Web Locator**

Ubicación: *src/main/java/fronted/pages/objects*

Por cada página que se va navegando se va creando nuevo archivo ts para guardar los locator. Si desea trabajar todo en un solo archivo para guardar todos sus locator solo recordar el orden.

* 1. **Secuencia del Flujo Programado**

package com.bot.frontend.pages.objects;  
  
import org.openqa.selenium.By;  
  
public class MainObject {  
  
 public By BTN\_MENU = By.*xpath*("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/androidx.compose.ui.platform.ComposeView/android.view.View/android.view.View/android.view.View[1]/android.widget.Button[1]");  
 public By MENU\_EMAIL\_USER = By.*xpath*("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/androidx.compose.ui.platform.ComposeView/android.view.View/android.view.View[2]/android.widget.ScrollView/android.view.View[1]/android.widget.TextView[3]");  
 public By BTN\_CLOSE = By.*xpath*("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/androidx.compose.ui.platform.ComposeView/android.view.View/android.view.View[2]/android.widget.ScrollView/android.view.View[12]/android.widget.TextView");  
 public By BTN\_CLOSE\_ACEPTAR = By.*xpath*("/hierarchy/android.widget.FrameLayout/android.widget.FrameLayout/android.widget.FrameLayout/android.view.ViewGroup/android.view.View/android.view.View/android.view.View/android.view.View[2]/android.widget.Button");  
  
}

package com.bot.frontend.pages.objects;  
  
public class LoginObject {  
  
 public By BTN\_ACCEDER = By.*xpath*("//android.widget.TextView[contains(@text,'Ya estoy registrado')]");  
 public By INPUT\_CORREO\_ELECT = By.*xpath*("//android.widget.EditText[1]");  
 public By INPUT\_PASSWORD = By.*xpath*("//android.widget.EditText[2]");  
 public By BTN\_INICIAR\_SESION = By.*xpath*("/hierarchy/android.widget.FrameLayout/android.widget.LinearLayout/android.widget.FrameLayout/androidx.compose.ui.platform.ComposeView/android.view.View/android.view.View/android.view.View[2]/android.widget.Button");  
  
}

package com.bot.frontend.pages.objects;  
  
import org.openqa.selenium.By;  
  
public class RegistroObject {  
  
 public By BTN\_REGISTRAR = By.*xpath*("//android.widget.TextView[contains(@text,'Registrarme')]");  
 public By INPUT\_CORREO\_RE = By.*xpath*("//android.widget.EditText[1]");  
 public By INPUT\_PASSWORD\_RE = By.*xpath*("//android.widget.EditText[2]");  
 public By CHECK\_TERMINOS = By.*xpath*("//android.widget.CheckBox");  
 public By BTN\_REGISTRO = By.*xpath*("//android.widget.TextView[contains(@text,'Siguiente')]");  
  
}

* 1. **Secuencia de Steps**

1. package com.bot.frontend.steps;  
     
   import com.bot.frontend.pages.components.LoginPage;  
   import com.bot.frontend.pages.components.MainPage;  
   import io.appium.java\_client.AppiumDriver;  
   import io.appium.java\_client.MobileElement;  
   import io.cucumber.java.en.Given;  
   import io.cucumber.java.en.When;  
     
   public class LoginStep {  
     
    private AppiumDriver<MobileElement> driver;  
    private LoginPage lp = new LoginPage(driver);  
    private MainPage mp = new MainPage(driver);  
     
    @Given("Usuario ingresa login del aplicativo {string}")  
    public void usuarioIngresaLoginDelAplicativo(String cp) throws Throwable {  
    lp.AccederLogin(cp);  
    }  
     
    @When("para acceder en vendemas {string}")  
    public void paraAccederEnVendemas(String cp) throws Throwable {  
    lp.validarLogin(cp);  
    mp.validarUsuario(cp);  
    mp.finalizaSesion(cp);  
    }  
   }

package com.bot.frontend.steps;  
  
import com.bot.frontend.pages.components.LoginPage;  
import com.bot.frontend.pages.components.MainPage;  
import com.bot.frontend.pages.components.RegistroPage;  
import io.appium.java\_client.AppiumDriver;  
import io.appium.java\_client.MobileElement;  
import io.cucumber.java.en.Given;  
import io.cucumber.java.en.When;  
  
public class RegistroStep {  
  
 private AppiumDriver<MobileElement> driver;  
 private RegistroPage lp = new RegistroPage(driver);  
   
 @Given("Usuario ingresa al aplicativo {string}")  
 public void usuarioIngresaAlAplicativo(String cp) throws Throwable {  
 lp.RegistroLogin(cp);  
  
 }  
  
 @When("se ingresa datos generales {string}")  
 public void seIngresaDatosGenerales(String cp) throws Throwable {  
 lp.validarRegistro(cp);  
  
 }  
}