**Tabla De Contenido**

[1. OBJETIVO 2](#_Toc152593866)

[2. GLOSARIO 2](#_Toc152593867)

[3. CONDICIONES GENERALES 4](#_Toc152593868)

[4. MODELO DE ARQUITECTURA GENERAL DE PRUEBAS AUTOMATIZADAS 5](#_Toc152593869)

[5. DETALLE DE CAPAS PROCEDIMENTALES 7](#_Toc152593870)

[Capa de Negocio 7](#_Toc152593871)

[Capa de Definición / Definition 9](#_Toc152593872)

[Capa Page/Driver (Steps) 11](#_Toc152593873)

[5.1 Tipo de Automatización 12](#_Toc152593874)

[5.1.a Frontend 12](#_Toc152593875)

[5.1.b Backend 14](#_Toc152593876)

[5.2 Herramientas Recomendadas 15](#_Toc152593877)

[5.3 Mantenibilidad 15](#_Toc152593878)

[6. OBTENCIÓN DE DATA 16](#_Toc152593879)

[7. MANEJO DE CONTROL DE VERSIONES 17](#_Toc152593880)

[8. BUENAS PRACTICAS DE AUTOMATIZACION DE PRUEBAS 17](#_Toc152593881)

[9. BUENAS PRACTICAS PARA CREACIÓN DE CASOS AUTOMATIZADOS 19](#_Toc152593882)

# OBJETIVO

El objetivo principal de este instructivo es proporcionar una guía detallada y estandarizada para la implementación eficaz de herramientas y el framework de automatización en las pruebas funcionales de aplicaciones web, abarcando tanto la interfaz de usuario (UI) como las interfaces de programación de aplicaciones (API). Este documento tiene como propósito servir como recurso integral para los equipos de desarrollo, facilitando la adopción y uso coherente de las herramientas de automatización en todos los proyectos. Se busca agilizar y mejorar el proceso de aseguramiento de calidad, garantizando la consistencia y eficiencia en cada fase del desarrollo de software.

# GLOSARIO

**Procesos Generales**

El proceso base de Automatización de pruebas comprende desde la implementación hasta la operación de esta.

|  |  |  |
| --- | --- | --- |
| **Grupo / Capa / Criterio** | **Criterio de segundo nivel** | **Lineamiento** |
| Proceso general | Necesidad de Automatizar | Se debe iniciar el proceso con una necesidad de Automatización identificada. |
| Proceso general | Validación de Aplicabilidad | Se debe realizar una validación de aplicabilidad utilizando los lineamientos de Aplicabilidad descritos. |
| Proceso general | Validación flujo de negocio | Se debe determinar si el flujo funcional de negocio es automatizable. |
| Proceso general | Validación Técnica | Se debe determinar si la(s) tecnología(s) a automatizar dentro del flujo de negocio tiene soporte con herramienta de Automatización. |
| Proceso general | Costo/Beneficio | Se debe realizar el análisis de Costo Beneficio de la Automatización. |
| Proceso general | Gestión de Pruebas (Test Management) | Se debe crear y agrupar el modelado de negocio en archivos extensión feature |
| Proceso general | Modelado de Negocio (Business) | Se debe crear el modelado de negocio de acuerdo con los lineamientos establecidos en la Arquitectura de pruebas automatizadas |
| Proceso general | Modelado de Negocio (Data Driven) | Se debe desarrollar el modelado de datos de acuerdo con los lineamientos establecidos en la Arquitectura de pruebas automatizadas |
| Proceso general | Modelado Técnico (Definición) | Se debe desarrollar el modelado técnico en definición de acuerdo con los lineamientos establecidos en la Arquitectura de pruebas automatizadas |
| Proceso general | Modelado Técnico (Driver) | Se debe desarrollar el modelado técnico en controlador de acuerdo con los lineamientos establecidos en la Arquitectura de pruebas automatizadas |
| Proceso general | Ejecución | La ejecución se debe establecer de acuerdo con la necesidad de implementación haciendo uso de las opciones establecidas en el lineamiento de la Arquitectura de pruebas automatizadas. |
| Proceso general | Resultados | Los resultados de ejecución de pruebas deben contener el resumen y las evidencias de soporte definida en los pasos anteriores del proceso. |

|  |  |
| --- | --- |
| **Termino** | **Definición** |
| Azure DevOps | Es un conjunto integral de servicios y herramientas en la nube proporcionado por Microsoft para respaldar el ciclo de vida completo del desarrollo y la entrega de software. Diseñado para equipos de desarrollo y operaciones, Azure DevOps ofrece una plataforma unificada que abarca diversas funcionalidades, incluyendo gestión de código fuente, seguimiento de problemas, construcción y compilación automatizada, pruebas, implementación continua, y monitoreo. |
| Despliegue Continuo (CI) | CI "Integración Continua" (Continuous Integration en inglés), es una práctica en el desarrollo de software que se enfoca en la integración frecuente de cambios en el código fuente de un proyecto por parte de varios desarrolladores. La integración continua implica la automatización de las pruebas y la construcción del software, lo que permite identificar y solucionar problemas de manera más rápida y eficiente. |
| Selenium WebDriver | Herramienta open source comprendida por un conjunto de APIs que permiten la automatización de navegadores web. WebDriver permite interactuar con elementos de la página, realizar acciones como hacer clic en botones y rellenar formularios, y verificar el contenido de la página. |
| Serenity | Es un marco de automatización de pruebas de código abierto diseñado para simplificar y mejorar la automatización de pruebas en aplicaciones web y móviles. Serenity, a menudo conocido como "Serenity BDD" (Behavior-Driven Development), se centra en la escritura de pruebas de comportamiento en un lenguaje claro y legible por humanos, lo que facilita la colaboración entre equipos de desarrollo y negocio. |
| Cucumber | Es una herramienta de automatización de pruebas y una metodología que se utiliza para la automatización de pruebas de aceptación basadas en comportamiento (BDD, por sus siglas en inglés, Behavior-Driven Development). Cucumber permite a los equipos de desarrollo y pruebas escribir pruebas en un lenguaje natural y comprensible por todas las partes interesadas, incluidos desarrolladores, probadores y personas no técnicas, como representantes de negocio y clientes. |
| RestAssured | Es una biblioteca de automatización de pruebas diseñada específicamente para realizar pruebas en servicios web RESTful. Se trata de una herramienta de código abierto en Java que facilita la creación de pruebas de integración y pruebas de API (Interfaz de Programación de Aplicaciones) al simplificar las solicitudes HTTP y las validaciones de respuestas. |

# CONDICIONES GENERALES

Para llevar a cabo este instructivo se debe tener en cuenta las siguientes condiciones antes de empezar a desarrollarlo:

* Se debe contar con un Project y al menos un Team en Azure DevOps del producto que se requiere probar, también deben estar configurados los usuarios que van a realizar las pruebas en el Team de Azure DevOps.

Responsables de cada actividad:

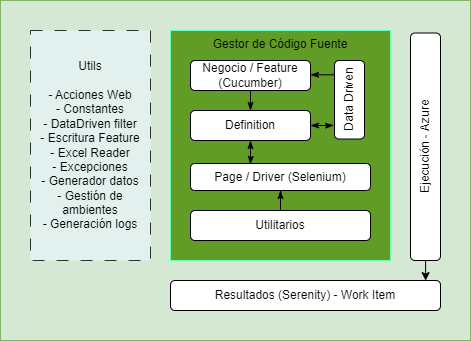
|  |  |
| --- | --- |
| **Actividad** | **Responsable** |
| Crear proyecto y usuarios en Azure DevOps  Debe ser de conocimiento del equipo de pruebas. | Equipo DevOps |
| Crear la automatización del proyecto asociado basado en el framework entregado | Equipo de Desarrollo/Pruebas |
| Crear pipeline en Azure DevOps con la integración de la automatización |
| Análisis de resultados |
| Manejo de los issues |

**Repositorio y Cargue de Archivos**

A continuación, se conocerá el estándar de nomenclatura y directorios que se deben seguir en el cargue de los archivos necesarios para la ejecución de la prueba en el repositorio.

**Repositorio:** El repositorio para los desarrollos con automatización de pruebas tendrá el mismo nombre del repositorio que se creó para el desarrollo con la palabra “Automation” (Ej: *Proyecto\_Automation*) al final de su nombre, lo que permitirá diferenciarlos. El desarrollo de funcionalidades y cobertura de escenario de pruebas debe realizarse bajo los estándares de manejo de repositorios Git.

# MODELO DE ARQUITECTURA GENERAL DE PRUEBAS AUTOMATIZADAS



**Capa de Negocio (Feature)**: Esta capa incluye todo lo que está relacionado con las reglas de negocio, la descripción de casos y/o escenarios de prueba.

Los componentes de esta capa se utilizan para editar y navegar por las estructuras de la suite de pruebas, relacionar casos de prueba con objetivos de pruebas (criterios de aceptación) o requisitos del SUT (Sistema Bajo Prueba), documentar en diseño de la prueba.

En esta capa se tiene en cuenta aspectos como:

* Escenarios de prueba en formato Cucumber (como, cuando, entonces).
* Desarrolla, capturar o derivar los datos para la prueba.
* Documentar el diseño de la prueba, es decir, cada caso de prueba o escenario.

**Capa de Definición**: Esta capa comprende el desarrollo de las definiciones para los escenarios o casos de prueba descritos en la capa de negocio, los cuales se pueden describir en definiciones y pasos.

Algunos aspectos para tener en cuenta:

* Especificar métodos de prueba para un caso de prueba o conjunto de casos de prueba.
* Definición de scripts de prueba para la ejecución de los casos de prueba.
* Especificar secuencias de prueba o comportamientos de prueba completos.
* Es conveniente documentar los datos de prueba, casos de prueba y/o los métodos de la prueba.

**Capa Page/Driver:** Esta capa define la iteración entre el SUT (Sistema Bajo Prueba) con las acciones a ejecutar en cada uno de los elementos objeto que son parte de las pruebas. En resumen, es el que maneja los recursos propios de la herramienta de automatización que posibilitan esta interacción.

**Capa de Datos (Data Driven):** Esta capa define la interacción con diferentes fuentes de datos que se tengan, con el fin de que los datos sean canalizados a las capas que corresponden y que a su vez se mantengan un mismo patrón resultado. Independiente de la fuente esta deberá ser consumido de una misma manera en la automatización. En este caso se hace consumo de los datos de una fuente externa al código fuente, pero presente en el proyecto.

**Ejecución:** Esta capa consta de herramientas de soporte que están en capacidad de ejecutar casos de prueba automáticamente (en el caso de que se hayan construido, asegurarse de la disponibilidad de ambientes, notificar que la ejecución fue exitosa o tuvo algún error).

En esta capa lo importante para tener en cuenta es:

* Se ejecutan los casos de prueba automáticamente vía consola de comandos.
* Se generan los reportes de los resultados de la ejecución de los casos de prueba

**Resultados:** Los resultados corresponden al informe de pruebas y las evidencias que soportan las mismas.

Lo resultados se pueden ver de diferentes formas:

**• Informes tipo HTML:** Este tipo de informe de Serenity se publica cómo un link en la ruta interna del proyecto correspondiente y presenta los Screenshots, steps, tiempos, resultados específicos e informe de resultados resumen.

**• Logs de resultados:** Generados de manera personalizada pro el robot de pruebas. Están en formato de texto, TXT o PROPERTIES.

**Capa de Utilitarios:** Los utilitarios en esta capa abordan diversas funciones, como la manipulación de datos, la captura y gestión de registros (logs), acciones de interacción con la interfaz de una aplicación web, manejo de archivos externos (Excel), control de data, generar datos aleatorios (fechas, números y similares), entre otros. La idea central es encapsular lógica comúnmente utilizada y ofrecer interfaces sencillas y coherentes para su implementación en los scripts de automatización.  
Son clases declaradas por funcionalidades que a su vez contienen métodos complementarios para su objetivo. Estos utilitarios están abiertos dentro del proyecto, lo cual quiere decir que pueden ser manipulados/modificados de acuerdo a la necesidad de cada automatización, al igual se pueden crear más métodos utilitarios en caso de ser necesario siguiendo las siguientes características:

* **JavaDoc:** Es una herramienta de generación de documentación en Java que extrae automáticamente comentarios del código fuente para crear documentación legible, para documentar de esta manera coloca comentarios /\*\* ... \*/ justo antes de clases, métodos o campos, describiendo su propósito y uso. Incluye etiquetas como @param y @return para describir parámetros y valores de retorno.
* **Análisis de Código:** Si durante el análisis de código estático se presenta alguna novedad con los métodos creados posteriormente, se deben crear métodos auxiliares más simples que complementen la ejecución del método principal, esto permitirá mayor legibilidad y mantenibilidad al código.

# DETALLE DE CAPAS PROCEDIMENTALES

**Nota:** *Este documento detalla un proyecto que utiliza Maven, Serenity y Selenium. Se asume que la persona implementadora posee al menos conocimientos básicos en programación y Java para comprender y ejecutar las instrucciones proporcionadas.*

## Capa de Negocio

Los archivos de características son aquellos donde los analistas de negocio (BA) almacenan los requisitos y pueden crear el puente entre los requisitos y las pruebas automatizadas. Deben estar escritos en lenguaje Gherkin, el cual es un lenguaje simple que se utiliza para explicar el comportamiento esperado de un sistema. Admite escritura en cualquier idioma y se define en ellos los datos de prueba.

Es importante que exista un Feature por cada historia de usuario generada, esto facilita la comprensión del robot y la mantenibilidad en el futuro.

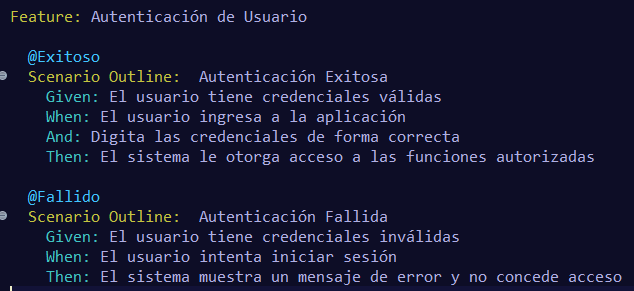
Los archivos feature se deben ubicar en la siguiente ruta dentro del proyecto de automatización: “**src/test/resources/features/file.feature”.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

El feature debe tener una estructura clara y comprensible. Debe incluir los siguientes elementos:

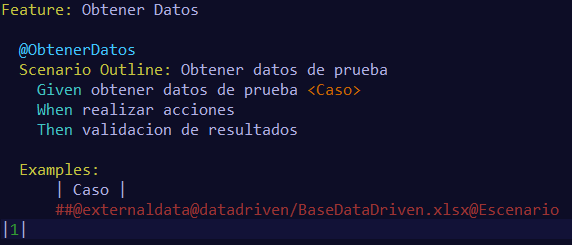
* **Nombre del Feature (Feature):** Debe ser descriptivo y conciso, reflejando la funcionalidad que se está implementando.
* **Tags:** son una forma de organizar y ejecutar escenarios, casos de prueba o Historias de usuario específicos según categorías o características, se declara con “**@**” más el resumen de lo que se ejecutará, se ubica antes de los escenarios que quiera ejecutar bajo este tag.
* **Descripción (Scenario Outline):** Proporciona una explicación detallada del comportamiento esperado del Feature.
* **Precondiciones de Ejecución (Given):** Describe el estado inicial o precondiciones necesarias para el escenario de prueba. Es usado para presentarnos los actores (y su entorno) de un escenario; Siempre va en el comienzo de un escenario.
* **Pasos de ejecución (When):** Describe la(s) acción(es) o evento(s) que desencadena el escenario de prueba; Aunque se pueden concatenar varios When no es recomendable, para ello existe la palabra clave And.
* **Criterios de Aceptación (Then):** Especifica los criterios que deben cumplirse para considerar el Feature como completado satisfactoriamente, se realizan las aserciones correspondientes para confirmar el resultado esperado; Siempre va en el final de un escenario y para concatenarlos se usa And.
* **Escenarios de Prueba:** Incluye ejemplos de casos de prueba que pueden utilizarse para verificar la implementación del Feature, cada uno de ellos con la estructura correspondiente mencionada anteriormente.



* **Examples:** Indica y actualiza la cantidad de ejecuciones a realizar de cada Feature, se debe relacionar la ruta donde encontrar el “DataDriven” que contiene la información de cada ejecución. El framework contiene un método que actualiza el feature basado en el archivo relacionado en este punto, por lo cual se debe actualizar la ruta.

La relación de variables debe tener el mismo nombre, tanto en la declaración del Feature como en el archivo DataDriven.

En el siguiente ejemplo “caso” es la variable que llamaremos desde el archivo externo, este corresponde a un “Int” (Número Entero) que determina la cantidad de veces a ejecutar el escenario.



## Capa de Definición / Definition

Las clases en la capa definition son la asociación desde los feature hacia el lenguaje y la lógica de programación, actúa como una definición de comportamiento para un conjunto de pasos relacionados con pruebas de datos (data-driven testing). En esta se asocian los términos de ejecución de los escenarios (Given, When, Then) con los métodos ejecutables en la capa “**Page/Driver”.** En ellas se encuentran los métodos asociados a cada etiqueta de los escenarios de prueba (Given, When, Then), que a su vez hacen la invocación de los métodos de “Steps” o pasos.

Los archivos relacionados a definición se deben ubicar en la siguiente ruta dentro del proyecto de automatización: “**src/test/java/definition/class.java”.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Los definition deben seguir con la nomenclatura de escenarios del feature, igualmente debe tener una estructura clara y comprensible. Debe incluir los siguientes elementos:

* **Declaración de Clase:** La declaración de las clases se deben realizar de forma pública, su nomenclatura debe llevar aquella parte del escenario a trabajar (DataDriven, Login, Transacciones, etc…) concatenado a la palabra “**Definition”,** esto permitirá reconocer fácilmente la asociación e invocación entre clases, la mantenibilidad futura y mayor comprensión sobre qué capa estamos trabajando. **Ej: “public class DataDrivenDefinition{}”.**
* **Anotaciones:** Se utilizan para proporcionar información específica para la generación de informes y la ejecución de pruebas. Por ejemplo, la anotación **@Steps** se utiliza para inyectar instancias de clases de pasos (steps) en las clases de prueba, la asociación que va después de este debe corresponder a la importación de la clase en la capa Steps correspondiente, las anotaciones **@Given, @When, @Then,** son utilizadas para describir y asociar los términos de ejecución declarados en el feature.

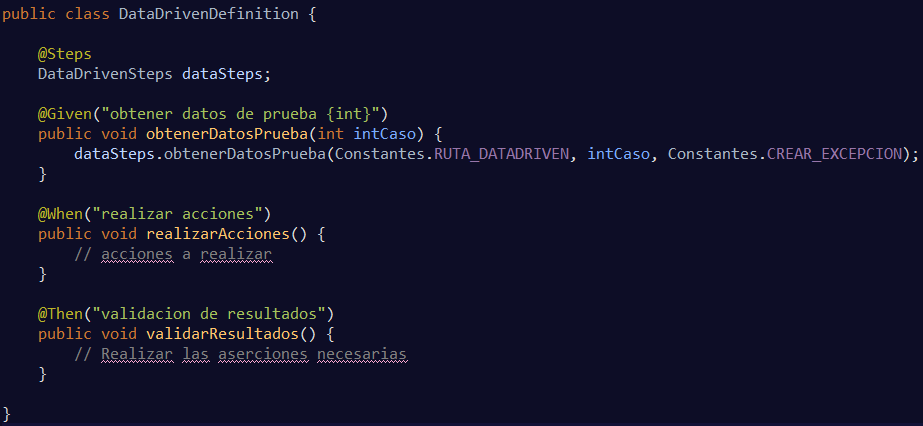
La declaración debe llevar la notación de ejecución (Given, When, Then), sumando el mismo escenario declarado en el feature entre paréntesis y comillas dobles.

**Ej:** @Given("obtener datos de prueba {int}"), en este caso el {int} hace referencia al tipo de dato relacionado en el feature como <Caso>, lo recibe como parámetro y lo pasa al método que va enseguida.

* **Declaración de Métodos:** Los métodos declarados en la clase “Definition” son funciones que encapsulan pasos específicos de la prueba. Son funciones que se ejecutarán invocando a su vez los métodos que se encuentran dentro, se deben declarar de forma pública y vacio ya que no retornará ningún tipo de información en su ejecución.

**Ej:** public void obtenerDatosPrueba(int intCaso) {}, se recibe como parámetro el dato trabajado en la declaración de la anotación anterior (si aplica).

* **Invocación de Métodos:** Se refiere al acto de llamar o ejecutar un método específico. En este contexto, los métodos que se relacionen dentro del método de definición se invocarán durante la ejecución de un escenario de prueba automatizado.



## Capa Page/Driver (Steps)

Las clases en la capa de Page o Steps son el detalle de los pasos que se deben realizar durante la ejecución de las pruebas. Sirve como una interfaz entre los escenarios a probar y la interacción con las interfaces a probar. Su función principal es encapsular la lógica de interacción con los elementos de la interfaz y proporcionar métodos utilitarios que reúnan acciones específicas para realizar acciones comunes o pasos completos unificando las capas de utilitarios con la interfaz a utilizar.

Los archivos relacionados a definición se deben ubicar en la siguiente ruta dentro del proyecto de automatización: “**src/test/java/steps/class.java”.**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Los steps igualmente debe tener una estructura clara y comprensible. Debe incluir los siguientes elementos:

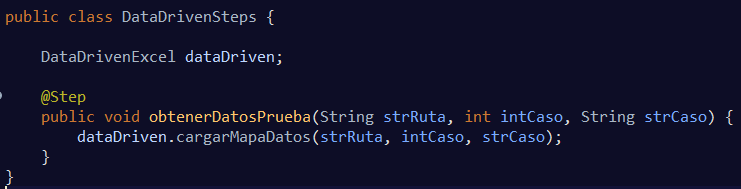
* **Declaración de Clase:** La declaración de las clases se deben realizar de forma pública, su nomenclatura debe llevar la acción completa que se ejecutará,esto permitirá reconocer fácilmente la asociación e invocación entre clases, la mantenibilidad futura y mayor comprensión sobre qué capa estamos trabajando. **Ej: “public class DataDrivenSteps {}**
* **Anotaciones:** se utiliza para marcar métodos en clases de pasos (steps) como pasos de prueba. Estos métodos representan acciones o pasos específicos que se ejecutan como parte de un escenario de prueba. La anotación **@Step** proporciona beneficios clave en términos de legibilidad del código, reutilización y generación de informes.
* **Declaración de Métodos:** Los métodos marcados con @Step pueden ser reutilizados en diferentes escenarios de prueba, mejorando el modularidad y la eficiencia del código de automatización.

**Ej: “public void obtenerDatosPrueba(){}”**, se recibe como parámetro el dato trabajado en la declaración de los métodos utilitarios (si aplica).

* **Invocación de Métodos:** Se refiere al acto de llamar o ejecutar un método específico. En este contexto, los métodos que se relacionen dentro del método de steps se invocarán durante la ejecución de un escenario de prueba automatizado.

**Ej:** En Definición tenemos un paso “Login a la aplicación web”, en la capa Steps dentro de este método se asociarán las tareas específicas para lograr este paso: Abrir driver(navegador), cargar URL de la aplicación, dar click en Login, digitar usuario correcto, digitar contraseña correcta, click en iniciar sesión, validar que inició sesión.

Esto es un ejemplo básico sobre como funcionan los métodos configurados dentro de las clases en la capa Page/Steps.



En el ejemplo anterior se muestra el método que obtiene datos de prueba, este invoca un utilitario que se encuentra dentro del proyecto el cual crea un mapa de datos con la información encontrada en el “Datadriven”.

Estos métodos pueden invocar uno o más métodos utilitarios según la acción/paso completo que se quiera realizar.

## Tipo de Automatización

Como parte de la cobertura y planteamiento base de la estrategia de calidad y la aplicación de los estándares, se crea una plantilla base para una automatización de aplicaciones Web de tipo con cobertura para Frontend y Backend, automatizando pruebas empleando la UI del aplicativo.

### Frontend

El tipo de Automatización FrontEnd es a nivel de interfaz gráfica la que simula el comportamiento de un usuario haciendo uso del sistema, esta se correlaciona al nivel de Automatización de UI.

En el framework base se encuentra una sección específica con secciones y configuraciones específicas que funcionan para la automatización de FrontEnd, las cuales son:

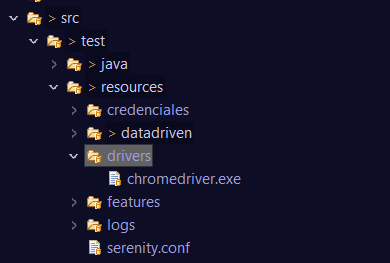
* **Localizadores:** Son identificadores únicos que permiten a los scripts de prueba localizar y interactuar con elementos específicos de una interfaz de usuario, como botones, campos de texto o enlaces, facilitando así la automatización de las acciones del usuario.

En el proyecto base de automatización se encuentra el directorio correspondiente que se encuentra en la ruta **“src/test/java/ui”** en esta ruta se deben realizar los mapeos de los localizadores para cada elemento con el que se vaya a interactuar. Los localizadores deben ser ubicados/guardados en archivos por pantalla, deben declararse como constantes y tipo de elemento que se vaya a trabajar según la necesidad de la automatización (By, WebElementFacade, etc…), adicional crear el método get correspondiente que nos retorne el elemento para usarlo en las clases de las capas correspondientes.



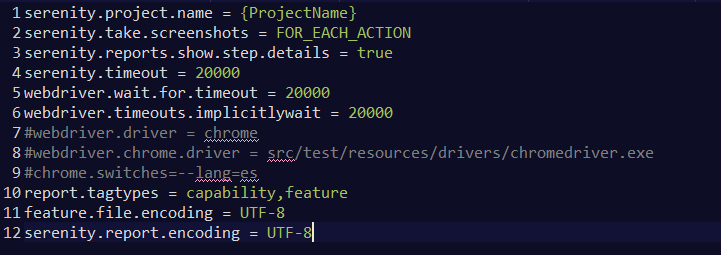
* **Driver:** El driver facilita la interacción con la aplicación bajo prueba y controla la ejecución de los scripts, permitiendo la automatización de acciones y la recopilación de resultados. Este se refiere al elemento funcional básico del navegador que se quiera utilizar y de este depende algunas configuraciones básicas que se deben realizar en el proyecto antes de iniciar a ejecutar las pruebas.

Lo primero que debemos realizar es descargar el driver del navegador correspondiente (Recomendación Chrome), el ejecutable se debe ubicar en la ruta: **“src/test/resources/drivers”.**



Al tener descargado el driver, debemos tener en cuenta la versión que se va a utilizar y que sea la misma versión del navegador instalado en el equipo, luego de adicional el driver correspondiente en el proyecto, debemos configurar qué driver estamos usando en el archivo serenity.properties de nuestro proyecto, este se encuentra en la carpeta base (src) de nuestro proyecto.

En el proyecto base se encuentra en comentario la configuración base para el driver de Chrome.



### Backend

El tipo de Automatización BackEnd es a nivel de lógica y procesos, se enfoca en la validación de servicios, bases de datos y lógica del servidor para asegurar su correcto funcionamiento, eficiencia y seguridad.

En el framework base se encuentra una sección específica con secciones y configuraciones específicas que funcionan para la automatización de BackEnd, las cuales son:

* **Servicios:** Se ubica una clase en el directorio de “utilities” con diferentes métodos útiles para probar los servicios, esto permitirá crear, modificar o eliminar elementos del Header y Body de la petición a los servicios.

Esta clase se encuentra en la ruta **“src/test/java/utilities/Servicios.java”**.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## Herramientas Recomendadas

|  |  |  |
| --- | --- | --- |
| **Area** | **Herramienta** | **Administración / Configuración** |
| Repositorios | Azure | Cliente Ecopetrol |
| Lenguaje de Programación | Java | Implementador de Solución |
| Constructores | Maven | Implementador de Solución |
| Entorno integrado de Desarrollo (IDE) | Eclipse - IntelliJ | Implementador de Solución |
| Lenguajes | Gherkin | Implementador de Solución |
| Framework Automatización | Cucumber | Implementador de Solución |
| Framework Automatización | Junit | Implementador de Solución |
| Framework Automatización | Selenium | Implementador de Solución |
| Framework Automatización | RestAssured | Implementador de Solución |
| Reporteria | Serenity | Implementador de Solución |

## Mantenibilidad

El código dispuesto es una base pensada para ser incremental y complementada de acuerdo con la necesidad en cada proyecto en el que sea empleado. Se encuentran métodos disponibles que facilitan el uso y configuración de cada proyecto, aparte de permitir iniciar de una forma rápida con las herramientas preconfiguradas.

Se debe tener en cuenta la configuración base del proyecto, gestión de ambientes, invocación de métodos como objetos dentro del proyecto, mapeo de localizadores de la UI, además crear los escenarios con los tags de Gherkin y su asociación a la capa de definition.

* Mantén la **estructura del proyecto** organizada de manera clara y coherente. Utiliza la estructura de directorios recomendada por Maven para separar recursos, configuraciones y código fuente.
* Incorpora **comentarios descriptivos** en el código, especialmente en partes críticas o complejas. Emplea Javadoc para documentar clases y métodos, facilitando la comprensión para futuros desarrolladores.
* Adopta una **nomenclatura clara** y consistente para las variables, métodos y clases. Sigue las convenciones de nomenclatura de Java para mejorar la legibilidad del código.
* Descompón el código en **módulos y clases reutilizables**. Utiliza la programación orientada a objetos para crear componentes independientes que puedan ser modificados o mejorados sin afectar otras partes del código.
* **Administra las dependencias** del proyecto mediante Maven. Mantén actualizadas las versiones de las bibliotecas y resuelve cualquier conflicto de dependencias. Mantén actualizadas y funcionales las suites de prueba utilizando Serenity, Selenium y RestAssured. Incorpora pruebas de regresión para asegurar que las nuevas implementaciones no afecten las funcionalidades existentes.
* Asegúrate de tener **documentación actualizada** sobre la arquitectura del proyecto, configuraciones y flujos de trabajo. Esto facilitará la comprensión del proyecto para los nuevos miembros del equipo.
* Utiliza sistemas de **control de versiones** como Git para rastrear cambios en el código. Realiza commits significativos y etiqueta las versiones para un seguimiento claro de la evolución del proyecto.

# OBTENCIÓN DE DATA

Los datos serán obtenidos de un archivo DataDriven en formato .xlsx (Excel), donde cada hoja de este archivo corresponderá a un caso automatizado construido, estas hojas deben llevar el nombre correspondiente al escenario/tag asociado al feature correspondiente. Esta información será consultada y cargada en un mapa de datos (HashMap), como primer paso definido en los features correspondientes y que al hacer la trazabilidad hasta la capa PageObject, se puede encontrar el método cargarDatos() donde se realiza la interacción con el DataDriven

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

# MANEJO DE CONTROL DE VERSIONES

Sistema de control de revisiones distribuido, Cada directorio de trabajo git es un repositorio completamente funcional y con total capacidad independientemente.

Su principal propósito es que se pueda regresar a un estado anterior del proyecto o conocer, incluso, toda su evolución en el tiempo, desde su inicio hasta donde la actualidad.

Gráfico, Diagrama, Gráfico de burbujas

Descripción generada automáticamente

El uso efectivo de ramas facilita el desarrollo paralelo y la implementación de nuevas características. Recomendamos adoptar una estrategia de ramificación, como Gitflow, para organizar el flujo de trabajo. Esto incluye ramas principales (main, develop) y ramas de características o correcciones (feature, hotfix).

# BUENAS PRACTICAS DE AUTOMATIZACION DE PRUEBAS

Los principios SOLID fueron introducidos por Robert C Martin, estas guías buscan crear código limpio que sea mantenible, reutilizable, legible entre otros aspectos en una solución de software e indican que:

* S: Principio de responsabilidad única (Single responsibility principle)
* O: Principio de abierto/cerrado (Open/closed principle)
* L: Principio de sustitución de Liskov (Liskov substitution principle)
* I: Principio de segregación de la interfaz (Interface segregation principle)
* D: Principio de inversión de la dependencia (Dependency inversion principle)

Las **CLASES** deben ser sustantivos y en singular con la primera letra en mayúscula y sus siguientes palabras todas con la primera letra en mayúscula también:

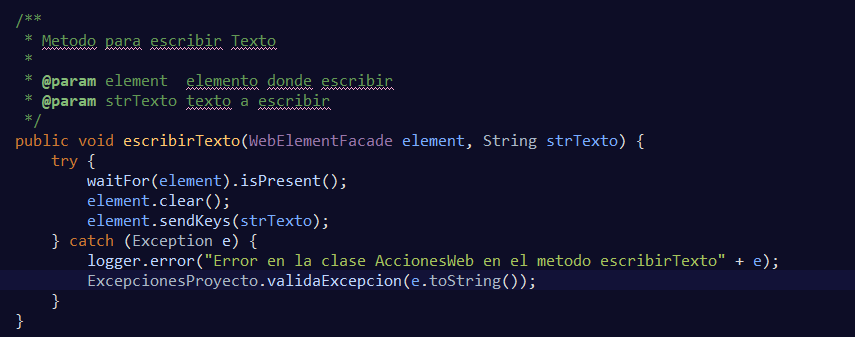
* **Uso Correcto**: MiClase, Vehiculo, FormularioRetiro, ConexionBaseDatos
* **Uso Incorrecto:** miClase, vehiculo, formularioregistro, Conexionbasedatos

Texto

Descripción generada automáticamente

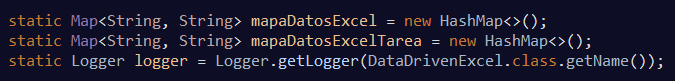
Los **MÉTODOS** deben ser verbos y en singular con la primera letra en minúscula y las siguientes palabras en mayúscula:

* **Uso Correcto**: calcularTotal(), validarUsuario(), enviarCorreo(), cargarDatos()
* **Uso Incorrecto:** CalcularTotal(), ValidarUsuario(), EnviarCorreo(), CargarDatos()



Las **VARIABLES** también deben ser escritas en notación lowerCamelCase, esto quiere decir, que la primera letra siempre debe iniciar en minúscula, mientras que las demás palabras siguientes la primera letra en mayúscula:

* **Uso Correcto**: monto, esAfiliado, personaNaturalPensionado
* **Uso Incorrecto**: Monto, Esafiliado, PersonanaturalPensionado



Las **CONSTANTES** se deben escribir con letras mayúsculas, separando las palabras con raya al piso:

* Uso Correcto: NUMERO\_HIJOS, NAVEGADOR, PERSONAS\_POR\_DEPARTAMENTO
* Texto

  Descripción generada automáticamenteUso Incorrecto: numero\_hijos, Navegador, Personas\_por\_Departamento

Emplee una **ESTRUCTURA MODULAR** y reutilizable, crear módulos más pequeños como parte de los utilitarios ayudará a la mantenibilidad y escalabilidad del código, permitirá trabajar sobre lo que ya se ha creado y dará legibilidad al código.

# BUENAS PRACTICAS PARA CREACIÓN DE CASOS AUTOMATIZADOS

* **Descripción de Casos de Uso:** Antes de comenzar con la automatización, comprenda claramente los casos de uso que se automatizarán. Documente los escenarios de prueba y sus resultados esperados.
* **Estabilidad y Repetibilidad:** Priorice casos de prueba estables y repetibles para la automatización. Evite escenarios demasiado volátiles que puedan cambiar frecuentemente.
* **Esperas Explícitas:** Utilice esperas explícitas para sincronizar la ejecución de la prueba con la carga dinámica de elementos en la interfaz de usuario. Esto significa que las esperas en selenium deben corresponder al cumplimiento de acciones y NO el cumplimiento de tiempos exactos.
* **Revisiones de Código:** Realice revisiones periódicas del código de automatización para identificar áreas de mejora y asegurar la coherencia con los estándares del equipo.
* **Selectores:**  
  La prioridad principal es garantizar la unicidad y estabilidad de los localizadores para evitar flaquezas en las pruebas. Usa selectores específicos y evita dependencias en elementos dinámicos.

Construye localizadores basados en la estructura jerárquica del DOM y abstrae su definición para facilitar la gestión y mantenimiento. Se prefiere CSS sobre XPath, pero esto depende de la necesidad a cubrir en el momento y la arquitectura de la aplicación.

Prioriza la legibilidad de los localizadores y realiza revisiones regulares para adaptarte a cambios en la aplicación.

**FIN DEL DOCUMENTO**

**RELACIÓN DE VERSIONES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Documento Anterior** | | | |
| **Versión** | **Fecha**  **dd/mm/aaaa** | **Código y Título del documento** | **Cambios** |
| **No aplica** |  |  |  |
| **Documento Nuevo** | | | |
| **Versión** | **Fecha**  **dd/mm/aaaa** | **Cambios** | |
| 1 | 07/11/2023 | Elaboración del instructivo de implementación para pruebas automatizadas, siguiendo los principios y buenas prácticas usando el framework y librerías estandarizadas por la empresa. | |

|  |
| --- |
| **Para más información dirigirse a:** |
| **Elaboró:** Jhon Sánchez – Consultoría Calidad – NTT Data.  **Revisó:** Angélica Barrero, Líder DevOps - CoordinaciónTransición del Servicio  **Dependencia:** Departamento de Calidad Digital – Vicepresidencia de Tecnología e Innovación – VTI |

|  |  |
| --- | --- |
| **Revisado electrónicamente por:** | **Aprobado electrónicamente por:** |
| **ANGÉLICA BARRERO**  **Profesional Integral de Operación Digital B**  **Coordinación de Transición de Servicio - VTI** | **PAULA CARDONA**  **Coordinador Transición de Servicio**  **Coordinación de Transición de Servicio -VTI** |
| Documento firmado electrónicamente, de acuerdo con lo establecido en el **Decreto 2364 de 2012**, por medio del cual se reglamenta el artículo 7 de la Ley 527 de 1999, sobre la firma electrónica y se dictan otras disposiciones.  Para verificar el cumplimiento de este mecanismo, el sistema genera un **reporte electrónico** **que evidencia la trazabilidad de las acciones** de revisión y aprobación por los responsables. Si requiere verificar esta información, solicite dicho reporte a Service Desk. | |