# Selenium con Java y Cucumber

# SELENIUM IDE

Es esencialmente un record and playback (recorder), se hacen grabaciones de un sitio web y vos podés reproducir de nuevo los pasos que recorriste y tambien permite hacer validaciones.

# SELENIUM WEB DRIVER

Es una librería que permite interactuar con los elementos de una pagina web, te permite modificar programáticamente la lógica de lo que estas haciendo a diferencia de SELENIUM IDE.

# CUCUMBER

Es una librería que nos permite escribir en un lenguaje amigable para personas no técnicas como un manager o bussiness analyst la funcionalidad o el comportamiento de nuestros sistemas, distintos escenarios o feautures que debemos crear. Usa el lenguaje Gherkin. Se puede implementar en Python, JAVA, CSHARP, Javascript.

Nos permite tener documentación conocida como “documentación viva” y al mismo tiempo nos permite catalogar las pruebas, los features y colocar TAGS, nos permite parametrizar nuestras pruebas para poder ejecutarla múltiples veces con diferentes datos.

Cucumber es una manera que tenemos de documentar el comportamiento de nuestro sistema “features” que estamos probando, al mismo tiempo que estamos probando los casos de prueba. Esto sirve mucho para que el cliente nos defina los casos de prueba de manera viva y generarnos un criterio de aceptación. Con Cucumber tenemos documentación viva, casos de prueba y reportes, todo en un mismo lugar.

EJ:

Texto

Descripción generada automáticamente

**Contiene los siguientes elementos:**

**Feature:** Define la funcionalidad que estamos probando

**Rules:** Definición de reglas de negocios que probamos dentro del archivo. EJ: puedo hacer el filtro de los elementos que quiero comprar antes de comprarlo.

Cucumber NO ES una herramienta de automatización.

# Gradle

Es una herramienta de “buildeo”, se encarga de agarrar las dependencia que nosotros definimos en nuestros archivos gradle, si necesito Selenium y Cucumber nos bajamos las dependencias y las tengo dentro de mi biuld para que la acplicacion java pueda funcionar. Tambien nos va a permitir configurar tareas o lo que se conoce como “tareas de gradle” para ejecutar nuestras pruebas usando un tag o distintos argumentos, esto es útil para ser corrido en una herramienta de “continuous integration” ya sea GIT Actions, Jenkins, Azure Devops Pipelines.

Dentro de nuestro Framework se va a ver lo siguiente:



Y dentro de Build.gradle se encontrara lo siguiente:

Texto

Descripción generada automáticamente

Aquí se definen las dependencias que usamos y algunas tareas.

**COMANDO - GRADLE BUILD:** Se baja todas las dependencias que todavía no se hayan bajado de las que tenemos incorporadas en el código de arriba.

**IMPLEMENTATION:** Librerias que actúan sobre el Main

**TEST IMPLEMENTATION:** Librerias que actúan sobre el Test

**CARPETA MAIN**: Es donde va a estar la lógica y el código que va a compilar para producción

**CARPETA TEST:** Es donde van a estar los casos de prueba para la lógica que tenemos.

**ORGANIZACIÓN:**

Interfaz de usuario gráfica

Descripción generada automáticamente

**ESTRUCTURA ORGANIZATIVA DEL PROYECTO:**

Diagrama

Descripción generada automáticamente con confianza media

**Patrón de diseño POM – Page Object Model:** Es un patrón de diseño, es una manera de estructurar el código para que sea mas sencillo el mantenimiento. Definir las acciones y elementos de una pagina web en un solo lugar y volver a usarlas donde necesitemos.

**Page Object Class**: Modelar la página, significaría representar dentro de las funciones y variables que tenemos dentro de la page object class las interacciones que un usuario puede tener en esa página web con los web elements, ya sea interactuar con un click en un botón o el llenado de un campo, etc.

Texto, Pizarra

Descripción generada automáticamente

**Ejemplo de uso del POM:**

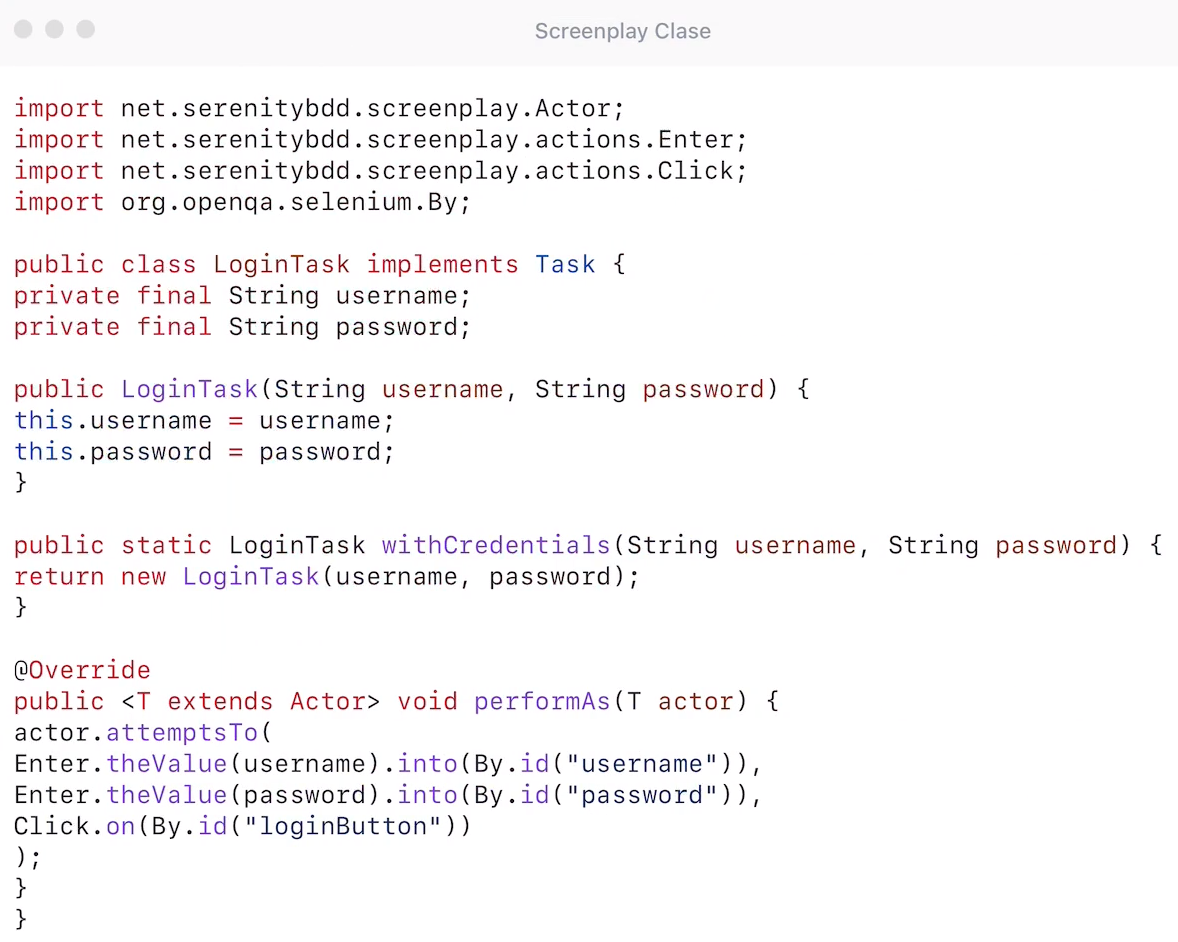
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Patrón de diseño Screenplay**

**Screenplay:** Modela la aplicación sergun las tareas y acciones que hacemos contra la app web y seugn los actores (usuarios) que van a intentar realizar tareas, acciones y preguntas que son las que nos van a dar los casos de prueba para los assertions. Trata de dividir según la responsabilidad única de las secciones por tarea, por secciones. Plantea un actor como usuario que intenta realizar una tarea como podría ser “loggearse” y se plantea la pregunta de saber si esta o no logeado en base a nuestros casos de prueba.

Ejemplo de uso de Screenplay:



# Texto Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamentePAGE OBJECT MODEL VS. SCREENPLAY:

# Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamentePAGE OBJECT MODEL VS. SCREENPLAY:

**FEATURE FILE:** Nos da la posibilidad de escribir en un lenguaje normal NO TECNICO una serie de pasos que vamos a hacer para evaluar un comportamiento, Cucumber da forma a lo que es BDD Behavior Dreaven Development que seria desarrollar de acuerdo al comportamiento que tenemos delante de nosotros.

**GLUE**: Es la ubicación donde se encuentran los Steps Definition

**PLUGIN**: Lugar donde se crean los reportes con un nombre especifico y un HTML

**SELENIUM WEBDRIVER:** Es una librería, una interfaz que nos permite interactuar con una página web y sus elementos de manera programática. Nos permite hacer click, llenar campos de texto, etc.

**LOCATORS**: Selenium utiliza unos “locators” o localizadores que le permiten ubicar al elemento y ejecutar una acción sobre él. De esta misma forma puede también acceder a atributos visibles como el texto (mediante el método getText()) y a atributos no visibles desde la interfaz de usuario a través de comandos como getAttribute(“nombre del atributo”).

**BASE PAGE**: Capa primaria donde definimos cosas que reutilizamos en nuestras PAGE OBJECT CLASSES. No pertenece a ninguna pagina en particular.

**ASSERTIONS**: TestNG proporciona una serie de métodos de aserción a través de la clase Assert para verificar que las condiciones de las pruebas sean cumplidas. Estas aserciones son esenciales para validar el comportamiento esperado de la aplicación bajo prueba. A continuación, te detallo algunas de las aserciones más comunes y te proporciono ejemplos utilizando Selenium y Java:

1. **assertEquals**

Verifica que dos valores sean iguales.

  Assert.assertEquals(actualTitle, expectedTitle, "El título de la página no es el esperado.");

2. **assertNotEquals**

Verifica que dos valores no sean iguales.

   Assert.assertNotEquals(actualTitle, incorrectTitle, "El título de la página no debería ser este.");

3. **assertTrue**

Verifica que una condición sea verdadera.

  Assert.assertTrue(isElementPresent, "El elemento debería estar presente.");

4. **assertFalse**

Verifica que una condición sea falsa.

   Assert.assertFalse(isElementPresent, "El elemento no debería estar presente.");

**Tags en Cucumber: Se usan para correr pruebas particulares demarcando esta limitación en los Features de Cucumber.**

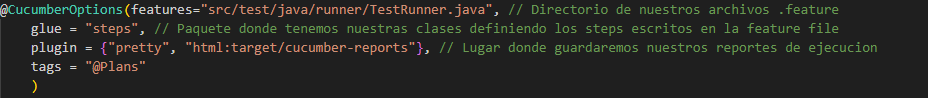
**EJ:**

**Tag llamado “Plans” declarado en un Feature**

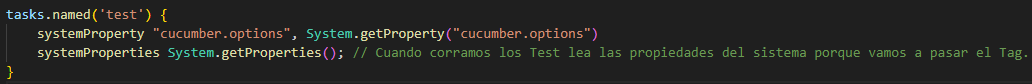
**Texto

Descripción generada automáticamente**

**Tags de “Plans” definido en TestRunner.java**

****

**Tags definido en build.gradle.**



**Opciones a la hora de correr tags**

*gradle test -Dcucumber.filter.tags="@Plans"* **-> Va a correr Scenarios taggeados con @Plan**

*gradle test -Dcucumber.filter.tags="Not @Plans"* **-> Va a correr todos los Scenarios que no tengan el tag @Plan**

*gradle test -Dcucumber.filter.tags="@Plans" and "@Courses"***-> Va a correr todos los scenarios que tengan ambos, @Plan y @Courses tags al mismo tiempo.**

*gradle test -Dcucumber.filter.tags="@Plans" or "@Courses"* **-> Va a correr los scenarios que tengan los tags @Plans o @Courses (osea...todos los scenarios que tengan uno u otro).**

*gradle test -Dcucumber.filter.tags="@Plans" and not "@Courses"* **-> Va a correr todos los scenarios que tengan el tag "@Plans" y no tengan "@Courses".**

Texto

Descripción generada automáticamente

**BACKGROUND - CUCUMBER**: Son prerrequisitos – Presteps, son un conjunto de steps que son comunes a todos os escenarios que tenemos dentro de nuestro Feature Files. Es lo que se ejecuta antes de que se ejecute cada escenario. NO deberían tener mas de 2 steps.

**RULE - CUCUMBER**: La “Rule” nos dice que regla del Bussines estamos probando con los escenarios que tenemos en el Feature file. Es la pieza fundamental del Feature file.

**¿Qué ES UN IDE?**

Es Integrated Development Enviroment – Ambiente integrado de desarrollo. Es donde vamos a escribir nuestro código en base a las herramientas que carguemos para que funcione a la par de nuestro accionar programatico.

**JAVA**

**JRE - Java Runtime Enviroment:** Es algo que tenemos en el disco de nuestra computadora y es lo que crea el Java Virtual Machine.

**JVM – Java Virtual Machinne:** Se encarga de ejecutar las aplicaciones en Java.

**JDK – Java Development Kit:** Lo usamos para desarrollar las aplicaciones Java y permite que JVM ejecute las mismas.

**SELENIUM ACTIONS**: Proporciona un control granular sobre exactamente lo que pueden hacer los dispositivos de entrada designados. Nos provee el poder ejecutar click derecho, doble click, entre otras funciones para nuestras pruebas.

**Patrón de fondo

Descripción generada automáticamenteEJEMPLO DE ACCIONES**:

EJEMPLO DE INCORPORACION EN EL CODIGO:



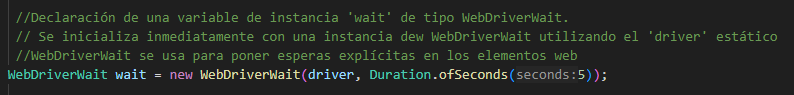
Texto

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**ESPERAS EXPLICITAS E IMPLICITAS**

**EJEMPLO DE ESPERA EXPLICITA E IMPLICITA (POR SER DEFINIDA EN LA BASEPAGE EN ESTA FUNCION EN PARTICULAR):**



**ESPERA IMPLICITA:** Aquella que seteamos de manera Global para todo el Framework.

**DEBUGGING**: Explorar el comportamiento de la lógica para ver el detalle.

**MANEJAR POP-UP O VENTANA EMERGENTE AL EJECUTAR PRUEBAS:**

Texto

Descripción generada automáticamente

La idea es llegar al “iFrame” o las “Alert Boxes”. El Dismiss va a “volar” la alerta que aparezca en el DOM.

BREAKPOINT: Orden de pausar el proceso de Debugging para mirar lo que esta pasando.

# **OPCIONAL: Otra manera de manejar los WebElements en nuestro Page Object Model.**

Esta clase es opcional ya que representa OTRA MANERA de hacer lo que hicimos previamente en nuestras clases para página. En esta oportunidad vamos a aprender a utilizar **PageFactory**, una librería muy usada hoy en día en proyectos de automatización debido a la simplificación que supone para manejar nuestras clases de página.

Recuerdan que estamos usando el Page Object Model (POM, por si les preguntan en una entrevista)? Bueno, esta librería lo que hace es darnos un par de trucos bajo la manga para que sea más fácil visualizar los **webelements** y tengamos que escribir menos código que lo que normalmente haríamos.

De nuevo, para recordarles: El framework que hicimos hasta ahora NO NECESITA de PageFactory, de hecho funciona mejor, en mi opinión, y es más robusto. El tutorial apunta a que aprendan a usarlo porque se que es algo que le van a preguntar o exigir en muchos puestos de trabajo. Está en ustedes decir "No muchachos, tengo ésta manera de hacer las cosas que es mejor!" y PAF! Les muestran el framework que aprendieron acá. En caso que la idea no prenda, también habrán aprendido a usar Page Factory en esta clase.

#### **Ahora si, manos a la obra! Instalemos las dependencias necesarias.**

Vamos por partes, lo primero que vamos a hacer es crear una clase, bajo el paquete que tenemos de Páginas (o el nombre que le hayan puesto al directorio donde están creando las clases para las páginas y donde está la BasePage). Le pueden poner el nombre que quieran, en este ejemplo, para verlo claro, le puse PageFactoryPage.

Texto

Descripción generada automáticamente

Observemos un par de cosas acá. Primero y principal, estamos usando dos librerías que vienen con **Selenium**, las cuales son **FindBy** y **WebElement** por supuesto. Estamos usando acá WebElement porque les recuerdo que no estamos más creando el WebElement en la **BasePage** como lo hacemos en nuestro Framework principal. Es por eso que necesitamos crearlos en esta clase. Ven ya una de las diferencias? Van a estar necesitando poner esto en todas las clases de página.

La clase que creamos va a extender la clase base de la misma manera para la construcción del WebDriver. Luego, vemos el corazón de PageFactory: la anotación @FindBy!

Esta anotación lo que va a hacer es ahorrarnos la creación de WebElements de la manera que tradicionalmente se hace. Por defecto, busca id o name iguales a lo que le digamos, pero podemos especificar otras maneras de localizar los webelements, como XPath, CSS, etc.

Usando la anotación, diciendo cómo se llama el campo por el cual vamos a encontrar el elemento y diciendo abajo WebElement además del nombre que queramos darle, vamos a tener creado nuestro WebElement listo para usarse.

Ahora, otra cosa...como no tenemos la inicialización de los WebElements por ningún lado, nuestros tests van a fallar diciendo "che...no hay ningún WebElement por acá!".

Para eso, vamos a añadir en la BasePage que estamos heredando en esta nueva clase que creamos, lo siguiente:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

En el constructor de la clase base, vamos a añadir lo que está en el recuadro. Esto va a permitir que, al instanciar las clases de página que heredan a ésta, los elementos con la anotación de PageFactory se inicialicen correctamente para ser usados.

Eso es todo el misterio detrás de PageFactory. La dificultad que tiene la mayoría es en saber dónde inicializar los elementos. Pueden hacerlo en el constructor de la clase de la página, pueden hacerlo en las step definitions...aunque mi consejo es que lo hagan en la clase base que instancia el WebDriver para que siempre esté hecho y no tengan que acordarse de arrancarlo ustedes a mano en cada lado que lo quieran usar.

De nuevo les recuerdo: Esto es una alternativa al Framework que hicimos hasta ahora. Mi preferencia personal es usar lo que les enseñé hasta ahora. Les estoy enseñando esta opción ya que es muy usado y requerido en la industria.

Prueben ésto en el framework que (espero!) ya tienen creado, fíjense qué les resulta mejor y elijan lo que les sea más útil. Es un tema muy subjetivo y he aprendido con los años que no hay una única respuesta a todos los problemas!

# **PageFactory y un truco ingenioso.**

Bienvenidos a una nueva lectura en este curo! En esta oportunidad les traigo un truco muy sencillo pero que, sobre todo, quiero explicarles el fundamento detrás. Como recordarán, en la lectura anterior estuvimos viendo la librería de Page Factory, una manera muy usada de manejar las clases de página cuando hablamos de Page Object Model.

Para los que no recuerden, son las anotaciones @FindBy para localizar webelements. Bueno, acá viene la parte teórica para que entiendan lo que vamos a hacer:

Con PageFactory, nosotros necesitamos inicializar los webelements con la función initElements que nos provee. Al hacer eso, se abren los proxies que comunican con el webdriver (sea Chrome, Firefox o cualquiera) pero aún no se busca ningún elemento. Esto ocurre cuando cada uno de ellos es usado.

Ahora bien, PageFactory lo que hace es enviar básicamente un request REST a la API de Selenium preguntando por este elemento...cada vez que lo tengamos que usar. Se imaginan que si hablamos de decenas de elementos siendo usados cientos de veces, el tiempo de ejecución se ve incrementado de manera significativa! Cómo podemos solucionarlo? Es que acaso los creadores de Selenium no pensaron en esta debilidad catastrófica?!? Tranquilos amigos y amigas! Que si lo hicieron y ofrecen una solución por demás sencilla.

Aplicación

Descripción generada automáticamente con confianza media

Con ustedes, el **@CacheLookup**! Esta anotación, que la podemos poner arriba del @FindBy, arriba como en el caso de la screen, lo que hace es guardar en caché el elemento en cuestión para que, en posteriores usos, no tenga que hacerse esa llamada a la API de Selenium para buscar nuevamente algo que ya se había encontrado. Esto lo que hace es salvar preciosos segundos de ejecución que, si multiplicamos por casos de prueba, scenarios, ambientes y veces que se corre algo, termina generando una ganancia en tiempo para nada despreciable.

#### **Pero esta solución también puede presentar un problema...**

Y es que, si estamos tratando de un elemento dinámico, como lo puede ser un contador, algo que una vez cargada la página cambia...vamos a recibir un StaleElementException. Por eso es que solamente deben usar esta técnica cuando se trata de elementos que sabemos que, una vez la página cargó, permanecen iguales. Si es algo dinámico como lo mencionado anteriormente, dejen todo como ya lo aprendimos para que el elemento sea ubicado nuevamente con sus nuevas características.

#### **Conclusión**

Como podrán ver, es muy fácil de usar. Solo una anotación agregada y estamos listos para salvar segundos (o recibir excepciones inesperadas si no me hicieron caso!).

Espero que les resulte útil y lo apliquen en sus proyectos!

# **Expresiones regulares en Cucumber: Cómo potenciar nuestra creación con ellas!**

# **Expresiones Regulares: Qué son?**

Las expresiones regulares son esta cosa misteriosa que no muchos saben exactamente cómo usar. De hecho, muchísimos ingenieros de Test experimentados que yo conozco no tienen idea de cómo usarlas. Bien hecho, pueden darnos una flexibilidad impresionante en nuestros steps de Cucumber, permitiéndonos permutar y redirigir flujos de acción de acuerdo a cómo escribamos el step sin la necesidad de tener que hacer uno muy parecido por duplicado.

#### **Tengo que usarlas como si no hubiese un mañana?**

Por favor, NO! De hecho, parte de las buenas prácticas que les enseño en esta clase son sobre no usar demasiado las expresiones regulares. Como regla general, uso lo siguiente:

Están bien si me dan una flexibilidad necesaria para no duplicar el step, pero no si no permite una clara legibilidad

Las expresiones regulares que nos van a servir en nuestras automatizaciones van a ser selectas, no muchas. Así que acá van las que más van a usar, cómo es su sintaxis y ejemplos!

**Anclaje**

Para evitar la ambigüedad en la definición de los steps, usar anclas (anchors) correctamente es esencial. Pensemos sobre lo siguiente: La expresión

**Given** I log in

va a matchear también con la siguiente si no colocamos los símbolos de comienzo y fin:

**Given** I log in as an almighty admin

Y ambas significan cosas muy diferentes en la funcionalidad. Para eso, vamos a utilizar los siguientes símbolos, los cuáles van a delimitar el comienzo y fin del string que define al step.

^ es usado para iniciar y $ para terminar. En definitiva, va a quedar:

**Given** **^**I log in**$**

De esta manera, estamos evitando el famoso error de Ambiguous Match.

**Comodines y cuantificadores.**

Vieron que cuando en un step definimos un término como un argumento que la función va a necesitar, en la clase de step defs nos aparece con unos símbolos estrambóticos? Bueno, esos son los comodines y cuantificadores. Por ejemplo:

**.\***Esto nos va a matchear cualquier String que le mandemos.

**.+** Esto nos va a matchear uno o más de cualquier cosa.

**d\*** Esto nos va a matchear cualquier dígito.

**d+** Esto nos va a matchear uno o más dígitos.

**"[^"]\*"** Esta aberración nos va a matchear algo que está entre comillas dobles.

**an?** Esto nos va a matchear un a o an y puede ir sin signo de pregunta. Tranquilos que vamos a verlo más claro en ejemplos.

#### **Grupos de captura: Identificando los argumentos en nuestros steps.**

Siempre que tengamos uno de esos comodines o cuantificadores entre paréntesis, significa que eso va a ser requerido como argumento en nuestra función que defina al step. Por ejemplo, tener

[**Given**(**@"^**I'm logged in as an**? (.\*)$"**)]

Va a significar que nuestra función va a requerir un String. También que podemos escribir el step como "I'm logged in as a customer" o "I'm logged in as an admin" ya que dijimos que a o an están ambas bien.

Si tuviésemos:

[**Given**(**@"^**I have **(d\*)$"**)]

Estaría pidiendo por un solo dígito, como integer, en el argumento de la función que define este step. Para poder aceptar cualquier número, deberíamos escribir:

[**Given**(**@"^**I have **(d+)$"**)]

Ahora, para ir cerrando...imaginemos que quiero establecer un opcional lógico pero no quiero que sea un grupo de captura. Osea, no quiero que eso signifique tener que agregar un nuevo argumento a la función. Por ejemplo, el siguiente caso:

**When** I log in as an admin **Given** I'm logged in as an admin

Ambos hacen lo mismo, pero están escritos diferente. Lo que puedo hacer es lo siguiente:

[**When**(**@"^(**I'm logged**|**I log**)** in as **an? (.\*)$")**]  
**public** **void** LogInAs(**string** role)  
{  
// código para el step  
}

Pero esto no va a buildear amigos. Saben por qué? Porque la función está pidiendo un solo argumento, el role, mientras que estamos capturando dos grupos: El primero es el OR lógico que tenemos aceptando **I'm logged** o **I log**. Como verán este opcional lógico se construye con el carácter **|**entre los paréntesis y el segundo es el String que viene después del an.

Para solucionar ésto, vamos a necesitar agregar algo al grupo de captura que tenemos primero para que...bueno, no capture nada y no pida un argumento en la función.

[**When**(**@"^(?:**I'm logged**|**I log**)** in as **an? (.\*)$")**]  
**public** **void** LogInAs(**string** role)  
{  
// código para el step  
}

Como verán, la diferencia es sutil, muy sutil. Simplemente agregamos el ?: al comienzo del opcional lógico para indicar que no queremos que ésto se traduzca en un argumento requerido.

#### **Conclusión y machete:**

Como verán, hay todo un submundo para la construcción de Steps de manera eficiente y prolija con Cucumber. También tengan en cuenta lo que les dije sobre evitar el abuso de estos recursos, lo que puede terminar resultando en frases Gherkin demasiado complejas, uno de los grandes problemas en mi opinión en cualquier proyecto de Automation. Para que vean que hay mucho más por cubrir, les voy a dejar el siguiente machete, cortesía de algún sitio web de Automation de la red de redes

Calendario

Descripción generada automáticamente con confianza media