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

Calendar

Description automatically generated with medium confidence