Behave

* **Instalación**

Tener instalado pip

&> pip install behave

* **Estructura básica**

features/

features/tutorial.feature

features/environment.py

features/steps/

features/steps/tutorial.py

En el fichero tutorial.feature se definen los scenarios del feature, en el fichero tutorial.py se implementan los steps para pasar el escenario y el fichero environment.py es un fichero donde podemos definir código que se ejecutará antes o después de ciertos eventos.

* **Estructura de nuestro proyecto y descripción de ficheros**

behave.ini

.gitignore

commons/

commons/environment\_commons.py

commons/page\_model.py

features/

features/cancel\_purchase.feature

features/help.feature

features/login.feature

features/login\_delegated.feature

features/myaccount.feature

features/navigation.feature

features/purchase.feature

features/reset\_pass.feature

features/signup.feature

features/SSOFunambol.feature

features/update\_purchase.feature

features/vouchers.feature

…

…

features/environment.py

features/steps/

features/steps/steps\_commons.py

features/steps/steps\_co.py

features/steps/steps\_uk.py

pages/

pages/basic\_page/

pages/basic\_page/…

pages/co\_pages/

pages/co\_pages/…

pages/o2\_pages/

pages/o2\_pages/…

reports/

resources/

resources/logger\_utils.py

resources/logging.conf

resources/properties.json

Por cada funcionalidad básica tenemos un feature. Todos los steps necesarios están definidos en el fichero steps\_commons.py excepto alguno que daba conflicto y se han definido en steps\_co.py y steps\_uk.py.

En los ficheros contenidos en pages/basic\_page/, pages/co\_pages/ y pages/o2\_pages/ están definidos las funciones para acceder e interactuar con los elementos de las diferentes páginas web.

En reports/ se guardan los ficheros XML generados por behave con el resultado de los test.

El fichero **logger\_utils.py** tenemos una función para inicializar el logger. **logging.conf** es el fichero de configuración del logger, tal y como está configurado, por consola solo muestra los mensajes críticos y por fichero todos los niveles del logging, además este fichero se sobrescribe cada vez que se lanza behave. También está configurado el formato de los mensajes.

En **properties.json** están las direcciones de acceso a las webs de UK y Colombia (normalmente NV), la dirección de Admin BE. El valor “testing” lo utilizamos para indicar donde queremos lanzar behave, las opciones son “local”, “Ruben” (el Selenium Grid que tiene montado el compañero Rúben” o “CBT“ (para lanzarlo en el Selenium de la herramienta CrossBrowserTesting que no funciona). El valor “proxy” tiene como opciones “ON” y “OFF”, si estamos dentro de TCP deberá estar en “ON”.

**behave.ini** es un fichero de configuración de behave donde actualmente tenemos el color a True, que nos saque los reports con junit a True además del directorio donde queremos que los guarde. En cuanto a los tags tenemos configurado que no lance los scenarios tageados con “no” ya que en este momento no nos interesan o corresponden al login delegado así como los de colombia, ya que actualmente no está disponible este entorno. show\_skipped está a False para que no muestre por consola los scenarios que no se ejecutan. show\_timings esta a False para que no se impriman los tiempos de cada steps, ya que es algo innecesario. Las dos últimas líneas corresponden a la configuración para poder rerunear los scenarios fallados.

[behave]  
color = 1  
junit = 1  
junit-directory = reports  
tags = ~@no  
 ~@co  
  
show\_skipped = 0  
show\_timings = 0  
format = rerun  
outfiles = rerun\_failing.features

Esto es equivalente a escribirlo en la línea de comandos, por lo que con este fichero solo es necesario lanzar behave sin parámetros.

* **Environment.py**

Los métodos que podemos implementar en el environment.py son:

before\_step(context, step), after\_step(context, step)

before\_scenario(context, scenario), after\_scenario(context, scenario)

before\_feature(context, feature), after\_feature(context, feature)

before\_tag(context, tag), after\_tag(context, tag)

before\_all(context), after\_all(context)

En nuestro proyecto tenemos implementadas las funciones before\_all que llama al setup definido en el fichero environment\_common.py donde lee el fichero de configuración e incializa el driver, y la función after\_all donde llama a los métodos (implementados en environment\_common.py) de borrado de customers y cierre del navegador y driver.

* **Feature Files**

Feature: feature name

Scenario: some scenario

Given some condition

Then some result is expected.

[http://pythonhosted.org/behave/tutorial.html#feature-files](http://pythonhosted.org/behave/tutorial.html)

* **Python Step Implementations**

Se implementan dentro del directorio steps y no hace falta decirle a behave cual usar, los usará todos.

Los steps se identifican con los decoradores que coinciden con la frase del scenario (@given, @when o @then), o podemos usar el genérico @step. En nuestro proyecto todos los steps tienen el decorador @step en lugar del correspondiente, ya que varios de ellos se reutilizan en varios features siendo en cada uno de ellos un paso diferente.  
El decorador acepta una cadena que contiene el resto de la frase utilizada en el step del escenario al que pertenece.

Podemos definir un step que realmente sean varios steps por ejemplo: “**I buy the offer o2ss8”** es un step que se compone de varios. Para implementar este step usaremos el método del context: execute\_steps()

@step(u'I buy the offer {offer}')  
def I\_buy\_offer(context,offer):  
 context.execute\_steps((u"""  
 given I chose {offer} subscription  
 and I accept the terms  
 and I confirm the purchase  
 then the widget of my {offer} is active  
 """).format(offer=offer.encode('utf-8')))

* **Paso de parametros**

En los Features, si utilizamos Scenario Outline con la sección de Examples, los parámetros se indican en el step de esta manera: <param>

En la implementación de los steps se indican con: {param}

* **Context**

En un lugar donde podemos almacenar información y almacenar a ella donde queramos, ya que en cada step tenemos que pasar esta variable como parámetro. Lo podemos usar para pasar valores entre steps.

Behave tambien añade valores a esta variable:

table: contiene cualquier table de datos asoaciada con un step. Ejemplo de uso:

for row in context.table:  
 model.add\_user(name=row['name'], department=row['department'])

text: contiene cualquier texto multinea de un step

failed: se establece cuando el step falla.

* **Tags**

Podemos tagear scenarios y los features con @tag.  
Para lanzar los scenarios o features que estén tageados con el tag elegido ejecutamos behave con los parámetros:   
-t @tag  
 y si queremos lanzar todos los scenarios excepto los tageados con ese tag:  
 -t @-tag o –t ~@tag

Y si queremos lanzar los scenararios que estén tageados con @tag1 o @tag2:

-t @tag1,@tag2

Los tags asignados a los scenarios estas disponibles en las funciones del environment a traves de los objetos “feature” o “scenario” que son pasados a las funciones. En estos objetos existe el atributo tags que es una lista de todos los tags en el orden en el que se encuentran en el fichero del feature.

**8.1 Tags en nuestro proyecto**

Tenemos tageados todos los features con el mismo nombre del fichero, para lanzar un feature de forma más cómoda por ejemplo, el lugar de lanzar:

$>behave features/cancel\_purchase.feature

Lanzamos

$>behave –t @cancel\_purchase

Todos los scenarios están tageados con @o2 o @co según a la OB a la que pertenezcan. Los scenarios que utilizan el login delegado tienen la etiqueta @delegated así como estos y los correspondientes a reset password y los vouchers tienen la etiqueta @no.

Para runear los scenarios fallados ejecutamos:  
  
$>behave @rerun\_failing.feature

* **Otros**

Script