Informe de la práctica de Laboratorio Nº5: Behaviour Driven Development en el framework de Django.

Autor: Adán Rafael López Lecuona.

Objetivo de la práctica

El objetivo de esta práctica es introducir las técnicas de desarrollo dirigido por pruebas, métodos de limpieza de código y demás técnicas para el mantenimiento eficiente de código.

Para realizar este ejercicio, debemos aprender a utilizar las herramientas BDD que tenemos para Python. En este caso utilizaremos lettuce para aplicar la metodología BDD.

Primero tuve que instalar lettuce que me dio bastantes problemas y lo instalé manualmente clonándolo de un repositorio remoto hacia el mío local.

user@machine:~/Downloads$ git clone git://github.com/gabrielfalcao/lettuce.git

user@machine:~/Downloads$ cd lettuce

user@machine:~/Downloads/lettuce$ sudo python setup.py install

Una vez ya clonado e instalado el setup.py, lo clonamos a la carpeta de proyectosDjango . Una vez dentro de proyectosDjango añadimos en el PYTHONPATH lo siguiente por consola:

$echo “export PYTHONPATH=$HOME/DSI/proyectosDjango/lettuce:$_PYTHONPATH" >> $HOME/.

Modificamos en el archivo settings.py introducimos en Installed_apps a lettuce.django.

Vamos a realizar nuestro primer test.

Lo primero que hacemos es modificar el archivo views.py my_app introduciendo:

```
# my_app/views.py
from django.http import HttpResponse

def quick_test(request):

        return HttpResponse("Hello testing world!");
```

Luego modificamos el archive urls.py:

```
# lettuce/urls.py
from django.conf.urls import patterns, include, url

from my_app.views import quick_test

urlpatterns = patterns('',
    url(r'^quick-test/$', quick_test),
)
```

Luego creo dos archivos dentro de my_app que son test.feature y terrain.py. Dentro de test_feature introducimos lo siguiente:

```
Feature: Test
        As someone new to testing
        So I can learn behavior driven development
        I want to write some scenarios

        Scenario: I can view the test page
                Given I am at "/quick-test/"

                Then I should see "Hello testing world!"
```

En el archivo terrain.py introducimos lo siguiente:

```
from django.core.management import call_command
from django.test.simple import DjangoTestSuiteRunner

from lettuce import before, after, world
from logging import getLogger
from selenium import webdriver

try:
        from south.management.commands import patch_for_test_db_setup
except:
        pass

logger = getLogger(__name__)
logger.info("Loading the terrain file...")

@before.runserver
def setup_database(actual_server):
        '''
        This will setup your database, sync it, and run migrations if you are using South.
        It does this before the Test Django server is set up.
        '''
        logger.info("Setting up a test database...")
```

```
        # Uncomment if you are using South
        # patch_for_test_db_setup()

        world.test_runner = DjangoTestSuiteRunner(interactive=False)
        DjangoTestSuiteRunner.setup_test_environment(world.test_runner)
        world.created_db = DjangoTestSuiteRunner.setup_databases(world.test_runner)

        call_command('syncdb', interactive=False, verbosity=0)

        # Uncomment if you are using South
        # call_command('migrate', interactive=False, verbosity=0)

@after.runserver
def teardown_database(actual_server):
        '''
        This will destroy your test database after all of your tests have executed.
        '''
        logger.info("Destroying the test database ...")

        DjangoTestSuiteRunner.teardown_databases(world.test_runner,
world.created_db)

@before.all
def setup_browser():
        world.browser = webdriver.Firefox()

@after.all
def teardown_browser(total):
        world.browser.quit()
```

En el archivo settings.py tenemos que añadir lo siguiente:

```
# Nose
TEST_RUNNER = 'django_nose.NoseTestSuiteRunner'

# Lettuce
LETTUCE_SERVER_PORT = 9000
```

Creamos dentro de my_app una carpeta llamada templates, y dentro de ésta creamos nuestra página de inicio base.html con el siguiente código de html dentro:

```
<html>
        <head>
                <title>Learning Lettuce!</title>
        </head>
        <body id='content'>
                {% block content %}{% endblock %}
```

```
        </body>

</html>
```

Ahora dentro de templates creamos otro archivo html llamado blog.html con el siguiente código:

```
{% extends "base.html" %}

{% block content %}
Hello testing world!

{% endblock %}
```

Necesitamos actualizar el archivo de las vistas views.py de nuestra app con lo siguiente:

```
from django.shortcuts import render_to_response

def quick_test(request):

        return render_to_response("blog.html", {})
```

También debemos modificar en el archive de configuración el siguiente código:

```
## Add this at the top of settings.py
import os.path
root = os.path.dirname(__file__).replace('\\','/')

## Make your TEMPLATE_DIRS variable look like this
TEMPLATE_DIRS = (
    root + "/../blog/templates/",
)
```

Actualizamos nuestro fichero terrain.py con el siguiente código :

```
@step(u'I am at "([^"]*)"')
def i_am_at_url(step, url):

    world.browser.get(url)

@step(u'I should see "([^"]*)"')
def i_should_see_content(step, content):
        if content not in world.browser.find_element_by_id("content").text:

            raise Exception("Content not found.")
```

Corremos los tests y vemos que son pasados correctamente.

Realizamos otro ejercicio de ejemplo para realizar el factorial de números propuesto en el tutorial de lettuce.

Con lo creado anteriormente simplemente cambiamos los nombres y modificamos los archivos correspondientes.

Por lo tanto creamos los archives zero.feature y steps.py que están dentro de este árbol de directorios:

/home/DSI/projectosDjango/lettuce/mymath

| tests

| features

- zero.feature

- steps.py

Modificamos los dos archivos, en el zero.feature ponemos la implementación del factorial cuando un número es 0. Este archivo siempre tiene que estar dentro del directorio features y debe tener la extensión .feature. En este archivo ponemos lo que esperamos del comportamiento del factorial:

Feature: Compute factorial

In order to play with Lettuce

As beginners

We'll implement factorial

Scenario: Factorial of 0

Given I have the number 0

When I compute its factorial

Then I see the number 1

Ahora definimos los pasos del escenario. Así la herramienta lettuce puede entender el comportamiento de la descripción. En el archivo steps.py introducimos el código que describe los pasos:

from lettuce **import** *

```
@step('I have the number (\d+)')
def have_the_number(step, number):
    world.number = int(number)
@step('I compute its factorial')
def compute_its_factorial(step):
    world.number = factorial(world.number
@step('I see the number (\d+)')
def check_number(step, expected):
    expected = int(expected)
    assert world.number == expected, \
        "Got %d" % world.number
def factorial(number):return -1
```

El último return -1 nos dará errores AssertionError: Got -1.

Ahora realizamos el cambio en el return -1 en un return 1 y vemos que el error se ha solucionado.

Vamos añadiendo steps en el fichero steps.py y también tendremos que cambiar lo que devuelve en steps.py , quedando finalmente el código de la definición del factorial en el archivo steps.py así:

```
def factorial(number):
   number = int(number)
    if (number == 0) or (number == 1):
        return 1
    else:
        return number*factorial(number-1)
```

Podemos ver que para los steps 0 y 1 retornamos la primera condición y para los demás la otra.

Con lettuce seguimos un círculo de acciones para comprobar que funciona todo correctamente en los tests que pasamos a nuestro código:

1º describe behaviour.

2º define steps in Python.

3º run and watch it fail.

4º write code to make it pass.

5º run and watch it pass.

Creación de páginas estáticas con el framework de Django

Dentro de de proyecto de Django my_app
Lo primero que hago es entrar en settings.py y añado 'django.contrib.flatpages' en Installed_apps.
Modificamos en el fichero proyectosDjango/my_app/views.py para para crear una vista que nos permita que salga por pantalla las vistas que queramos mostrar:

```
from django.http import HttpResponse
from django.template import loader, Context

def home_view (request):
   t=loader.get_template("home.html")
   c=Context({})
   return HttpResponse(t.render(c))
```

Ahora queremos generar una nueva URL de la vista en el fichero djangoproject/urls.py añadimos las urls :

```
from django.conf.urls.defaults import patterns, include, url
from proyectosDjango.my_app.views import home
from my_app.views import quick_test

 from django.contrib import adminadmin.autodiscover()

urlpatterns = patterns('',

     ('r^proyqctosDjango/my_app','home_view'),
```

```
urlpatterns = patterns('',
```

```
    url(r'^quick-test/$', quick_test),
)
```

En el fichero proyectosDjango /templates/home.html introduzco el código de mi vista de página de inicio en html :

```
<!DOCTYPE html>
<html>
    <head>
      <title> Esta es mi página de inicio </title>
    </head>
    <body>
        <h1>Bienvenidos a DSI</h1>
    </body>
</html>
```

Instalamos los paquetes python-lxml.
Podemos ver la página home generándolo con nuestro servidor:
En nuestra proyecto my_app creamos los archivos home.features y home.py:

```
from lettuce import *
from lxml import html
from django.test.client import Client
from nose.tools import assert_equals

@before.all
def set_browser():
    world.browser = Client()
@step(r'I access the url "(.*)"')
def access_url(step, url):
    response = world.browser.get(url)
    world.dom = html.fromstring(response.content)
@step(r'I see the header "(.*)"')
def see_header(step, text):
    header = world.dom.cssselect('h1')[0]
    assert header.text == text
```

En el fichero my_app/features/home.features introducimos lo siguiente:

```
Feature: Rocking with lettuce and django

    Scenario: Simple pagina estatica Django
        Given I access the url "http://127.0.0.1:8000/proyectosDjango/my_app"
        Then I see the header "<h1>Bienvenidos a DSI</h1>"
```

Una vez creados los archivos, ejecutamos los tests y vemos que nada falla.

Con esto terminamos un primer acercamiento para la creación de páginas estáticas con Django.