Universidad Autónoma de Madrid

Departamento de Informática

Proyecto de Sistemas Informáticos
Práctica - 2

Roberto Marabini Ruiz

Índice

1. Introducción

El principal objetivo de esta práctica es familiarizarse con el framework *Django* y con la plataforma *Heroku*. En esta práctica utilizáremos como libro de referencia *Tango with Django 2* el cual está accesible en *Moodle*. si queréis instalar *Django* en vuestro ordenador hay una descripción de como hacerlo en los capítulos titulados *Setting up your System Installing Python 3* y *Getting Ready to Tango* del libro de referencia.

Para llevar control de las distintas versiones del código usaremos `git`. Probablemente, la forma más sencilla de empezar a usar git sea crear un repositorio en *Github* y luego clonarlo (`git clone ...`). Finalmente, haría falta crear un proyecto en *pycharm* sobre el repositorio clonado. Para ello, en el menú principal selecciona File → Open y en la ventana de diálogo selecciona el directorio que contenga el repositorio. Si necesitáis ayuda para realizar este paso pedidla, os la daremos encantados. En el libro de referencia tenéis un capítulo titulado *A Git Crash Course* el cual describe el uso de git.

2. Descripción del Trabajo a Realizar Durante la Primera Semana de la Práctica

Leed los capítulos *Django Basics* y *Templates and Static Media* del tutorial *Tango with Django* ejecutando los ejemplos descritos. Realizad los ejercicios propuestos al final de cada capítulo. Subid el código creado a un NUEVO repositorio en *Github*. No reuséis el repositorio creado en la práctica anterior.

Nota: en uno de los ejercicios se os solicita que busquéis una imagen para mostrarla en una página web. Por algún motivo existe una tendencia a buscar imágenes de alta definición que ocupan varios mega-bytes. Estas imágenes, por lo general, no son las más adecuadas para ser usadas en una página web.

2.1. Tests

En la página *Moodle* de la asignatura se encuentra el fichero `tests.py` y `tests_first_day` los cuales contienen una colección de tests. Para ejecutar los tests copiad estos fichero en el directorio rango y teclead:

```
python ./manage.py test rango.tests.GeneralTests
python ./manage.py test rango.tests.IndexPageTests
python ./manage.py test rango.tests.AboutPageTests
```

Listado 1: Output of the coverage command

Name	Stmts	Miss	Cover	Missing
rango/__init__.py	0	0	100 %	
rango/admin.py	1	0	100 %	
rango/apps.py	3	3	0 %	1–5
rango/migrations/__init__.py	0	0	100 %	
rango/models.py	1	0	100 %	
rango/urls.py	3	0	100 %	
rango/views.py	8	0	100 %	
TOTAL	18	5	80 %	

No modifiquéis el fichero con los tests, si algún test no se satisface modificar vuestro proyecto.

2.1.1. Cobertura de los Tests (coverage)

La utilidad `coverage` nos permite comprobar el porcentaje de código al que accedemos desde los tests. Es decir, cuanto código estamos realmente probando con nuestros tests.

Si ejecutas los comandos (el símbolo ␣ marca explicitamente los espacios)

```
#␣erase␣coverage␣data␣before␣running␣test␣case
coverage␣erase
#␣run␣test␣in␣rango␣application
coverage␣run␣--omit="*/test*"␣--source=rango␣./manage.py␣test␣rango.tests
#␣show␣test␣coverage␣results
coverage␣report␣-m␣-i
```

Obtendrás un resultado similar al mostrado en Listado 1:

Conseguir una cobertura del 100 % es difícil pero deberíamos intentar mantener en, al menos, un 75 %

3. Resumen del Trabajo a Realizar Durante la Primera Semana de la práctica

Ejercicio 1: Introducción a Django-I

1. Subid vuestro proyecto a *Github* usando `git`. Usad el fichero `.gitignore` para evitar subir los ficheros con extensión `pyc`
2. El proyecto de *Django* debe contener los ejercicios solicitados en los capítulo *Django Basics* y *Templates and Static Media*.
3. El projecto de *Django* debe: (a) satisfacer todos los tests enumerados en la sección previa y (b) seguir las directivas *PEP8*. Esto es, si se comprueba el código con el analizador llamado `flake8`, la salida por pantalla no debe contener ni errores ni advertencias (puedes ignorar las advertencias relacionas con lineas de código producidas automáticamente por *Django*).
4. En estos ejercicios se hace referencia a una aplicación desarrollada en la "official Django tutorial". No hace falta que entreguéis la susodicha aplicación.

4. Descripción del Trabajo a Realizar Durante la Segunda Semana de la práctica

Leed los capítulos *Models and Databases* y *Models, Templates and Views* del tutorial *Tango with Django* ejecutando los ejemplos descritos. Al finalizar cada capítulo, realizad los ejercicios propuestos. En estos ejercicios se hace referencia a una aplicación desarrollada en la *official Django tutorial*. No hace falta que entreguéis la susodicha aplicación.

IMPORTANTE-1: durante la realización de los ejercicios tendréis que crear un superusuario para acceder a la base de datos usando el comando `python manage.py createsuperuser` utilizad `alumnodb` tanto para el nombre como para la clave. Usad `psi` como nombre de la base de datos asociada al proyecto.

IMPORTANTE-2: como base de datos usad `PostgreSQL` en lugar de `sqlite3` que es la opción por defecto. En la subsección *Telling Django about your database* se

describe la variable `DATABASES` la cual se deberá configurar en el fichero `settings.py` de forma similar a la mostrada en el listado siguiente.

```
#define an enviroment variable called DATABASE_URL and use it to
    initializate DATABASES
#dictionary
#export DATABASE_URL = 'postgres://alumnodb:alumnodb@localhost:5432/psi
    '
import dj_database_url
DATABASES['default'] = dj_database_url.config(default='postgres://
    alumnodb:alumnodb@localhost:5432/psi')
```

4.1. Persistencia de las Bases de Datos

A diferencia de lo que ocurre con los datos existentes en vuestra cuenta, las bases de datos creadas usando PostgreSQL no se borran al apagar el equipo. Por ello, es posible que os encontreis en el ordenador una base de datos ya existente llamada *psi* con una estructura diferente a la que necesitáis. Para evitar conflictos, os recomendamos que al principio de cada sesión borréis la base de datos *psi*. Para borrar la base de datos podéis usar el comando `dropdb -U alumnodb -h localhost psi`, y a continuación recrear la base con `createdb -U alumnodb -h localhost psi`.

4.2. Tests

En la página *Moodle* de la asignatura se encuentra el fichero `tests_second_day.py` en el cual están definidos los tests relacionados con el segundo día de la práctica. Para ejecutar los tests copiad este fichero en el directorio rango y teclead:

```
python ./manage.py test rango.tests.GeneralTests
python ./manage.py test rango.tests.IndexPageTests  --keepdb -v 3
python ./manage.py test rango.tests.AboutPageTests  --keepdb -v 3
python ./manage.py test rango.tests.ModelTests  --keepdb -v 3
python ./manage.py test rango.tests.Chapter5ViewTests  --keepdb -v 3
```

No modifiquéis el fichero con los tests, si algún test no se satisface modificar vuestro proyecto.

Necesitaréis descomentar la línea `from rango.tests_second_day import Chapter5ViewTests` en el fichero `tests.py` para poder ejecutar los nuevos tests.

5. Resumen del Trabajo a ENTREGAR Durante la Segunda Semana de la práctica

Ejercicio 2: Introducción a Django-II

1. Subid vuestro proyecto a *Github* usando `git`.
2. Usando *Moodle* entregad un proyecto de Django que contenga los ejercicios solicitados en los capítulos *Models and Databases* y *Models, Templates and Views*. En concreto se deberá subir a *Moodle* el fichero obtenido ejecutando el comando `git archive --format zip --output ../assign2_second_week.zip master` desde el directorio del projecto (`tango_with_django_project`)
3. Aseguraros de que todos los tests definidos en el fichero *tests.py* son satisfechos por vuestros código.
4. Comprobad el código con `flake8`
5. Calcula la cobertura de los tests e inserta en la memoria de la practica una tabla similar a la que aparece en el Listado 1
6. En estos ejercicios se hace referencia a una aplicación desarrollada en la "official Django tutorial". No hace falta que entreguéis la susodicha aplicación.

6. Descripción del Trabajo a Realizar Durante la Tercera Semana de la práctica

Leed los capítulos *Forms* y *Working with Templates* ejecutando los ejemplos descritos. En el capítulo *Forms* implementar el ejercicio *Now let users add pages to each category, see below for some example code and hints.* En el capítulo *Working with Templates* realizad los ejercicios propuestos.

7. Resumen del Trabajo a Realizar Durante la Tercera Semana de la práctica

Ejercicio 3: Introducción a Django-II

1. Mantened vuestro repositorio en *Github* actualizado.
2. Realizad los ejercicios solicitados.
3. Aseguraros de que todos los tests desarrollados son satisfechos por vuestro código

8. Cuarta semana: Autenticación de Usuarios y Desplegando Aplicaciones en *Heroku*

En esta semana vamos a aprender a autenticar usuarios y a desplegar aplicaciones Web realizada en *Django* en *Heroku*. *Heroku* es una "plataforma como servicio" de computación en la nube (PaaS) que nos permite desplegar nuestra aplicación en la nube de manera gratuita.

Nota: la documentación de Heroku está basada en el post: `https://amatellanes.wordpress.com/2014/02/25/django-heroku-desplegando-una-aplicacion-django-en-heroku/`

8.1. Autenticación de usuarios

Leed el capítulo titulado "User Authentication" ejecutando los ejemplos descritos e implementad los ejercicio propuestos al final del capítulo.

8.1.1. Tests

En la página *Moodle* de la asignatura se encuentra el fichero `tests_fourth_day.py` en el cual están definidos los tests relacionados con el cuarto día de la práctica. Para ejecutar los tests copiad este fichero en el directorio rango y teclead:

```
python ./manage.py test rango.tests.GeneralTests
python ./manage.py test rango.tests.IndexPageTests  --keepdb -v 3
python ./manage.py test rango.tests.AboutPageTests  --keepdb -v 3
python ./manage.py test rango.tests.ModelTests  --keepdb -v 3
```

```
python ./manage.py test rango.tests.Chapter5ViewTests  --keepdb -v 3
python ./manage.py test rango.tests.UserAuthenticationTests  --keepdb -v 3
```

No modifiquéis el fichero con los tests, si algún test no se satisface modificar vuestro proyecto.

Necesitaréis descomentar la línea `from rango.tests_fourth_day import UserAuthenticationTests` en el fichero `tests.py` para poder ejecutar los nuevos tests.

8.2. *Heroku*

8.2.1. Creando una cuenta en Heroku

El primer paso a realizar es crear una cuenta de usuario en Heroku que puedes conseguir de manera gratuita en `https://signup.heroku.com/identity`.

8.2.2. Definiendo Nuestro Procfile y runtime.txt

Vamos a hacer uso de un fichero llamado `Procfile`. Este fichero debe localizarse en el directorio raíz del proyecto (`tango_with_django_project`) y es donde declaramos los comandos que deberían ser ejecutados al arrancar nuestra aplicación.

Para que nuestra aplicación se ejecute, lo primero que deberemos hacer es arrancar Gunicorn (servidor de Django). Para ello creamos el fichero `Procfile` y añadimos lo siguiente:

```
web: gunicorn tango_with_django_project.wsgi --log-file -
```

Igualmente debemos decirle a Heroku qué versión de python deseamos utilizar. Para ello se creará el fichero `runtime.txt` y se escribirá una única linea con la versión de python deseada. Por ejemplo: `python-3.6.8`, usad la misma versión de python que haya en los laboratorios.

8.2.3. Definiendo las Dependencias con Pip

Heroku reconoce una aplicación Python por la existencia de un fichero `requirements.txt` en el directorio raíz del repositorio. En este fichero se especifican los módulos Python adicionales que nuestra aplicación necesita. El fichero se puede crear de forma automática tecleando el comando

```
pip freeze > requirements.txt
```

el contenido del fichero resultante se debe parecer al Listado 2

Listado 2: File requirements.txt

```
coverage==4.5.3
dj-database-url==0.5.0
dj-static==0.0.6
Django==2.1.7
entrypoints==0.3
flake8==3.7.7
gunicorn==19.9.0
image==1.5.27
mccabe==0.6.1
Pillow==5.4.1
# https://github.com/pypa/pip/issues/4022
# pkg-resources==0.0.0
psycopg2==2.7.7
psycopg2-binary==2.7.7
pycodestyle==2.5.0
pyflakes==2.1.1
pytz==2018.9
static3==0.7.0
```

pudiendo variar el número de versión. Si existen más líneas con diferentes paquetes os recomendamos que las borréis. No os olvidéis de añadir la linea que empieza por `gunicorn`.

8.2.4. Configurando la Aplicación para que se Pueda Desplegar en Heroku

El siguiente paso es configurar nuestra aplicación Django empezando por la configuración de la base de datos Postgres que usaremos en Heroku. En python tenemos instalado un módulo llamado `dj-database-url`, este módulo convierte la variable de entorno `DATABASE_URL` en código que entiende nuestra aplicación Django. Tendremos que añadir las siguientes lineas de código al final de nuestro fichero `settings.py`:

```
# Parse database configuration from $DATABASE_URL
import dj_database_url
DATABASES['default']= dj_database_url.config(default='postgres://alumnodb:alumnodb@lo
STATIC_ROOT = 'staticfiles'
```

A continuación modificamos el fichero `tango_with_django_project/wsgi.py` añadiendo el siguiente código:

```
from django.core.wsgi import get_wsgi_application
```

```
from dj_static import Cling

application = Cling(get_wsgi_application())
```

Con estos pasos queda configurada nuestra aplicación para ser desplegada en Heroku.

8.2.5. Creando un repositorio Git para la aplicación

Nuestra aplicación ya debería estar guardada en un repositorio git. De todas formas, repetimos a continuación los pasos a ejecutar. IMPORTANTE: el directorio `.git` debe estar en el mismo directorio que la carpeta con la aplicación.

El primer paso para crear nuestro repositorio es definir un fichero oculto .gitignore donde definiremos aquellos ficheros y directorios que queremos que sean ignorados por nuestro repositorio. Nos aseguramos que nuestro fichero .gitignore contenga, al menos, las siguientes restricciones:

```
*.pyc
staticfiles
uploads
```

A continuación, si no estamos usando un repository en *Github* creamos nuestro repositorio y guardamos nuestros cambios, en caso contrario se puede ignorar este paso:

```
$ git init
Initialized empty Git repository in /home/xxx/yyy/.git/

$ git add .

$ git commit -am 'initial commit'
[master (root-commit) da56753] initial commit
 6 files changed, 128 insertions(+)
 create mode 100644 Procfile
 create mode 100644 hellodjango/__init__.py
 create mode 100644 hellodjango/requirements.txt
 create mode 100644 hellodjango/settings.py
 create mode 100644 hellodjango/urls.py
 create mode 100644 hellodjango/wsgi.py
 create mode 100644 manage.py
```

8.2.6. Desplegando la aplicación en Heroku

El siguiente paso es subir la aplicación que acabamos de crear a un repositorio de Heroku. Para ello usamos el siguiente comando:

```
$ heroku login
Enter your Heroku credentials.
Email: python@example.com
Password:
...
$ heroku create
Creating gentle-gorge-9766... done, stack is cedar
http://gentle-gorge-9766.herokuapp.com/ | git@heroku.com:gentle-gorge-9766.git
Git remote heroku added
```

Este comando ha creado un repositorio remoto en Heroku. Como último paso antes de subir el código a Heroku debemos editar una vez más el fichero `settings.py` y modificar la variable `ALLOWED_HOSTS` añadiendo el nombre del host en el que se va a ejecutar la aplicación (el cual es devuelto al ejecutar el comando `heroku create`). En este ejemplo el host es `gentle-gorge-9766.herokuapp.com`

```
ALLOWED_HOSTS = [u'pure-bayou-13155.herokuapp.com', u'localhost', u'127.0.0.1']
```

Usando git "comite" esta última modificación (`git commit -m ''modify ALLOWED_HOST variable'' nombredelprojecto/settings.py`) y sube la aplicación usando el siguiente comando:

```
$ git push heroku master

Initializing repository, done.
Counting objects: 11, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (11/11), 2.36 KiB, done.
Total 11 (delta 0), reused 0 (delta 0)

-----> Python app detected
-----> No runtime.txt provided; assuming python-2.7.6.
-----> Preparing Python runtime (python-2.7.6)
```

```
-----> Installing Setuptools (2.1)
-----> Installing Pip (1.5.4)
-----> Installing dependencies using Pip (1.5.4)
       Downloading/unpacking Django==1.6.2 (from -r requirements.txt (line 1))
     ...
       Successfully installed Django dj-database-url dj-static gunicorn psycopg2 stat
       Cleaning up...

-----> Discovering process types
       Procfile declares types -> web

-----> Compressing... done, 34.8MB
-----> Launching... done, v5
       http://gentle-gorge-9766.herokuapp.com deployed to Heroku

To git@heroku.com:gentle-gorge-9766.git
* [new branch]      master -> master
```

El siguiente paso es arrancar la aplicación que ya está en Heroku. Para ello ejecutamos el siguiente comando:

```
$ heroku ps:scale web=1
```

Ahora puedes comprobar si tu aplicación se está ejecutando correctamente abriendo la dirección proporcionada por Heroku. También puedes ejecutar el siguiente comando:

```
$ heroku open
Opening gentle-gorge-9766... done
```

Muy importante: tu aplicación no funcionará hasta que no ejecutes el comando `python manage.py migrate` en *Heroku*. Una descripción de como hacerlo se encuentra en la subsección 8.2.8.

8.2.7. Comprobando los logs

Heroku permite consultar el log de nuestra aplicación. Para ello hay que ejecutar el comando:

```
$ heroku logs
2014-02-23T19:38:25+00:00 heroku[web.1]: State changed from created to starting
2014-02-23T19:38:29+00:00 heroku[web.1]: Starting process with command ‘gunicorn hell
2014-02-23T19:38:29+00:00 app[web.1]: Validating models...
2014-02-23T19:38:29+00:00 app[web.1]:
```

8.2.8. Usando el Shell en Heroku

Heroku nos proporciona el comando heroku run que nos permite ejecutar los comandos que usamos cuando trabajamos en local con nuestra aplicación Django,

Por ejemplo, podemos usar el comando heroku run para ejecutar la shell de Django y trabajar con los datos almacenados en nuestra aplicación desplegada en Heroku:

```
$ heroku run bash
```

Desde el shell podemos, por ejemplo, crear el esquema inicial de la base de datos con el siguiente comando:

```
$ python manage.py migrate
```

8.2.9. Ejecutando los test en Heroku

Como puede que hayáis notado, para ejecutar los test se crea una base de datos temporal. Desafortunadamente, los usuarios gratuitos de Heroku no tienen privilegios para crear bases de datos en postgres y por lo tanto los test fallarían al ser ejecutados.

Una solución a este problema sería usar en el proyecto sqlite3 en lugar de postgres. Esta solución en general no es aceptable porque tras 30 minutos de inacción Heroku borra el ordenador virtual en donde se ejecuta la aplicación y con ello se pierden los datos almacenados en sqlite3 (aunque no se pierden los datos almacenados en postgres). Por lo tanto para ejecutar nuestra aplicación tenemos que usar postgres pero no es el caso para ejecutar los test puesto que tras la finalización del test no es necesario persistir los datos. Por lo tanto la solución propuesta sería modificar el fichero `settings.py` de forma que si esta definida la variable de entorno “SQLITE” use la base de datos de sqlite y en caso contrario use la de postgres. El codigo necesario seria:

```
DATABASES = {}
if os.getenv('SQLITE', False):
    DATABASES['default'] = {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
        }
else:
    import dj_database_url
    DATABASES['default'] = dj_database_url.config(default='postgres://alumnodb:alumno
```

Antes de ejecutar los tests habría que: logear en *Heroku*, definir en la terminal la variable SQLITE `export SQLITE=1`, ejecutar los tests y una vez finalizada la ejecución habría que anular la definición `unset SQLITE`.

Tras desplegar tu proyecto en *Heroku*, ejecuta todos los tests y comprueba que funcionan correctamente.

8.2.10. Ficheros "estaticos" on heroku

Por diseño, Django en Heroku no coge los ficheros estáticos (css, js, imagenes, etc) del proyecto sino que exige que estén en un directorio aparte que él crea usando el comando `python manage.py collectstatic`. Para que funcione correctamente la generación, si todavía no lo has hecho, debes definir en el fichero `settings.py` las variables.

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticHeroku')
```

y añadir al fichero `urls.py`

```
from django.conf import settings
from django.conf.urls.static import static
...


urlpatterns += static(settings.STATIC_URL,\
                      document_root=settings.STATIC_ROOT)
```

9. Material a ENTREGAR al Finalizar la Práctica

1. Escribid una memoria llamada memoria.pdf de no más de dos páginas de extensión en formato pdf. En la memoria describid la arquitectura de Django. En particular se desea que comentéis cada página de la presentación titulada Arquitectura de Django (accesible en *Moodle*). Añadid este fichero al repositorio del proyecto. Esta memoria NO debe describir vuestro proyecto sino como funciona *Django*.

2. Subid a *Moodle*, en un único fichero zip, el proyecto de PyCharm conteniendo la aplicación desarrollada en esta práctica llamada `rango`. En concreto se deberá subir a *Moodle* el fichero obtenido ejecutando el comando `git archive --format zip --output ../assign2_final.zip master` desde el directorio del proyecto (`tango_with_django_project`)

3. Proporcionar en el fichero settings.py la direccion de *Heroku* en la que está accesible vuestra aplicación añadiéndola a la lista $ALLOWED\_HOSTS$. `ALLOWED_HOSTS = [u'xxxxxxxxxxxxx.herokuapp.com', u'127.0.0.1']`

4. Aseguraros de que **TODOS** los tests desarrollados para esta entrega son satisfechos por vuestros código (tanto localmente como en Heroku). No es admisible que se modifique el código de los tests.

```
# commandline that executes all tests
python ./manage.py test rango.tests --keepdb -v 3
```

5. Calcula la cobertura de los tests e inserta en la memoria de la practica una tabla similar a la que aparece en el Listado 1

9.1. Enlaces de Interés

- `https://devcenter.heroku.com/articles/getting-started-with-python#introduction`

10. Criterios de evaluación

La calificación de esta práctica es *Apto* o *No Apto*. Para aprobar es necesario satisfacer los siguientes criterios:

- El código creado para esta practica debe: (a) ejecutarse correctamente en los ordenadores de los laboratorio usando python 3.7 y *Django* 2.1.7 y (b) satisfacer todas las comprobaciones realizadas por `flake8`. Nota: excepcionalmente el apartado (b) puede no ser satisfecho por el código generado AUTOMÁTICAMENTE por *Django*.
- La aplicación rango descrita en el libro de referencia debe estar desplegada en *Heroku* (y debe ser accesible usando el URL proporcionado en el fichero `settings.py` a través de la variable $ALLOWED\_HOSTS$). En *Heroku* la base de datos debe estar poblada con el contenido del fichero populate.py
- El código creado en esta práctica debe estar almacenado en un repository PRIVADO en github.
- La memoria y el código necesario para ejecutar la aplicación deben entregarse en *Moodle* y debe ser posible ejecutarlos localmente en los laboratorios de informática.
- El código entregado debe satisfacer TODOS los tests.
- Se incluye en la meoria el resultado de ejecutar la utilidad `coverage` (vease Listado 1

NOTA: El código usado para corregir esta práctica será el subido a *Moodle*. En ningún caso se usará el código existente en el repositorio de *Github* o en Heroku.