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 entorno de desarrollo Django y Heroku. En esta práctica utilizáremos el tutorial "on line" *Tango with Django* accesible en el URL `http://www.tangowithdjango.com/` (usaremos la última versión del libro).

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 bitbucket 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, 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. 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. Al finalizar cada capítulo realizad los ejercicios propuestos. Subid el código creado a un NUEVO repositorio en Bitbucket. No reuseis 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 imagenes de alta defición que ocupan varios megabytes. Estas imágenes, por lo general, no son las más adecuadas para ser usadas en una pagina web.

3. Trabajo a Entregar Durante la Primera Semana de la práctica

Ejercicio 1: Introducción a Django-I

1. Subid vuestro proyecto a `bitbucket` usando `git`. Usad el fichero `.gitignore` para evitar subir los ficheros con extensión `pyc`
2. Usando moodle entregad un proyecto de Django que contenga los ejercicios solicitados en los capitulo *Django Basics* y *Templates and Static Media*.
3. En concreto se deberá subir a moodle el fichero obtenido ejecutando el comando `git archive --format zip --output ../assign2_first_week.zip master` desde el directorio del projecto (`tango_with_django_project`). (Recordad que git hará un backup únicamente de los ficheros que hayan sido "añadidos" -git add- y registrados -git commit-).
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. 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 capitulo 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. Use `psi` como nombre de la base de datos asociada al proyecto.

IMPORTANTE-2: como base de datos usad `postgres` 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()
```

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 postgres no se borran al apagar el equipo. Por ello es posible que os encontrais en el ordenador con una base de datos llamada *psi* con una estructura diferente a la que necesitáis. Por ello, y para evitar conflictos, os recomendamos que al principio de cada sesión borréis la base de datos *psi*. Para borrar la base podéis usar el comando `dropdb -U alumnodb -h localhost psi`, 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.py`. Aseguraros de que TODOS los tests definidos en las clases *GeneralTests*, *IndexPageTests*,*AboutPageTests* y *ModelTests* son satisfechos por vuestros código. No modifiquéis el fichero con los tests, si alguno test no se satisface modificar vuestro proyecto.

Para ejecutar los tests copiad este fichero en el directorio rango y teclead:

```
python ./manage.py test rango.tests.GeneralTests  --keepdb -v 3
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.Chapter4ViewTests  --keepdb -v 3
python ./manage.py test rango.tests.Chapter5ViewTests  --keepdb -v 3
```

5. Trabajo a Entregar Durante la Segunda Semana de la práctica

Ejercicio 2: Introducción a Django-II

1. Subid vuestro proyecto a `bitbucket` usando `git`.
2. Usando moodle entregad un proyecto de Django que contenga los ejercicios solicitados en los capitulos *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.

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

Leed el capítulo *Fun with Forms* ejecutando los ejemplos descritos. Al finalizar el capítulo realizad los ejercicios propuestos. Igualmente leed el capítulo *Working with Templates(1.7)*(lo encontrareis en la version 1.7 del libro). Este capítulo asume una aplicación *rango* más compleja de la que teneis desarollada en la cual se ha creado un formulario para hacer "log-in/out". En el apéndice A hemos reescrito parte del código para adecuarlo a vuestra aplicación. En este capítulo sólo se solicita que implementéis los ejercicios:

- Update all other existing templates within Rango's...
- Change all the references to rango urls to use the url template tag.

7. Trabajo a Entregar Durante la Tercera Semana de la práctica

Ejercicio 3: Introducción a Django-II

1. Subid vuestro proyecto a `bitbucket` usando `git`.
2. Usando moodle entregad un proyecto de Django que contenga los ejercicios solicitados para los capitulos *Fun with Forms* y *Working with Templates(1.7)*. En concreto se deberá subir a moodle el fichero obtenido ejecutando el comando `git archive --format zip --output ../assign2_third_week.zip master` desde el directorio del projecto (`tango_with_django_project`)
3. Aseguraros de que todos los tests desarrollados son satisfechos por vuestro código
4. No hace falta que realiceis el ejercicio titulado "Undertake part XXXX of the official Django tutorial...".

8. Cuarta semana: Usando Heroku

En esta semana vamos a ver aprender a desplegar aplicaciones Web realizada en Django en Heroku. Heroku es una "plataforma como servicio" de computación en 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. 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. 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 que 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-2.7.13`.

8.3. 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 a

```
$ cat requirements.txt
Django==1.9
dj-static==0.0.6
dj-database_url==0.3.0
psycopg2==2.6.1
Pillow==2.8.2
static3==0.7.0
gunicorn==19.6.0
```

pudiendo varias el numero de version. Si existen más lineas con diferentes paquetes os recomendamos que las borréis. No os olvidéis de añadir la linea `gunicorn==19.6.0`.

8.4. 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()}
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.

8.5. 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 Bitbutcket creamos nuestro repositorio y guardamos nuestro cambios, en caso contrario se puede ignorar este paso:

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

$ git add .

$ git commit -m '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
```

8.6. 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']
```

Usando git "comite" esta ultima modificación (`git commit -m ''modify ALLOWED_HOST variable'' myshop/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 crees la base de datos de postgres en Heroku. Una descripción de como hacerlo se encuentra en la subsección 8.8

8.7. 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]:
```

8.8. 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, para crear el esquema inicial de la base de datos

```
heroku run python manage.py migrate
```

8.9. 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 ha 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()
```

Antes de ejecutar los test habría que definir en la terminal la variable SQLITE `export SQLITE=1` y una vez finalizada la ejecución habría que anular la definición `unset SQLITE`.

8.10. 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 el 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 pagina de la presentación titulada Arquitectura de Django (accesible en moodle). Añadid este fichero al repositorio del proyecto.

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. Debajo del link usado para subir el proyecto hay otro llamado `Heroku URLS práctica_2`, conectaros al mismo y escribid la dirección de Heroku donde esté desplegada vuestra aplicación. En Heroku la base de datos debe estar poblada con el contenido del fichero populate.py

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.

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:

- Que la aplicación rango descrita en el libro de referencia este desplegada en Heroku (y el URL este accesible en el URL subido a moodle usando el link denominado Heruko URL practica_2). En Heroku la base de datos debe estar poblada con el contenido del fichero populate.py
- Que la memoria y el código necesario para ejecutar la aplicación se haya entregado en moodle y sea posible ejecutarlos localmente en los laboratorios de informática.
- Que el código entregado satisfaga todos los tests.

A. Working with Templates

Code for section "10.2.1. Abstracting Further"

```
 <!DOCTYPE html>

<html>
    <head>
        <title>Rango - {% block title %}How to Tango with Django!{% endblock %}</titl
    </head>

    <body>
        <div>
            {% block body_block %}{% endblock %}
        </div>

        <hr />

        <div>
            <ul>
                <li><a href="/rango/add_category/">Add a New Category</a></li>
                <li><a href="/rango/about/">About</a></li>
            </ul>
        </div>
    </body>
</html>
```

Code for section "10.3. Template Inheritance"

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

{% load staticfiles %}

{% block title %}{{ category.name }}{% endblock %}

{% block body_block %}
    <h1>{{ category.name }}</h1>
    {% if category %}
        {% if pages %}
```

```
            <ul>
                {% for page in pages %}
                <li><a href="{{ page.url }}">{{ page.title }}</a></li>
                {% endfor %}
            </ul>
        {% else %}
                <strong>No pages currently in category.</strong>
        {% endif %}

                <a href="/rango/add_page/{{category.slug}}/">Add a Page</a>
    {% else %}
        The specified category {{ category.name }} does not exist!
    {% endif %}

{% endblock %}
```

Code for section “10.4. Referring to URLs in Templates” (tercer listado)

```
 <div>
    <ul>
        <li><a href="{% url 'add_category' %}">Add a New Category</a></li>
        <li><a href="{% url 'about' %}">About</a></li>
    </ul>
</div>
```

Code for section “10.4. Referring to URLs in Templates” (cuarto listado)

```
{% for category in categories %}
    <li><a href="{% url 'show_category'  category.slug %}">{{ category.name }}</a></l
{% endfor %}
```