Universidad Autónoma de Madrid

Computer Science Department

Computer Systems Project
Assignment - 2

Roberto Marabini Ruiz

Contents

1 Introduction

The main goal of this assignment is to understand how Django and Heroku work. We will use the on line tutorial *Tango with Django-version* (`http://www.tangowithdjango.com/`).

We will also use git for version control. Probably, the easiest way to start using git is by creating a repository on bitbucket and then clone it (`git clone...`). Finally, create a project with *PyCharm* (the recomended IDE) on the cloned repository. In the main menu select File → Open, in the dialog box select the directory containing the repository. The reference book has a chapter entitled *A Crash Course Git* which describes how to use of git .

2 First Week

Read the chapters *Django Basics* and *Templates and Static Media* from the *Tango with Django* tutorial, execute the examples provided. At the end of each chapter solve all the proposed exercises. Upload the code to a NEW repository in Bitbucket do NOT reuse the repository created in the first assignment.

Note: In one of the exercises you are asked to implement a web site containing an image. For an unknown reason there is a tendency to use very large high resolution images. These images are not the most suitable for websites.

3 Summary of the First Week Goals

1. Upload your project to `bitbucket` using `git`. Create an appropriate `.gitignore` file to avoid uploading files with extension `pyc`
2. The Django project should satisfy the exercises proposed in the chapters *Django Basics* and *Templates and Static Media*.
3. In these chapter there is a reference to some extra exercises proposed in the "official Django tutorial". You may ignore this reference.

4 Second Week

Read the chapters *Models and Databases* and *Models, Templates and Views* from *Tango with Django* tutorial, execute the examples provided. At the end of each chapter solve all the proposed exercises and upload the code to Bitbucket.

NOTE-1: At some point, a superuser account needs to be created with the command `python manage.py createsuperuser`. Set `alumnodb` as username and as password. Use `psi` as name for the project database.

NOTE-2: As database you must use `postgres` instead of `sqlite3` which is the default option. In the subsection *Telling Django about your database* the variable `DATABASES` is described. You should modify `settings.py` so it looks like:

```
#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')
```

It is assumed that the environment variable `DATABASE_URL` has been properly set.

4.1 A Note about Data Base Persistency

The databases created using postgres do not disappear when the computer is switched off. Therefore, it is very likely that you will find a database named *psi*, created by another group, with a structure/content different from the one you need. In order to avoid conflict, we suggest that you delete and recreate the database *psi* at the beginning of each class. The command `dropdb -U alumnodb -h localhost psi` will delete the database and, `createdb -U alumnodb -h localhost psi` will create it again.

4.2 Tests

In moodle you may find the file `tests.py` that contains a collection of tests. In order to execute them download the test file into the directory `rango` and type:

```
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
```

If the code does not pass some tests fix your code do NOT modify the test.

5 Deliverables to be Handed in after the Second Week

1. Push your project to the `Bitbucket` repository using `git`.
2. Upload to moodle a Django project with the exercised proposed at chapters *Models and Databases* and *Models, Templates and Views*. Specifically, you should upload to moodle the zip file created by running the command `git archive --format zip --output ../assign2_second_week.zip master` in the project directory (`tango_with_django_project`). Remember that git will only backup files that have been "added" -git add- and registered -git commit. Do not forget to include the .gitignore file.
3. Double check that all tests defined in *tests.py* are satisfied.

6 Third Week

Read the chapter entitled *Forms* and execute the examples provided. After that read the chapter entitled *Working with Templates(1.7)* (chapter *Working with Templates* is not available in the last version of the book, use version 1.7). This chapter assumes an application *rango* more elaborated than the one you have. In appendix A we have rewritten part of the chapter code so it fits with your application. For this chapter you only need to implement the exercises:

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

7 Summary of the Third Week Goals

1. Push your project to the `Bitbucket` repository using `git`.
2. Upload to moodle a Django project with the exercises proposed at chapter *Working with Templates(1.7)*. Specifically, you should upload to moodle the zip file created by running the command `git archive --format zip --output ../assign2_third_week.zip master` in the project directory (`tango_with_django_project`). Remember that git will only backup files that have been "added" -git add- and registered -git commit. Do not forget to include the .gitignore file.
3. Double check that all tests defined in *tests.py* are satisfied.
4. You do NOT need to: "Undertake part XXXX of official Django tutorial".

8 Fourth Week

This week we are going to learn how to deploy, in Heroku, Web applications created using Django. Heroku is a PaaS (platform as service), that is, a cloud computing services that provides a platform allowing customers to develop, run, and manage applications.

Note: The following documentation is based on the post: `https://amatellanes.wordpress.com/2014/02/25/django-heroku-desplegando-una-aplicacion-django-en-heroku/`

8.1 Register

To start with you need to create an account in Heroku. You may sign in at URL `https://signup.heroku.com/identity`.

8.2 Creating Procfile and runtime.txt files

We will need to create a file named `Procfile`. This file, located in the project root directory contains the commands needed to execute our application. In our case we need to start a web server (gunicorn) and execute the application. Create the file `Procfile` and type the following line:

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

By default Heroku uses python 3. Since we are working with python 2.7 we need to tell Heroku that we are not going to use the default python. Heroku will use whatever python version is specified in a file called `runtime.txt` placed in the root directory. Just create `runtime.txt` containing a single line with the following test: `python-2.7.14`.

8.3 Define Dependencies using Pip

Heroku will assume that we want to execute a Python application if there is a `requirements.txt` file in the project root directory. This file should contain all the extra python modules needed by the project. The file can be created typing the command:

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

and removing the modules that we do not need. The final result should be similar to:

```
$ 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
```

The actual version may differ but delete any extra line. Do not forget to add the line `gunicorn==19.6.0`.

8.4 Configure the application so it can be deployed in Heroku

Heroku will create an appropriate `DATABASE_URL` variable for you. Double check that your `settings.py` looks like:

```
# Parse database configuration from $DATABASE_URL
import dj_database_url
DATABASES = {}
DATABASES['default']= dj_database_url.config(default='postgres://alumnodb:alumnodb@localhost:5432/ps
STATIC_ROOT = 'staticfiles'
```

Finally, modify the file `wsgi.py` adding the following lines:

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

application = Cling(get_wsgi_application())
```

8.5 Create a git repository

Code is uploaded to Heroku using git. In the following we will summarize how to create a git repository. Very likely you already have your git repository and therefore youy may ignore this subsection. IMPORTANT: the `.git` directory should be in the same directory that the application folder.

The first step is to create a file called `.gitignore` containing those files/directories that we do not want to upload. Our `.gitignore` file should contain at least the following lines:

```
*.pyc
staticfiles
uploads
```

In case you are not working in a git repository execute the following commands

```
$ 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 Deploying

The next step is to upload and deploy our code to Heroku

```
$ 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
```

This command has created a remote repository in Heroku. Before we can deploy the code in Heroku we need to modify `settings.py` adding to the variable `ALLOWED_HOSTS` the name of the host where we are going to execute the code. This name is returned by the command `heroku create`). In the above example the value is `gentle-gorge-9766.herokuapp.com`

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

Using git commit this modification (`git commit -m ''modify ALLOWED_HOST variable'' myshop/settings.py`) and deploy the code in Heroku using the command

```
$ 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 static pystache
       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
```

Now, we can start our application in Heroku. We may skip this step since the application will be started by Heroku as soon as a browser tries to connect to it.

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

We are ready to test that everything went OK by connecting to the server:

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

8.7 Accessing Log Files

Heroku allows you to check the log files:

```
$ 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 hellodjango.wsgi‘
2014-02-23T19:38:29+00:00 app[web.1]: Validating models...
2014-02-23T19:38:29+00:00 app[web.1]:
```

8.8 Accessing Heroku's shell

Using the `run` command we can execute commands in Heroku. For example, we may execute Django's shell:

```
$ heroku run bash
```

or create the database

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

8.9 Running tests in Heroku

Django creates an auxiliary database when executing a test. Unfortunately, non-paid Heroku users cannot create postgres databases.

A possible way around this limitation is to use sqlite3 instead of postgres. In general we want to use postgres in our application because Heroku deletes the virtual home directory of any application that has been inactive by 30 minutes. Therefore, no data will be persisted from one day to the next one if stored in a local sqlite datafile. In summary, we must use postgres for the normal execution of the application and sqlite3 for testing. You can achive this goal modifiying your `settings.py` file with the following code.

```
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:alumnodb@localhost:543
```

Before executing a test you need to define the enviromental variable SQLITE `export SQLITE=1` and you must cancel the definition once the test is finished `unset SQLITE`.

8.10 Django static files on heroku

By design, Django on Heroku does not load static files (ie css, js, or images) from the project folder but from a new directory created usando de command `python manage.py collectstatic`. Djhango needs to know where to place this new directory. Therefore, the following two varibles must be defined in `settings.py` (just doble check that they are defined otherwise define them).

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

and add to the file `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 Deliverables to be Handed in after the Fourth Week

1. Write a short report ($\leq$ 2 pages) calle report.pdf. In this report summarize Django architecture. In particular comment the ppt presentation entitled *Arquitectura de Django* (accessible in moodle). Add this file to your repository.
2. Upload to moodle a Django project with the application (`rango`) developed in this assignment. Specifically, you should upload to moodle the zip file created by running the command `git archive --format zip --output ../assign2_fourth_week.zip master` in the project directory (`tango_with_django_project`). Remember that git will only backup files that have been "added" -git add- and registered -git commit. Do not forget to include the .gitignore file. The postgres database in django should be populated with the populate.py script.
3. After the link used to upload the project there is another called `Heroku URLS práctica_2`, connect to it and write down the Heroku's address in which your application was been deployed.
4. Double check that all tests defined in *tests.py* are satisfied. Do NOT modify the test file.

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

9.1 Interesting Links

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

10 Grading Criteria

This assignment will be grade as PASS or NOT PASS. In order to pass you must satisfy the following criteria:

- You have created and deployed in Heroku a functional rango application that satisfies all the requirements described in the reference book. The application is accessible at the URL registered in moodle (link `Heruko URL practica_2`). The heroku database should be initialized with the populate.py script.
- The report and all the files needed to execute the application has been uploaded to moodle. It is possible to execute the project locally using the uploaded code.
- All tests are satisfied.

Note: The code used to grade the project will be the one uploaded to Moodle. Under no circumstance we will use the one push into Bitbucket or Heroku.

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 %}</title>
    </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></li>
{% endfor %}
```