

FILMSCRAPPING - DOCUMENTACIÓN

SISTEMAS DISTRIBUIDOS

Curso 2015/2016

Ernesto Wulff Olea

Juan José García Beza

Antonio Ruíz Rondán

<http://filmscrapping.herokuapp.com/>



ÍNDICE

- [1 - INTRODUCCIÓN E IDEA](#)
- [2 - EXPLICACIÓN](#)
- [3 - CONTROL DE VERSIONES](#)
- [4 - ESTRUCTURA](#)
- [5 - DESPLIEGUE](#)
- [6 - GRÁFICAS](#)
- [7 - COMPONENTES](#)
- [8 - PLANIFICACIÓN Y REPARTO](#)
- [9 - BIBLIOGRAFÍA](#)

1 - INTRODUCCIÓN E IDEA

[FilmScrapping](#) es una aplicación web desarrollada en la versión de python 2.7.6 sobre el framework de desarrollo web Django en su versión 1.9.6. Con este proyecto hemos querido hacer una introducción al mundo del desarrollo en Django, motivados por el uso previo de Flask. También hemos ahondado en el concepto de *Scrapping* dando uso a la librería BeautifulSoup.

La idea surge debido problema actual y real con que los usuarios se encuentran al querer contrastar puntuaciones de películas en distintas webs de opinión. Actualmente han de entrar en las distintas webs de cine para conocer las puntuaciones de una determinada película teniendo que abrir y cerrar varias páginas. Nuestra aplicación resuelve este conflicto de forma que para la búsqueda de una película se mostrarán las puntuaciones obtenidas en las webs anteriormente mencionadas en un sólo espacio.

2 - EXPLICACIÓN

Podríamos dividir el proyecto en dos partes: realización de la aplicación Django y desarrollo de los módulos de scrapping.

Comenzando por la segunda, hemos hecho uso de la librería BeautifulSoup para la recuperación del texto que nos interesa (puntuación de las películas buscadas por el usuario y su nombre) de algunas de las más influyentes web de *film rating*: imdb, filmaffinity y fotogramas.

Una vez se obtienen estos datos, la aplicación los indexa en una base de datos Postgres y los muestra al usuario. El motivo de almacenar una copia de los datos en “caché” es que la próxima vez que un usuario buscara la misma película, obtendría los datos mucho más rápido ya que la aplicación está diseñada para que, en caso de que encuentre coincidencias con la búsqueda del usuario en la base de datos “local”, no busque en el exterior, ahorrando así una enorme cantidad de tiempo a ambos, servidor y usuario.

Pero claro, las puntuaciones de las películas pueden fluctuar de un día para otro. Por ello, se añade una nueva posibilidad: que aun teniendo la base de datos los datos que el usuario precisa, estos hayan caducado. En este caso se descarta la búsqueda interna y se procedería a realizar una nueva búsqueda externa. Actualmente el tiempo de refresco está establecido en un día.

```
def refrescar(listado):
    for peli in listado:
        if (datetime.utcnow().replace(tzinfo=utc) - peli.updated_at).total_seconds() > 60 * 60 *
24:
            return True
```

Detalle de la función que determina si hay que refrescar o no la información en la BD

¿Y si el usuario no se fía de nuestros datos? En tal caso puede seleccionar con un clic la opción “Forzar nueva búsqueda”, lo cual ocasionará que independientemente de si tenemos o no los datos en nuestra BD, estos sean reemplazados por nuevos datos tomados del exterior.



```
forzar = request.POST.get('forzar') # Objeto multivaluado. Devolverá True o False
peli = request.POST['nombre_peli']
if peli:
    coincidencias, nota, datos = busqueda_interna(peli)
    if coincidencias and not forzar:
```

Detalle de la función que determina si hay que forzar o no la búsqueda de datos en el exterior

Como extras, hemos añadido también concurrencia al programa, haciendo que las búsquedas de datos en nuestras tres fuentes vayan de forma paralela. [También tenemos una sección dedicada al tema de gráficas.](#)

3 - CONTROL DE VERSIONES

Hemos empleado dos sistemas de control de versiones, Heroku y Github. El primero de ellos es necesario para el despliegue de la aplicación ([ver punto de despliegue](#)). En el segundo se puede ver el log del historial de cambios completo:

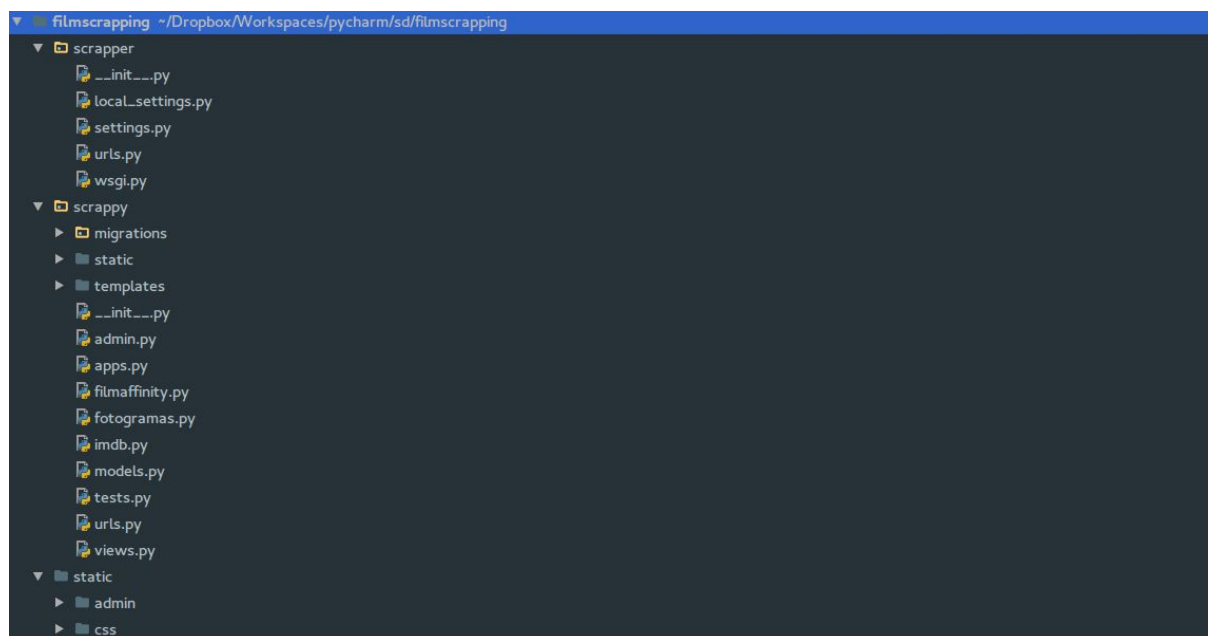
<https://github.com/Erwol/filmscrapping/commits/master>

Muestra del primer log en GitHub:

<https://github.com/Erwol/filmscrapping/commit/07134f9261799f5aca5ed8292708650e7d634e3c>

4 - ESTRUCTURA

La estructura empleada es la de un proyecto Django.



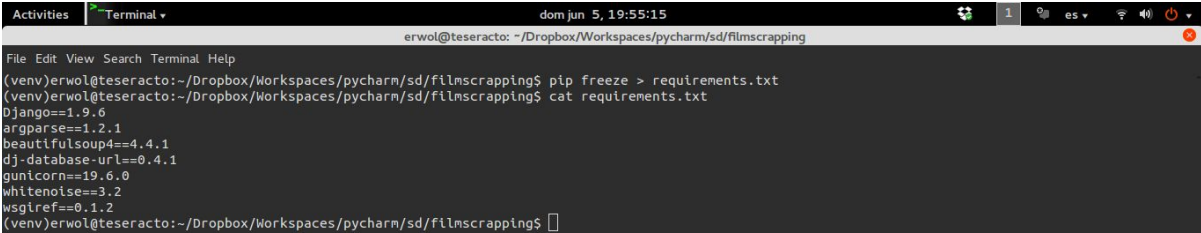
5 - DESPLIEGUE

Para desplegar la aplicación escogimos el servicio de virtualización *Heroku*. Su plan gratuito nos ofrece 450 horas mensuales de *Dyno Hours* más otras 550 horas, también mensuales, si enlazamos nuestra tarjeta de crédito con el servicio. Estas horas, que no son más que tiempo en *uptime*, las podemos distribuir entre tantas aplicaciones desplegadas como queramos, teniendo en cuenta también la limitación de la máquina virtual que *Heroku* nos presta y que posee unos generosos 512mb de ram y una CPU compartida entre un grupo de usuarios gratuitos del servicio.

El proceso de despliegue requiere de varios documentos que *Heroku* necesita para procesar correctamente nuestra aplicación Django. Éstos son los siguientes:

1.- Archivo `requirements.txt`. Servirá para que Heroku sepa los paquetes que necesitamos para ejecutar nuestra aplicación. Para crearlo, teniendo nuestro entorno virtual activado hacemos:

```
pip install dj-database-url gunicorn whitenoise
pip freeze > requirements.txt
```



```
erwol@teseracto: ~/Dropbox/Workspaces/pycharm/sd/filmscrapping
File Edit View Search Terminal Help
(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$ pip freeze > requirements.txt
(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$ cat requirements.txt
Django==1.9.6
argparse==1.2.1
beautifulsoup4==4.4.1
dj-database-url==0.4.1
gunicorn==19.6.0
whitenoise==3.2
wsgiref==0.1.2
(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$
```

Añadimos unos cuantos requerimientos más a la lista de forma manual:

```
dj-database-url==0.4.1
dj-static==0.0.6
psycopg2==2.5.4
```

2.- Procfile, necesario para indicar a Heroku el tipo de aplicación que estamos desplegando, en este caso, una aplicación web que será gestionada por gunicorn.

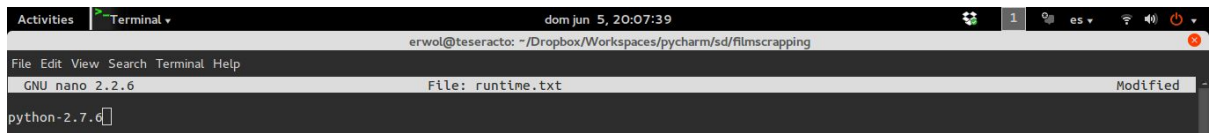
```
touch Procfile
nano Procfile
```

E introducimos:

```
web: gunicorn scrapper.wsgi
```

3.- runtime.txt, archivo en el cuál indicaremos la versión de python con la que estamos trabajando.

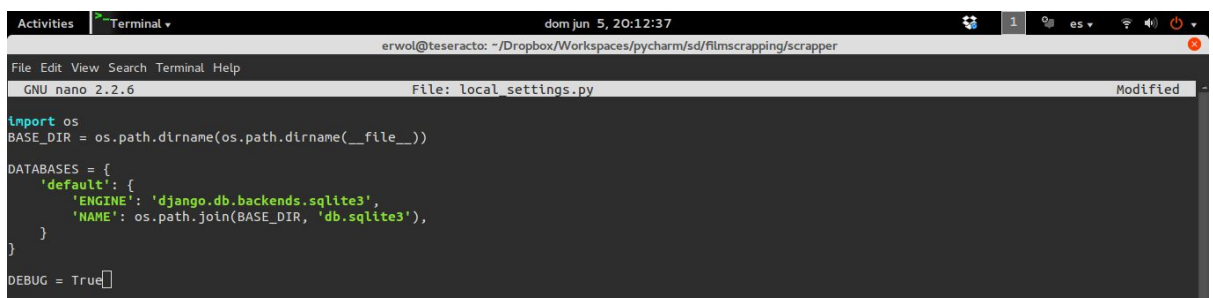
```
touch runtime.txt
nano runtime.txt
```



```
Activities Terminal
erwol@teseracto: ~/Dropbox/Workspaces/pycharm/sd/filmscrapping
File Edit View Search Terminal Help
GNU nano 2.2.6 File: runtime.txt Modified
python-2.7.6
```

4.- local_settings.py. En nuestro proceso de desarrollo hemos trabajado con una base de datos SQLite, y Heroku necesita que le demos una configuración válida de Postgres. Por tanto, crearemos este fichero que sólo nos valdrá para nuestro entorno de desarrollo.

```
cd scrapper
touch local_setting.py
```



```
Activities Terminal
erwol@teseracto: ~/Dropbox/Workspaces/pycharm/sd/filmscrapping/scrapper
File Edit View Search Terminal Help
GNU nano 2.2.6 File: local_settings.py Modified

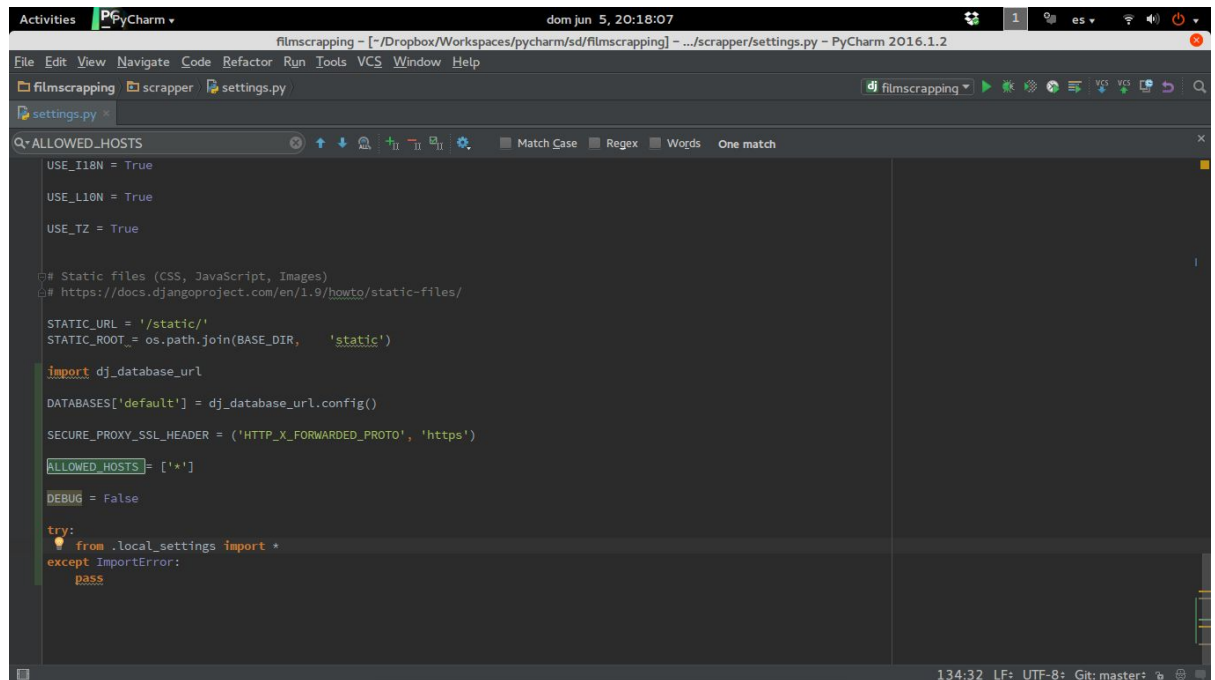
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

DEBUG = True
```

5.- settings.py. Editaremos el archivo principal de configuración de nuestra app para que, si estamos en nuestro entorno de desarrollo tome las opciones de local_settings.py y si estamos en Heroku, tome la configuración que él precise.

Importante desactivar el debug en producción.



6.- wsgi.py. Necesario para que Heroku ejecute correctamente nuestras rutinas.

En versiones modernas de gunicorn el archivo se crea junto a la instalación.

```
cd scrapper
touch wsgi.py
nano wsgi.py
```

E introducimos:

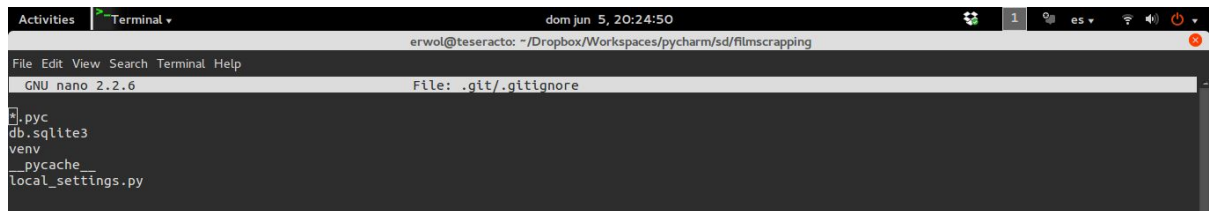
```
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE",
"scraper.settings")

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

application = Cling(get_wsgi_application())
```

Modificación de nuestro .gitignore.

```
nano .git/.gitignore
```

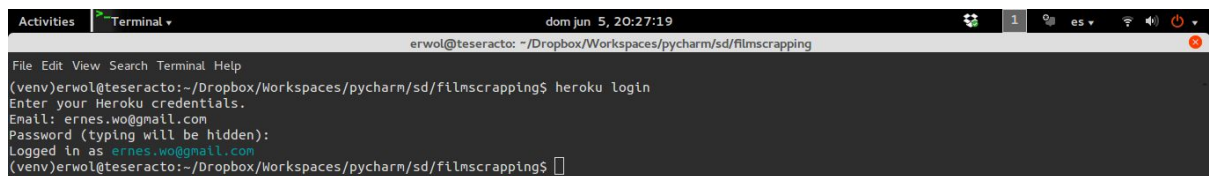


```
erwol@teseracto: ~/Dropbox/Workspaces/pycharm/sd/filmscrapping
File Edit View Search Terminal Help
GNU nano 2.2.6 File: .git/.gitignore
.pyc
db.sqlite3
.env
__pycache__
local_settings.py
```

Ahora necesitamos el Heroku toolbet: <https://toolbelt.heroku.com/>

Una vez lo instalemos, introducimos los siguientes comandos:

heroku login



```
erwol@teseracto: ~/Dropbox/Workspaces/pycharm/sd/filmscrapping
File Edit View Search Terminal Help
(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$ heroku login
Enter your Heroku credentials.
Email: ernes.wogmail.com
Password (typing will be hidden):
Logged in as ernes.wogmail.com
(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$
```

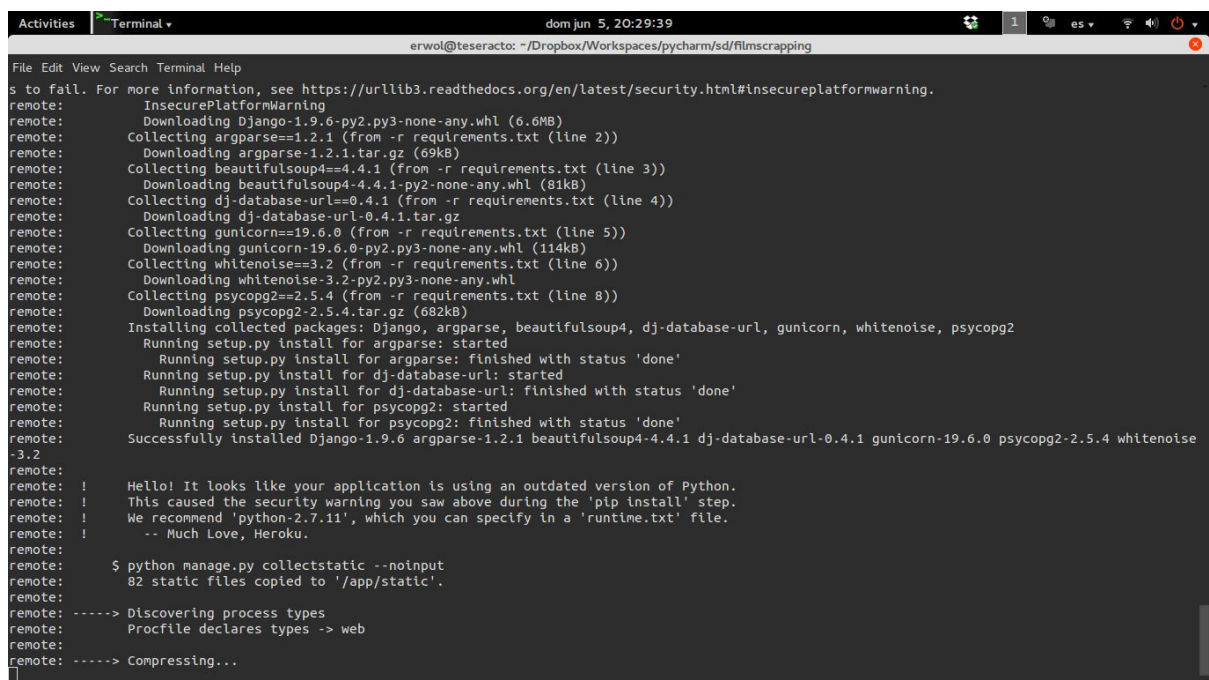
git add --all

git commit -m "Desplegando en Heroku"

filmscrapping será el nombre de la aplicación y el subdominio que Heroku nos dará para acceder a la misma.

heroku create filmscrapping

git push heroku master



```
erwol@teseracto: ~/Dropbox/Workspaces/pycharm/sd/filmscrapping
File Edit View Search Terminal Help
s to fail. For more information, see https://urllib3.readthedocs.org/en/latest/security.html#insecureplatformwarning.
remote: InsecurePlatformWarning
remote: Downloading Django-1.9.6-py2.py3-none-any.whl (6.6MB)
remote: Collecting argparse==1.2.1 (from -r requirements.txt (line 2))
remote: Downloading argparse-1.2.1.tar.gz (69kB)
remote: Collecting BeautifulSoup4==4.4.1 (from -r requirements.txt (line 3))
remote: Downloading BeautifulSoup4-4.4.1-py2.py3-none-any.whl (81kB)
remote: Collecting dj-database-url==0.4.1 (from -r requirements.txt (line 4))
remote: Downloading dj-database-url-0.4.1.tar.gz
remote: Collecting gunicorn==19.6.0 (from -r requirements.txt (line 5))
remote: Downloading gunicorn-19.6.0-py2.py3-none-any.whl (114kB)
remote: Collecting whitenoise==3.2 (from -r requirements.txt (line 6))
remote: Downloading whitenoise-3.2-py2.py3-none-any.whl
remote: Collecting psycogp2==2.5.4 (from -r requirements.txt (line 8))
remote: Downloading psycogp2-2.5.4.tar.gz (682kB)
remote: Installing collected packages: Django, argparse, BeautifulSoup4, dj-database-url, gunicorn, whitenoise, psycogp2
remote: Running setup.py install for argparse: started
remote: Running setup.py install for argparse: finished with status 'done'
remote: Running setup.py install for dj-database-url: started
remote: Running setup.py install for dj-database-url: finished with status 'done'
remote: Running setup.py install for psycogp2: started
remote: Running setup.py install for psycogp2: finished with status 'done'
remote: Successfully installed Django-1.9.6 argparse-1.2.1 BeautifulSoup4-4.4.1 dj-database-url-0.4.1 gunicorn-19.6.0 psycogp2-2.5.4 whitenoise-3.2
remote:
remote: ! Hello! It looks like your application is using an outdated version of Python.
remote: ! This caused the security warning you saw above during the 'pip install' step.
remote: ! We recommend 'python-2.7.11', which you can specify in a 'runtime.txt' file.
remote: ! -- Much Love, Heroku.
remote:
remote: $ python manage.py collectstatic --noinput
remote: 82 static files copied to '/app/static'.
remote:
remote: -----> Discovering process types
remote: Procfile declares types -> web
remote:
remote: -----> Compressing...
```

Ahora necesitamos que Heroku comience a ejecutar nuestra aplicación. Para ello:

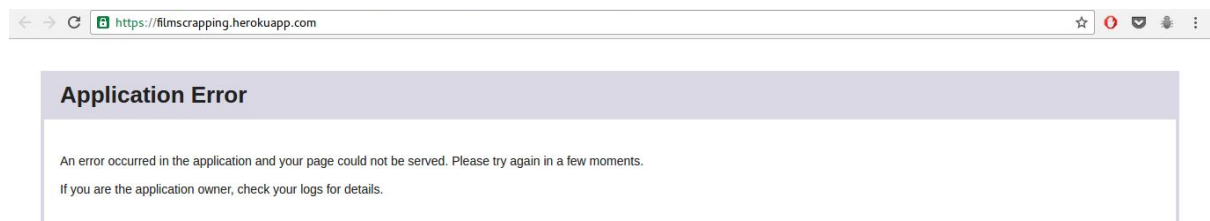
heroku ps:scale web=1


```
(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$ heroku ps:scale web=1
Scaling dynos... done, now running web at 1:Free
(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$
```

Y ya podremos entrar a la aplicación:

Este comando nos abrirá directamente una pestaña con la app cargada
heroku open

Obtendremos un error como:



Ocasionado por no haber migrado nuestra base de datos a Postgres. Para hacerlo:

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

```
^C(venv)erwol@teseracto:~/Dropbox/Workspaces/pycharm/sd/filmscrapping$ heroku run python manage.py makemigrations
Running python manage.py makemigrations on filmscrapping... up, run.6770
Migrations for 'scrappy':
  0008_auto_20160605_1841.py:
    - Alter field last_score on film
    - Alter field score on score
```

Y, de paso, nos creamos un superusuario en Django:

```
heroku run python manage.py createsuperuser
```

El resultado podemos verlo en: <http://filmscrapping.herokuapp.com/>

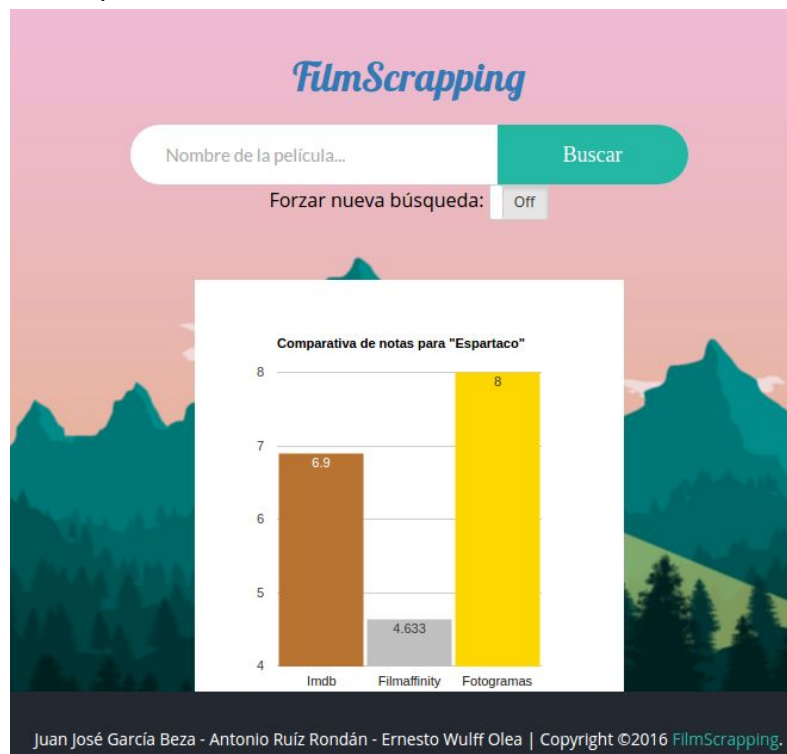
6 - GRÁFICAS

Hemos optado por generar gráficas de forma dinámica. Es decir, por cada búsqueda que realicemos en la aplicación, el sistema nos generará una gráfica distinta.

Por ejemplo, si buscamos la película Espartaco:



Y ahora clicamos en “Generar gráfica”, obtendremos una gráfica con la media que cada servicio otorga a cada película:



Si cualquiera de estos valores cambiara en nuestra base de datos o en el servidor original, la puntuación se modificaría.

7 - COMPONENTES

Se listan a continuación, divididos en dos grupos, los componentes a los que hemos dado uso en el desarrollo y despliegue de la aplicación.

Componentes Python

- python-2.7.6
- Django==1.9.6
- argparse==1.2.1
- beautifulsoup4==4.4.1
- dj-database-url==0.4.1
- gunicorn==19.6.0
- whitenoise==3.2
- wsgiref==0.1.2
- psychopg2==2.5.4
- Django==1.9.6
- argparse==1.2.1
- beautifulsoup4==4.4.1
- dj-database-url==0.4.1
- gunicorn==19.6.0
- whitenoise==3.2
- wsgiref==0.1.2
- Django==1.9.6
- argparse==1.2.1
- beautifulsoup4==4.4.1
- dj-database-url==0.4.1
- dj-static==0.0.6
- gunicorn==19.6.0
- static3==0.7.0
- whitenoise==3.2
- wsgiref==0.1.2

Componentes externos

- Heroku
- Postgres
- Google Charts
- Pycharm

8 - PLANIFICACIÓN Y REPARTO

El reparto de las tareas ha sido el siguiente:

- Ernesto Wulff Olea. Desarrollo del servidor Django y posterior despliegue en Heroku.
- Juan José García Beza. Control de procesos concurrentes con la librería threading, documentación, frontend y pruebas.
- Antonio Ruíz Rondán. Desarrollo de los módulos de *scrapping* imbd.py, filmaffinity.py y fotogramas.py con la librería beautifulsoup4 y testeo de la aplicación.

Un esquema aproximado de la planificación del trabajo que hemos querido llevar puede verse en la siguiente gráfica:

Actividad	Mayo				Junio
	Semana 1	Semana 2	Semana 3	Semana 4	Semana 1
Idea					
Planificación					
Desarrollo de módulos en Python					
Desarrollo de servidor en Django					
Implantación en Heroku					
Implementación de Hilos					
Pruebas					

9 - BIBLIOGRAFÍA

Desarrollo en Django y python

- <http://www.djangobook.com/en/2.0/index.html>
- <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Despliegue en Heroku

- <https://devcenter.heroku.com/articles/getting-started-with-python#introduction>
- <https://devcenter.heroku.com/articles/deploying-python>
- <https://djangogirls.gitbooks.io/django-girls-tutorial-extensions/content/heroku/>
- + Documento de elaboración propia, ya que ninguno de los manuales arriba citados se encuentra actualizado a Django 1.9.6.

Fuentes de los datos:

- <http://www.imdb.com>
- <http://www.filmaffinity.com>
- <http://www.fotogramas.es>