

Práctica 7 y 8

Introducción: En esta práctica se han migrado algunas de las páginas anteriores realizadas en Flask, un sistema de cuentas para acceder a determinadas páginas, conexión a base de datos MongoDB y despliegue a producción de la aplicación, todo esto, con el framework Django.

1) Páginas web modificadas

- Restaurants (Búsqueda o creación de nuevo restaurante)
- Restaurants/Search (Para obtener la información de un restaurante)
- Restaurants/Edit (Para modificar los datos de un restaurante)

2) Arquitectura de la aplicación

Esta aplicación se encuentra organizada de la siguiente forma:

- / (Raíz de la aplicación)
 - myweb
 - forms.py (código de los formularios)
 - settings.py (datos de configuración)
 - urls.py (configuración de URLs)
 - wsgi.py (configuración WSGI)
 - restaurants
 - urls.py (configuración de URLs para determinada página)
 - views.py (vista de la aplicación)
 - static (Contenido estático que utilizará la aplicación)
 - css
 - fonts
 - images
 - js
 - templates (Archivos HTML para dar formato a la web)
 - Makefile
 - manage.py

3) Funcionalidades

Vistas y formularios

Se ha ajustado el template con el HTML de la entrega anterior, realizado en Jinja, así como los formularios, que se han utilizado los propios que ofrece Django.

Los archivos HTML se encuentran en *myweb/templates*, la arquitectura de los mismos es igual que con Flask. Tenemos nuestro archivo base, que en este caso es *layout.html*, donde se encuentran aquellos elementos que se conservan para todas las páginas. En éste se cargará un bloque que se encuentra en *template.html* dependiendo de la página en la que nos encontremos.

Los formularios se han cambiado por los que Django nos aporta mediante la creación de clases. Por ejemplo para añadir un nuevo restaurante se ha creado un formulario en el archivo *forms.py* dentro de la carpeta *myweb/myweb* que es llamado desde el archivo de vistas *views.py* donde al renderizar el template le envíe la información para que lo muestre correctamente así como obtenga la información por POST.

Conexión con MongoDB

La conexión con la base de datos MongoDB no cambia mucho, básicamente como se hacía con Flask, salvo que con Django podemos decirle donde queremos que se ejecute el cliente, que puede ser a través del archivo *settings.py*.

APIs de Google Maps y Twitter

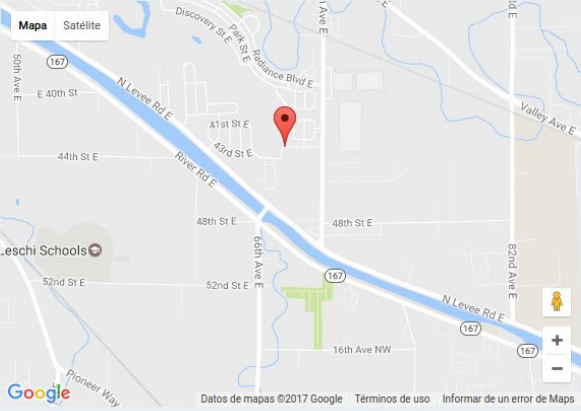
También se han usado las APIs de Google Maps y Twitter para esta práctica. Cuando vamos a guardar la información de un restaurante podemos realizar una búsqueda de la calle por Google Maps para obtener sus coordenadas:

Restaurant Alcala Restaurant found!

Name Cuisine

Borough Street

Coord Y Coord X



Si queremos buscar por ejemplo dicho restaurante en el buscador, nos aparecerán a parte de sus datos, los últimos tweets acerca de éste:

Restaurant Alcala Restaurant

Name Alcala Restaurant | Cuisine Spanish | Borough Manhattan | Street Street Avenue 43

[Modify Restaurant](#) [Back to search](#)



[Follow](#)

Aléjate de mi bronquitis del mal que no ves que quiero conectarla 🤔 — estoy bebiendo cerveza en Restaurant & Bar... fb.me/16DjlW980

2:18 AM - 27 Jan 2017

👤 🔄 ❤️



[Follow](#)

Paying customers foot the bill for the restaurant to serve dinner to homeless people, free of charge. buff.ly/2jZXig7

3:00 AM - 26 Jan 2017



Spain's 'Robin Hood Restaurant' Charges The Rich And Feed...
As the country reels from its financial crisis, a new restaurant, run by a Catholic priest, lets paying daytime customers foot the dinner bill npr.org

👤 🔄 25 ❤️ 38

Despliegue a producción

Una vez que tenemos nuestra aplicación lista para poder mostrarla al público, ya podremos realizar el despliegue a producción. Para ello podemos seguir los siguientes pasos:

En el archivo *settings.py* podremos cambiar las siguientes líneas y ponerlas como puede verse:

```
DEBUG = False  
ALLOWED_HOSTS = ['*']
```

Dejará de funcionar el servidor de desarrollo, y de servir los contenidos de */static* para hacerlo desde el servidor de producción.

Ahora hay que poner a funcionar el servidor web, donde se usa **gunicorn** y un servidor web como proxy inverso **nginx** para los archivos estáticos.

Tenemos que crear el archivo */etc/nginx/sites-available/<miproyecto>* y guardar la siguiente información:

```
server {  
    listen 8008;  
    server_name 127.0.0.1;  
  
    access_log off;  
  
    location /static/ {  
        alias /var/www/static/;  
    }  
  
    location / {  
        proxy_pass http://127.0.0.1:8000;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
    }  
}
```

Le estamos diciendo que nuestro servidor web escuche en el puerto 8002 del proxy dado con la dirección *http://127.0.0.1:8000* que es del servidor web de gunicorn. También el alias de la localización de la carpeta */static/* se ha puesto */var/www/static/* porque es donde en este proyecto se accede a los archivos estáticos cuando estamos en producción.

Creamos el enlace simbólico a */etc/nginx/sites-enabled* y reiniciamos el servicio de nginx.

Realizados estos pasos falta que todo esto se haga cada vez que se inicie la máquina, para ello usaremos **supervisor**.

Creamos el archivo `/etc/supervisor/conf.d/<proyecto>.conf` y añadimos la siguiente información:

```
[program:gunicorn]  
command=/usr/local/bin/gunicorn <miproyecto>.wsgi --bind 0.0.0.0:8000  
directory=/path/al/proyecto/<proyecto>  
user=elquesea  
autostart=true  
autorestart=true  
redirect_stderr=true
```

Actualizamos e iniciamos supervisorctl. Si entramos a `http://127.0.0.1:8008` ya tendríamos nuestro gunicorn funcionando en segundo plano.