Memoria Práctica 2 - PSI

1. Uso de Django

Django es un framework de desarrollo web de apps escrito en Python. Usa una arquitectura de diseño basado en el Modelo Vista Controlador (MVC) que separa los datos de la aplicación, la interfaz del usuario y la lógica de control en tres componentes distintos.

1.1 Modelo

La parte de los datos de la aplicación es la del *Modelo*. En nuestro caso, y siguiendo la guía sobre django de Moodle, podemos ver que de esta parte se encarga el fichero *models.py* que se encuentra en el proyecto dentro de la carpeta de *rango*.
A parte, esta sección de *Modelo* se encarga de estructurar los datos que se usan dentro de una base de datos. En este caso, la estructura de datos o base de datos que hemos utilizado es *PostgreSQL* (tal y como se especificó durante el enunciado de la práctica).

1.2 Vista

La parte de la interfaz de la aplicación es la de la *Vista.* Como se puede pensar cuando realizamos una página web, la vista se realiza sobre código *html.* Con esta parte de *Vista* se relacionan todos los templates realizados para la web app que se pueden encontrar bajo el directorio de *templates.*

1.3 Controlador

Esta parte de *Controlador* es la encargada de la lógica de control. Seguramente es la parte más delicada e importante de todo el diseño puesto que es la que conecta el Modelo de los Datos con la Vista de los mismos para hacer que el usuario final reciba una web app clara y funcional.
Para ello, cuando se recibe una petición HTTP, entra el controlador en juego mediante su fichero *urls.py* y localiza la url solicitada de la aplicación. Una vez encontrada esta url, para servirla adecuadamente, usa el fichero *views.py* que tiene una función asociada a esa página haciendo así la conexión entre vista y modelo efectiva.

2. Uso de Heroku

En este subapartado de la memoria, queremos hacer especial mención al uso de Heroku ya que es la otra gran tecnología usada en la práctica.

Heroku se podría definir como un servicio para el despliegue de aplicaciones web que funciona a través de repositorios git. La única diferencia para que se despliegue correctamente los cambios de nuestro repositorio git es que al realizar el push, hay que indicar su despliegue con Heroku:

git push heroku

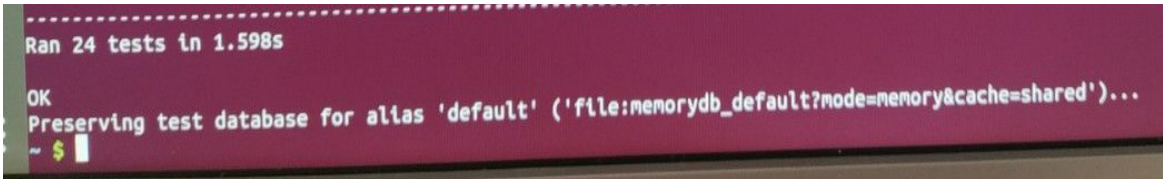
Para que todo esto funcione correctamente, hay que crear las dependencias necesarias de nuestro proyecto de git. Entre ellas la creación de un fichero *requirements.txt* (indicando la versión y las dependencias usadas), un fichero *Procfile* (con los logs) y el fichero de *runtime.txt* (para saber la versión de Python usada). No obstante, hay otras posibilidades de desplegar la aplicación con Heroku mediante la interfaz de la web pero la conexión con el servidor sería de más bajo nivel.

3. Resultados Tests

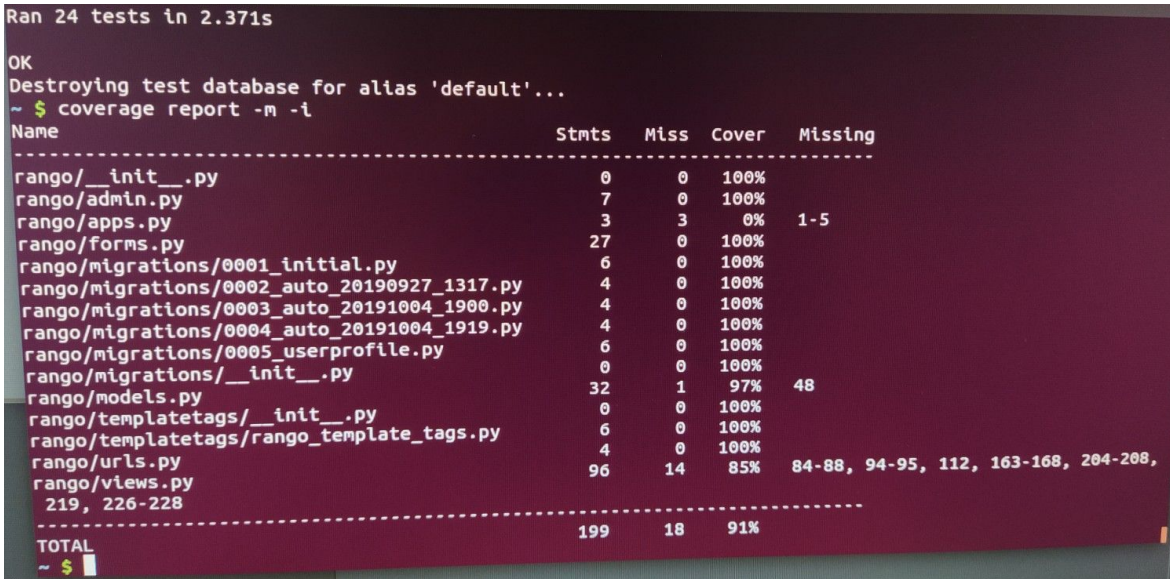
En este subapartado de la memoria, se incluirán capturas de la ejecución de los tests.

En primer lugar, podemos observar el resultado de ejecutar todos los tests de manera local:

En segundo lugar, observamos los resultados de los tests pero esta vez ejecutados dentro de su despliegue en Heroku.



Por último, en esta última captura, se observa el coverage de la aplicación que alcanza el 91% total y que lo baja el views.py puesto que no se prueba que pase por las excepciones del código.



Para finalizar solo queremos recordar que la aplicación se encuentra accesible desde la web:

https://nameless-shelf-33857.herokuapp.com/