Universidad Autónoma de Madrid

Departamento de Ingeniería Informática

Proyecto de Sistemas Informáticos (PSI)

Práctica 1

El Equipo Docente de PSI

Registro de Cambios

Versión[1]	**Fecha**	**Autor**	**Descripción**
1.0	03.01.2023	JAMI	Primera versión.
1.1	09.01.2023	AB	Traducción al inglés.
1.2	26.01.2023	RM	Revisado antes de subir a Moodle.
1.3	31.01.2023	JAMI	Cambio en el nombre de la aplicación (rango vs catalog) en coverage.
1.4	03.02.2023	AB	Resuelta inconsistencia con el nombre de la carpeta de los tests.
1.5	11.02.2023	JAMI	Incluidas las instrucciones para el uso de neo.tech como SGBD.

[1]La asignación de versiones se realizan mediante 2 números $X.Y$. Cambios en Y indican aclaraciones, descripciones más detalladas de algún punto o traducciones. Cambios en X indican modificaciones más profundas que o bien varían el material suministrado o el contenido de la práctica.

Índice

1. Introducción

Los objetivos de esta práctica son, por un lado, iniciarse en la programación de aplicaciones web a través del framework *Django* y, por otro, abordar el despliegue de una aplicación web en un entorno de producción mediante la plataforma *Render*. Para lograr estos objetivos, el estudiante deberá seguir atentamente, paso a paso, el tutorial existente en: `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django`.

Además, se deberán seguir las especificaciones complementarias que se darán a lo largo de este enunciado. De manera general para esta práctica, se tendrán en cuenta las siguientes consideraciones:

- El código se corregirá en un ordenar con sistema operativo *Ubuntu 22.04 LTS*, que es el que se encuentra disponible en los laboratorios de la EPS.
- La versión de Python a utilizar será la 3.9.
- La versión de *Django* a utilizar será la 3.2.
- Se recomienda utilizar, como entorno de desarrollo, el IDE Visual Studio Code.
- Se utilizará un entorno virtual (virtualenv) para trabajar y crear las dependencias específicas para el desarrollo de esta práctica.
- Se utilizará `git`, creando para esta práctica un nuevo repositorio privado a compartir sólo por la pareja de estudiantes que trabajan en el mismo equipo. El directorio (oculto) `.git` debe ser incluido obligatoriamente como parte de la entrega de esta práctica. Adicionalmente, se debe crear un fichero `.gitignore` para evitar subir ficheros irrelevantes (temporales, etc.). El equipo de estudiantes debe comentar con su profesor de laboratorio la conveniencia de compartir también con él el repositorio `git` creado para esta práctica. Se recomienda el uso de GitHub.
- El código Python implementado debe satisfacer la guía de estilos *PEP8*, lo cual deberá verificarse mediante la utilidad `flake8` (que se puede instalar mediante el comando `pip3 install flake8`).
- Aunque el propio tutorial irá indicando en qué momento instalar cada paquete con `pip3`, en *Moodle* se proporciona la versión final del fichero `requirements.txt` donde se encuentran todas las dependencias necesarias para realizar esta práctica.

- En *Moodle* se encuentran los test que se deberán pasar de manera obligatoria. La forma general de proceder será pasar los test correspondientes a cada semana. Esto puede hacerse al final de la implementación exigida, pero los test pueden utilizarse también como guía para la implementación de la práctica, ya que irán indicando ciertos requisitos concretos que habrá que satisfacer. Cualquier error que aparezca en los test debe corregirse en el código propio. Por tanto, los test proporcionados en *Moodle* no se pueden modificar, ya que además serán utilizados para evaluar la práctica.
- No deben modificarse los nombres de los objetos a crear a lo largo del tutorial (clases, campos, base de datos, etc.), ni tampoco los proporcionados en este enunciado.
- La forma general de proceder con esta práctica será ir acometiendo el trabajo propuesto para cada semana, lo que implica estudiar e implementar lo exigido en los capítulos del tutorial y realizar además, de manera obligatoria, los ejercicios propuestos al final de cada parte (*challenges*). Al final de cada capítulo se proporcionan también una serie de enlaces que el estudiante puede utilizar para consolidar o ampliar conocimientos.
- Esta práctica pretende fomentar el estudio paso a paso de los conocimientos básicos y necesarios para el desarrollo y despliegue de aplicaciones web. Por ello, es conveniente que, aunque se trabaje en equipo, cada estudiante interiorice de manera individual los conocimientos necesarios para avanzar y superar las siguientes prácticas y exámenes obligatorios de la asignatura.

2. Descripción del trabajo a realizar durante la primera semana

A modo introductorio, se deberán leer y poner en práctica los siguientes capítulos del tutorial comentado anteriormente, principalmente si se desea además una instalación personaliza del entorno en equipos propios. En cualquier caso, el sistema operativo de referencia será el comentado con anterioridad (*Ubuntu 22.04 LTS*). Los capítulos introductorios a abordar son los siguientes (en este caso, no es necesario entregar lo implementado en estos capítulos):

- *Django introduction.*
- *Setting up a Django development environment.*

Para comenzar a trabajar con la aplicación de la biblioteca (*locallibrary*), se deberá crear un nuevo repositorio *Git* para esta práctica. Además, se creará y usará (activará) un entorno virtual para la versión de Python exigida (3.9):

```
virtualenv p1_env --python=python3.9
source p1_env/bin/activate
```

Con el entorno virtual activado, se debe instalar la versión exigida de Django:

```
pip3 install django~=3.2
```

Una vez que se tenga un conocimiento sólido de lo que es Django, se debe leer e implementar, paso a paso, lo descrito en los siguientes capítulos del tutorial para comenzar a implementar la aplicación web de la biblioteca (proyecto `locallibrary`, aplicación `catalog`) que se deberá entregar según lo indicado en la planificación de esta práctica, realizando los ejercicios que se indican al final de cada capítulo:

- *Django Tutorial: The Local Library website.*
- *Django Tutorial Part 2: Creating a skeleton website.*
- *Django Tutorial Part 3: Using models.*

Se recuerda que el código creado debe seguir las directivas *PEP8*. Esto es, si se comprueba el código con el analizador `flake8`, la salida por pantalla no debe contener ni errores ni advertencias. Se pueden ignorar las advertencias relacionas con el código suministrado, o con las líneas de código producidas automáticamente por Django.

Una vez creados los modelos indicados en la Parte 3, se debe poblar la base de datos utilizando el fichero `populate_catalog.py` que puede descargarse de *Moodle*. Para ello, el fichero se deberá situar en la raíz del proyecto, y se deberá ejecutar de la forma:

```
python3 populate_catalog.py
```

2.1. Test

En la página *Moodle* de la asignatura se encuentran los ficheros `test_xxxx_week.py`, los cuales forman una colección de test a pasar durante la implementación de la práctica. Estos ficheros deben situarse en una carpeta llamada `tests` dentro de la carpeta de la aplicación (`catalog`) creada para el proyecto (`catalog/tests`).

Para esta primera semana, se deberán pasar los test correspondientes (`test_first_week.py`). Si surgen errores, se debe modificar el código propio implementado para satisfacer los test, y nunca el código de los test proporcionados. Los test de esta primera semana se ejecutarán de la siguiente forma:

```
python3 manage.py test catalog.tests.test_first_week --verbosity 2
```

2.2. Cobertura de los test

Se utilizará la aplicación `coverage` para medir la cobertura de los test, es decir, el porcentaje de código al que se accede desde los test, lo que permitirá saber cuánto código estamos realmente probando. Para ello, es conveniente tener instalada la versión 5.5 o superior de `coverage`. Al instalarse dentro del entorno virtual, es conveniente asegurarse de que la aplicación `coverage` que se ejecuta es la del entorno virtual, y no otra externa (que esté en el *path*).

Para ejecutar `coverage`, hay que proceder de la siguiente forma (más información en: `https://coverage.readthedocs.io`):

```
coverage erase
coverage run --omit="*/test*" --source=catalog manage.py test catalog.tests
coverage report -m -i
```

Es importante indicar que `coverage` ejecutará los test que haya en la carpeta anteriormente comentada, por lo que es conveniente ir añadiendo los test gradualmente para que no dé error por falta de código aún no implementado. Para hacer una prueba inicial, es necesario tener sólo en la carpeta `catalog/test` los test correspondientes a la primera semana.

Tras ejecutarlo, se obtendrá un resultado similar al mostrado en Listado 1. Esta cobertura se irá incrementando según se vaya avanzando en el tutorial y se vayan incorporando los test de las semanas sucesivas. Además, el porcentaje se incrementará cuando se hayan implementado, más adelante, los test exigidos en la Parte 10 del tutorial. Lo ideal es conseguir, al final, una cobertura cercana al 100 %.

3. Descripción del trabajo a realizar durante la segunda semana

Durante esta semana se configurará la persistencia de los datos mediante el sistema gestor de bases de datos `PostgreSQL`. Además, se continuará con el tutorial

Listado 1: Salida del comando `coverage`.

Name	Stmts	Miss	Cover	Missing
catalog/__init__.py	0	0	100%	
catalog/admin.py	1	0	100%	
catalog/apps.py	3	3	0%	1–5
catalog/migrations/__init__.py	0	0	100%	
catalog/models.py	1	0	100%	
catalog/urls.py	3	0	100%	
catalog/views.py	8	0	100%	
TOTAL	18	5	80%	

propuesto, siguiendo paso a paso lo indicado en cada capítulo y realizando los ejercicios propuestos.

Para garantizar la persistencia de los datos de la aplicación, se trabajará con `PostgreSQL`, en vez de `SQLite` que es el sistema gestor de bases de datos por defecto. Para ello se deben instalar dos paquetes (`dj-database-url` y `psycopg2-binary`), cuyas versiones vienen especificadas en el fichero `requirements.txt`. Esto se explica también en la Parte 11 del tutorial (*Django Tutorial Part 11: Deploying Django to production*), aunque en este caso la tarea se realizará durante esta semana.

Una vez instaladas las dependencias, se debe modificar la variable `DATABASES` del fichero `settings.py` de la siguiente manera:

```
#The following environment variable, called DATABASE_URL, has to be defined
#at the o.s. level: export DATABASE_URL =
#        'postgres://alumnodb:alumnodb@localhost:5432/psi'

import dj_database_url

db_from_env =
  dj_database_url.config(default='postgres://alumnodb:alumnodb@localhost:5432/psi',
    conn_max_age=500)

DATABASES['default'].update(db_from_env)
```

Se debe utilizar en todo momento `psi` como nombre de la base de datos asociada al proyecto, y `alumnodb` como usuario y contraseña correspondientes.

Por otro lado, las bases de datos creadas usando `PostgreSQL` no se borran al apagar el ordenador del laboratorio. Por ello, es posible que en el ordenador haya una base de datos ya existente llamada `psi` con una estructura diferente a la exigida. Para evitar conflictos, es recomendable que al principio de cada sesión se borre la base de datos `psi`. Para borrar la base de datos se puede usar el comando `dropdb -U alumnodb -h localhost psi`, y a continuación crearla de nuevo con `createdb -U alumnodb -h localhost psi`. Al eliminar la base de datos, es necesario volver a poblarla ejecutando el script `populate_catalog.py`.

Una vez realizadas estas tareas, se seguirá con los siguientes capítulos del tutorial, continuando con la implementación de la aplicación de la biblioteca, y realizando los ejercicios de ampliación que se proponen al final de cada capítulo:

- *Django Tutorial Part 4: Django admin site.*
- *Django Tutorial Part 5: Creating our home page.*
- *Django Tutorial Part 6: Generic list and detail views.*

Es importante comentar que en el tutorial el servidor de desarrollo de Django se arranca en el puerto 8000. Sin embargo, es posible que este puerto no esté disponible en los ordenadores de los laboratorio, por lo que puede utilizarse el puerto 8001 en su lugar ejecutando el comando:

```
python3 manage.py runserver 8001
```

Como se comenta en el tutorial, para acceder a la interfaz de administración se deberá crear un *superusuario*, lo que permitirá además acceder a la base de datos generada a partir de los modelos y manipular la información que contiene. Para ello, se utilizará el comando `python manage.py createsuperuser`. Tanto el usuario como la contraseña a utilizar serán, en ambos casos, `alumnodb`.

3.1. Test

Los test relacionados con el trabajo de esta segunda semana se encuentran en el fichero `test_second_weeky.py`. Se deben pasar los test de la primera y segunda semana. Para ejecutar los test, se procederá de la siguiente forma:

```
python3 manage.py test catalog.tests.test_first_week --verbosity 2
python3 manage.py test catalog.tests.test_second_week --verbosity 2
```

4. Descripción del trabajo a realizar durante la tercera semana

En esta semana, se estudiarán los últimos capítulos del tutorial. Se realizará además un despliegue en *Render* de la aplicación creada con *Django*, y se procederá con la entrega final de la práctica satisfaciendo todo lo exigido en el enunciado.

Se deberá leer e implementar, paso a paso, lo descrito en los siguientes capítulos del tutorial para seguir implementando la aplicación web de la biblioteca, realizando los ejercicios exigidos al final de cada parte:

- *Django Tutorial Part 7: Sessions framework.*
- *Django Tutorial Part 8: User authentication and permissions.*
- *Django Tutorial Part 9: Working with forms.*
- *Django Tutorial Part 10: Testing a Django web application.*
- *Django web application security.*
- *Django Tutorial Part 11: Deploying Django to production.*

Durante el estudio del capítulo 11 se tendrán en cuenta los aspectos generales relacionados con el despliegue de una aplicación en un entorno de producción real, si bien las instrucciones específicas del servicio a utilizar se darán en la sección *4.2. Consideraciones adicionales sobre Render* de más abajo.

4.1. Test

Se pasarán los test de la primera, segunda y tercera semana. Como siempre, ante cualquier error se debe modificar el código implementado para satisfacer los test, y nunca el código de los test proporcionados:

```
python3 manage.py test catalog.tests.test_first_week --verbosity 2
python3 manage.py test catalog.tests.test_second_week --verbosity 2
python3 manage.py test catalog.tests.test_third_week --verbosity 2
```

Adicionalmente, se deben pasar todos los test a implementar en la Parte 10 del tutorial (*Django Tutorial Part 10: Testing a Django web application*), junto con aquellos test implementados a partir del *challenge* propuesto al final de este capítulo. Todos los test deben situarse en la carpeta `/catalog/test`, y podrán ser ejecutados directamente, y en su totalidad, de la siguiente forma:

```
python3 manage.py test catalog.tests --verbosity 2
```

Es posible que, al ejecutar los test implementados en el capítulo 10, el sistema retorne el error comentando en el tutorial (*Missing staticfiles manifest entry*), lo cual puede solucionarse según las instrucciones suministradas dentro del propio capítulo. También, y dado que los datos de prueba pueden no coincidir en la base de datos con los mismos identificadores, es posible que aparezca un error adicional en aquellas instrucciones de los test que realizan una obtención explícita de los registros mediante `.objects.get(id=1)`. La forma de solucionar este error es añadir, en los test creados en dicho capítulo, donde corresponda, la creación explícita del registro con dicho `id`, por ejemplo: `Author.objects.create(id=1, first_name='Big', last_name='Bob')`.

4.2. Consideraciones adicionales sobre Render

Render es una *plataforma como servicio* de computación en la nube (PaaS) que permite desplegar la aplicación creada en un entorno de producción.

Para trabajar con Render, el primer paso es crear una cuenta individual gratuita, lo cual se puede hacer en el siguiente enlace: `https://www.render.com`. El paso de despliegue en producción se debe realizar una vez que la aplicación esté finalizada y libre de errores. Es un proceso costoso en tiempo, por lo que se recomienda finalizar, corregir todos los errores, y pasar todos los test primero en el entorno de desarrollo y preparar finalmente la aplicación, como último paso, para ser desplegada en entorno de producción en Render.

Es importante que el fichero `requirements.txt` esté en la raíz del proyecto. El fichero `requirements.txt` es el suministrado en *Moodle*, pero en cualquier caso se puede crear de forma automática a través del comando:

```
pip3 freeze > requirements.txt
```

El contenido del fichero resultante se debe parecer a lo mostrado en el Listado 2.

Listado 2: Fichero `requirements.txt`.

```
asgiref==3.6.0
coverage==6.4.3
dj-database-url==0.5.0
Django==3.2.1
gunicorn==20.0.4
Pillow==9.4.0
psycopg2-binary==2.9.5
```

```
PyJWT==2.6.0
pytz==2022.7
six==1.15.0
sqlparse==0.4.3
text-unidecode==1.2
urllib3==1.26.9
whitenoise==5.2.0
python-dotenv==0.21.0
Brotli==1.0.9
```

Para realizar el despliegue, deberás crear un (nuevo) servicio web (*New Web Service*). No es necesario crear un servicio PostgreSQL con Render ya que, aunque se podría, la base de datos creada tendrías limitaciones no convenientes; la conexión con el gestor de bases de datos se explicará más adelante. El despliegue se puede realizar automáticamente vinculando el repositorio de GitHub con el proyecto Render, identificándose convenientemente y otorgando credenciales. Es posible realizar incluso el despliegue de manera automática tras cada *commit* en el repositorio GitHub, aunque es más recomendable hacerlo manualmente (*Manual Deploy - Deploy Latest Commit*).

Es conveniente crear o modificar el fichero oculto `.gitignore`, indicando aquellos ficheros a ser ignorados en el despliegue. Dicho fichero debería contener, al menos, una restricción para no subir los ficheros compilados (`*.pyc`).

Se pueden seguir las instrucciones que se indican en el siguiente enlace con el objetivo de configurar correctamente el proyecto para ser desplegado en Render:

`https://testdriven.io/blog/django-render`

Se deben tener en cuenta además las consideraciones que se indican a continuación:

- La aplicación web se debe iniciar, para este caso, de la forma `gunicorn locallibrary.wsgi`. Este parámetro se debe indicar en la configuración del servicio en Render.
- En el script `build.sh`, que se encuentra en la raíz del proyecto, se pueden añadir todos los comandos necesarios para poblar la base de datos, etc. Por otro lado, en el comando `createsu` que se ejecuta en el fichero script, se debe indicar el usuario y contraseña ya indicados (`alumnodb`).
- Se debe prestar especial atención a la gestión de ficheros estáticos. Esto es además especialmente conveniente de cara a futuras prácticas. Se recomienda crear una carpeta `static` en la raíz del proyecto.

- La base de datos PostgreSQL que tenemos en local no funcionará en Render, por lo que hay que utilizar el mecanismo adecuado para poder acceder a la base de datos de manera remota y persistir la información en ella. Para ello, se utilizará *neon.tech*, un servicio de PostgreSQL en la nube que nos permite crear y usar una base de datos en remoto, proporcionando una URL de acceso (a usar a través de la variable `dj_database_url` en Django, y también en Render. Para obtener este servicio, hay que registrarse en `https://www.neon.tech`. Recuerda actualizar la URL obtenida para el acceso a la base de datos en la variable anterior, en el fichero `settings.py`, y en Render (DATABASE_URL).

  Sin embargo, la BD creada en *neon.tech* no permitirá ejecutar los test, ya que no está permitido crear nuevas bases de datos en remoto, por lo que es necesario crear una variable de entorno que permita ejecutar los tests en PostgreSQL local y persistir los datos de la aplicación en remoto en *neon.tech*. Para ello, es necesario incluir la siguiente modificación en el fichero `settings.py`:

```
# To run the tests: export TESTING=1, or to use the app: unset TESTING
# To see the current value just type echo $TESTING

if 'TESTING' in os.environ:
   db_from_env = dj_database_url.config(default=POSTGRESQL_URL', conn_max_age=50
else:
   db_from_env = dj_database_url.config(default=NEON_URL, conn_max_age=500)

DATABASES['default'].update(db_from_env)
```

  Donde `POSTGRESQL_URL` es la URL para usar PostgreSQL en local, y `NEON_URL` representa la URL que *neon.tech* proporciona para acceder a la BD (y que deberás copiar de la interfaz web).

Asegúrate de ejecutar todos los test y comprobar que funcionan correctamente. Se debe entregar, además, un fichero de texto con el resultado de ejecutar la utilidad `coverage` sobre todos los test implementados, lo que debería reportar una cobertura aproximada del 100 %.

5. Material a entregar al finalizar la práctica

1. Incluye en la raíz del proyecto un fichero llamado `authors.txt`, con el nombre de los autores y el número de equipo (pareja).

2. Calcular la cobertura de los test mediante la utilidad `coverage`, generando un fichero en formato texto (`coverage.txt`) que debe ser incluido en la raíz de proyecto. El resultado debe ser una tabla similar a la que aparece en el Listado 1.

3. Se debe indicar claramente, y en el momento de la entrega en *Moodle*, la URL donde está desplegado el proyecto en Render. Esta información debe aparecer correctamente configurada en el fichero `settings.py` entregado con el proyecto (variable `ALLOWED_HOSTS`). En *neon.tech*, la base de datos debe estar poblada con el contenido del fichero `populate_catalog.py`, y la interfaz de administración debe ser accesible mediante el usuario y contraseña exigidos (`alumnodb`).

4. Asegurase de que **TODOS** los test, tanto los de *Moodle* como los desarrollados en el tutorial, son satisfechos por el código implementado. No es admisible modificar el código de los test proporcionados.

5. Subir a *Moodle*, en un único fichero ZIP, el proyecto con todos los ficheros exigidos y necesarios para ejecutar la aplicación desarrollada en esta práctica. En concreto, se deberá subir a *Moodle* el fichero obtenido tras ejecutar el comando `"zip -r ../assign1.zip ."` o similar, asegurándose de incluir todos los ficheros del proyecto, incluida la carpeta oculta `.git`. No entregar el entorno virtual utilizado ni los ficheros con extensión `*.pyc`.

6. Criterios de evaluación

La calificación de esta práctica será *Apto* o *No Apto*. Para aprobar, y obtener así una calificación de *Apto*, es necesario satisfacer TODOS los siguientes criterios:

- El código creado para esta práctica debe: (a) ejecutarse correctamente usando Python 3.9 y Django 3.2, y (b) satisfacer todas las comprobaciones realiza-

das por `flake8`, excepto lo relativo al código proporcionado y al generado automáticamente por Django.

- Los datos de la aplicación se deben persistir en una base de datos PostgreSQL, según las especificaciones dadas.
- La aplicación completa, desarrollada a lo largo del tutorial, debe estar desplegada en *Render*, y debe ser accesible a través de la URL suministrada. En *Render*, la base de datos debe estar poblada en *neon.tech*. Por otro lado, la interfaz de administración debe ser completa, pudiendo visualizar y cambiar los datos a conveniencia mediante el superusuario creado (`alumnodb`).
- El código creado en esta práctica debe estar almacenado en un repositorio privado mediante `git`. Debe haber constancia, por tanto, del trabajo realizado en `git` a través de la carpeta `.git`, que debe entregarse obligatoriamente.
- El código necesario para ejecutar la aplicación deben entregarse en *Moodle*, y debe ser posible ejecutarlo localmente.
- El código entregado debe satisfacer TODOS los test.
- Se incluye el resultado de ejecutar la utilidad `coverage` (ver Listado 1) en un fichero en formato texto llamado `coverage.txt`, situado en la raíz del proyecto.

NOTA: El código usado para corregir esta práctica será el subido a *Moodle*. En ningún caso se usará el código existente en el repositorio *git* o en *Render*.