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 | 01.01.2026 | JAMI | Primera versión. |
| 1.5 | 13.01.2026 | JAMI | Se han mejorado algunas explicaciones y aclarado conceptos en los criterios de evaluación. |

[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 las instrucciones suministradas en este enunciado y realizar, paso por paso, lo exigido en el siguiente tutorial. Nota: se deberá seguir la versión en inglés del tutorial en todo momento, ya que es la más actualizada:

`https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/Django`

De manera general para esta práctica, se tendrán en cuenta las siguientes consideraciones:

- El código se corregirá en un ordenador con sistema operativo *Ubuntu*, que es el que se encuentra disponible en los laboratorios de la EPS.
- La versión de Python a utilizar será, como mínimo, la 3.11. Se exige como mínimo esta versión, pudiéndose utilizar la versión instalada en los ordenadores de los laboratorios.
- La versión de *Django* a utilizar será la 5.2.2.
- Se recomienda utilizar, como entorno de desarrollo, el IDE Visual Studio Code.
- Se utilizará un entorno virtual para trabajar y crear las dependencias específicas para el desarrollo de esta práctica. Debido a su tamaño, este entorno no debe entregarse con la práctica.
- Se utilizará `git` (GitHub), 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.
- El código Python implementado debe satisfacer la guía de estilos *PEP8*, lo cual deberá verificarse mediante la utilidad `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. Para realizar todas las instalaciones de una sola vez, se puede ejecutar el comando: `pip3 install -r requirements.txt` dentro del entorno virtual creado (esto se explicará más adelante).

- En *Moodle* se encuentran los tests que se deberán pasar de manera obligatoria. La forma general de proceder será pasar los tests correspondientes a cada semana. Esto puede hacerse al final de la implementación exigida, pero los tests 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 tests debe corregirse en el código propio. Por tanto, los tests proporcionados en *Moodle* no se pueden modificar.

- 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 realizando el trabajo propuesto para cada semana, lo que implica estudiar e implementar lo exigido en los capítulos del tutorial, con los matices adicionales que se indican en este anunciado, y realizar además, de manera obligatoria, los ejercicios propuestos al final de cada parte del tutorial (*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. Se recomienda, por tanto, que cada estudiante trabaje en un ordenador distinto, coordinando el trabajo en equipo a través de GitHub.

# 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*). 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 el proyecto de la biblioteca (`locallibrary`), se deberá crear un nuevo repositorio *Git* para esta práctica. Además, se creará y activará un entorno virtual basado en la versión de Python a usar (se recuerda que, como mínimo, se debe usar la versión 3.11):

```
> python3.11 -m venv p1_env
> source p1_env/bin/activate
```

Puedes utilizar otras aplicaciones para la gestión avanzada de entornos virtuales, como `virtualenv` o `virtualenvwapper`. Se recuerda que el entorno virtual creado no debe entregarse con la práctica. Es importante no modificar el nombre del entorno virtual ni copiarlo de una máquina a otra, ya que cuando se crea se generan dependencias específicas, por lo que podría no funcionar bien en otro entorno diferente al que se utilizó para su creación. Con el entorno virtual activado, se debe instalar la versión exigida de Django junto con el resto de dependencias (librerías), utilizando para ello el comando ya explicado anteriormente (`>pip3 install -r requirements.txt`).

Según se indica en el tutorial, se debe utilizar el script `django-admin` para crear el proyecto. Este script normalmente se ubica en el entorno virtual una vez creado. También es posible utilizar, si se encuentra en la ruta de ejecución, el script general instalado en el sistema. En caso de que este script no funcione o no se encuentre en la ruta por algún motivo, se puede realizar la creación ejecutando el módulo correspondiente de *Django* desde Pyhton, utilizando para ello el comando: `>python -m django startproject locallibrary`. Esto permitirá crear el esqueleto del proyecto y, muy importante, generar el fichero `manage.py` necesario para gestionar el proyecto y las aplicaciones. En resumen, se puede utilizar cualquiera de estos tres comandos:

```
a) > django-admin <comando> [opciones]
b) > python3  manage.py <comando> [opciones]
c) > python3 -m django <comando> [opciones]
```

Una vez que se tenga un conocimiento sólido sobre 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. Sin embargo, sí se pueden ignorar las advertencias relacionas con el código suministrado o con las líneas de código generadas 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. Tests de la primera semana

En la página *Moodle* de la asignatura se encuentran los ficheros `test_xxxx_week.py`, los cuales forman una colección de tests 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 tests correspondientes (`test_first_week.py`). Si surgen errores, se debe modificar el código propio implementado para satisfacer los tests, y nunca el código de los tests proporcionados. Los tests 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 tests

Se utilizará la aplicación `coverage` para medir la cobertura de los tests, es decir, el porcentaje de código al que se accede desde los tests, lo que permitirá saber cuánto código estamos realmente probando. Para ello, es conveniente tener instalado `coverage`, incluido en el fichero de requisitos. 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 tests que haya en la carpeta anteriormente comentada, por lo que es conveniente ir añadiendo los tests 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/tests` los tests 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 tests de las semanas sucesivas. Además, el porcentaje se incrementará cuando se hayan implementado, más adelante, los tests exigidos en la Parte 10 del tutorial. Para alcanzar una cobertura del 100 % se programará un fichero llamado `test_additional.py` con tests creados por el estudiante que garanticen alcanzar la máxima cobertura. Este fichero se situará en la carpeta `catalog/tests`.

# 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 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`), cuyas versiones vienen especificadas en el fichero `requirements.txt`. Esto se explica

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% | |

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:

```
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 el capítulo 4 del tutorial, continuando con la implementación de la aplicación de la biblioteca, y realizando los ejercicios de ampliación que se proponen al final:

- *Django Tutorial Part 4: Django admin site.*

Antes de continuar con la parte 5 del tutorial, se deben adquirir conocimientos esenciales sobre HTML y CSS. Para ello, se deben estudiar las siguientes páginas correspondientes a tutoriales del W3C. Estos tutoriales se pueden seguir paso a paso, de manera incremental, mirando los contenidos y videos existentes, y realizando los ejercicios interactivos propuestos que puedes probar directamente en el contexto de dichas páginas:

* `https://www.w3schools.com/html/`

* `https://www.w3schools.com/css/`

Una vez finalizados los tutoriales del W3C, y adquiridos los conocimientos esenciales sobre HTML y CSS, completa las partes 5 y 6 del tutorial, llevando a cabo las ampliaciones propuestas al final de cada capítulo:

- *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 laboratorios, 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 `python3 manage.py createsuperuser`. Tanto el usuario como la contraseña a utilizar serán, en ambos casos, `alumnodb`.

## 3.1. Tests de la segunda semana

Los tests relacionados con el trabajo de esta segunda semana se encuentran en el fichero `test_second_weeky.py`. Se deben pasar los tests de la primera y segunda semana. Para ejecutar los tests, 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.*

Una vez finalizada la parte 8, lleva a cabo las siguientes tareas adicionales, que deberás realizar sobre los ficheros HTML y CSS de la aplicación web de la biblioteca propuesta en el tutorial de Django, lo que permitirá adquirir destrezas en el desarrollo de la interfaz de usuario:

a. Crea estilos específicos para los ítems listados en la página principal de la aplicación, de forma que cada uno aparezca enumerado con el número en un recuadro. Para ello, crea estilos (incluyéndolos en un fichero `frontpage-styles.css`) que afecten sólo a esa lista de elementos, y no al resto de los que aparecen en la aplicación web.

b. Crea dos botones para los enlaces de las funcionalidades *Login* y *Logout* que aparecen en el menú lateral de la aplicación, utilizando para ello estilos distintos para cada tipología de botón.

c. Añade también, en la parte inferior de la página, el nombre de los autores de la práctica, de manera que esta información sea siempre visible en todas las páginas.

En la Figura 1 puede verse un ejemplo de cómo quedaría la página principal de la aplicación web de la biblioteca con estos cambios, una vez implementada la aplicación con los añadidos exigidos.

Posteriormente, continua con los últimos capítulos del tutorial en el siguiente orden, y realizando los ejercicios adicionales comentados al final de cada capítulo:

- *Django Tutorial Part 9: Working with forms.*

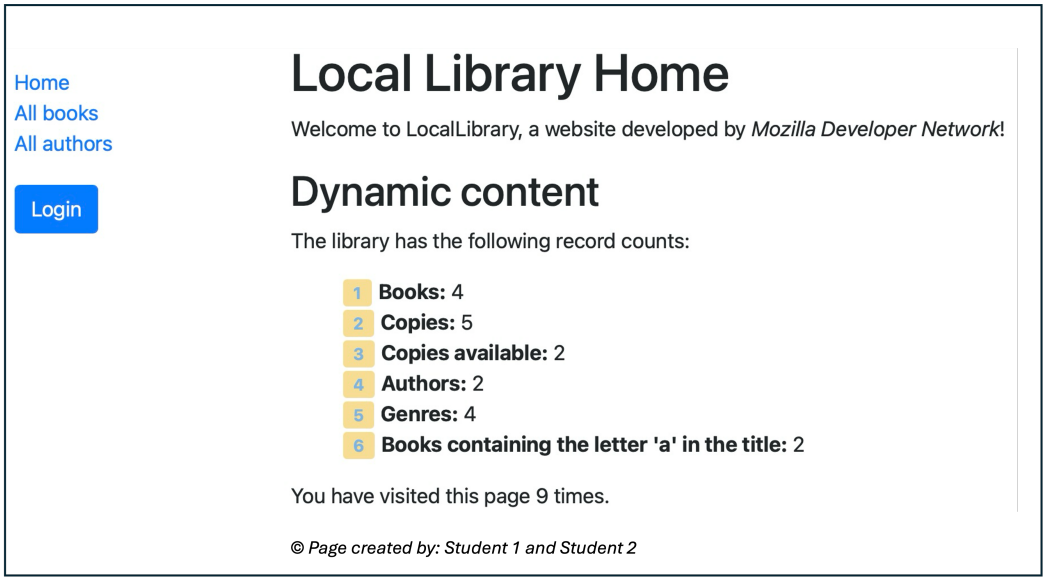


Figura 1: Modificaciones a realizar en la página principal de la aplicación web.

- *Django Tutorial Part 10: Testing a Django web application.*
- *Django web application security.*
- *Django Tutorial Part 11: Deploying Django to production.*

Al implementar los tests exigidos en la parte 10 del tutorial, es conveniente aclarar que los tests relacionados con la parte de modelos (`test_models.py`), contienen una serie de instrucciones del tipo "`.objects.get(id=1)`"que podrían ser problemáticas. Por ello, estas deben reemplazarse por instrucciones que hagan otro tipo de búsquedas más certeras (y no dependientes del orden) para evitar acceder por `id` al primer registro, evitando así problemas con la secuencia de auto-incremento de las claves de los diferentes objetos.

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. Tests de la tercera semana

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

nunca el código de los tests 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 tests a implementar en la Parte 10 del tutorial (*Django Tutorial Part 10: Testing a Django web application*), junto con aquellos tests implementados a partir del *challenge* propuesto al final de este capítulo. Todos los tests deben situarse en la carpeta `/catalog/tests`, 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 tests implementados en la parte 10 del tutorial, el sistema retorne el error comentando (*Missing staticfiles manifest entry*), el cual puede solucionarse siguiendo las instrucciones suministradas dentro del propio tutorial.

## 4.2. Consideraciones adicionales sobre Render

Render es una *plataforma como servicio* de computación en la nube (PaaS) que permite desplegar una aplicación web 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 pasando todos los tests primero en el entorno de desarrollo, y preparar finalmente la aplicación, como último paso, para ser desplegada en el 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 al mostrado en el Listado 2, donde aparecerán tanto las dependencias principales como secundarias instaladas automáticamente.

Listado 2: Fichero `requirements.txt`.

```
asgiref==3.11.0
coverage==6.4.3
dj-database-url==3.0.0
Django==5.2.2
django-extensions==4.1
flake8==7.3.0
gunicorn==23.0.0
mccabe==0.7.0
packaging==25.0
pillow==12.0.0
psycopg2==2.9.10
pycodestyle==2.14.0
pyflakes==3.4.0
python-dotenv==1.1.1
sqlparse==0.5.5
typing_extensions==4.15.0
whitenoise==6.9.0
```

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 tendría limitaciones no convenientes. La conexión con el gestor de bases de datos se explicará más adelante.

Las URLS creadas en Render deben tener el siguiente formato:

`https://Ppractica_equipo_grupo_año_version.onrender.com/`

Por ejemplo, la URL:

`https://P1_10_2312_2026_1.onrender.com/`

denotaría la práctica 1 (P1) del equipo (pareja) 10 del grupo 2312 del curso 2026 en su versión 1.

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 una 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 correspondiente 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 del fichero `settings.py` y en Render.

  Es importante indicar que la BD creada en *neon.tech* no permitirá ejecutar directamente los tests. Aunque es posible hacerlo, sería necesario indicar una configuración adicional más compleja en `settings.py`. Sin embargo, como además hay que cambiar de un entorno de desarrollo a otro de producción, y además eliminar posteriormente el mayor número posible de variables sensibles del fichero de configuración, se recomienda ejecutar los tests en local, usando PostgreSQL, y utilizar *neon.tech* para persistir los datos de la aplicación en remoto, ideando algún un mecanismos que permita cambiar de un gestor a otro. Esto se puede hacer mediante la siguiente modificación en el fichero `settings.py`:

```
# To use local PostgreSQL and run the tests, just export TESTING=1
# To use NEON, just unset TESTING
# To see the current value of TESTING just type echo $TESTING
```

```
if 'TESTING' in os.environ:
   db_from_env = dj_database_url.config(default=POSTGRESQL_URL,
      conn_max_age=500)
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 de *neon.tech*). La utilización de esta variable de entorno permitirá cambiar entre un gestor de bases de datos local u otro remoto en la nube, permitiendo así poblar la base de datos en *neon.tech*.

Otra variable de entorno importante a considerar (como se indica en el tutorial) es `DEBUG`. Con `DEBUG = True`, Django muestra una página técnica de error con:

- La traza completa del error.
- Variables locales de cada marco de la pila.
- Información del entorno, versión de Django, etc.
- Plantillas y contexto si el error ocurrió al renderizar.

La variable `DEBUG` debe tomar el valor 'false' cuando la aplicación se encuentre en el entorno de producción para evitar dar como salida una traza en caso de error y mostrar valores sensibles, y tomar el valor 'true' cuando la aplicación se encuentre en el entorno de desarrollo, para facilitar la localización de errores. Puedes utilizar además esta variable para alterar la configuración de `settings.py` según nos encontremos en un entorno u otro.

Por otro lado, Django no permite servir archivos estáticos en producción, por lo que debemos utilizar *WhiteNoise* (ya incluido en el fichero de requisitos suministrado). Además de la configuración establecida en el apartado de despliegue, es posible que tengas que modificar/añadir las siguientes líneas en `settings.py`, más acordes con la nueva versión de Django utilizada:

```
...
DEFAULT_FILE_STORAGE = 'django.core.files.storage.FileSystemStorage'
```

```
STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestSta3cFilesStorage'
...
```

Asegúrate de ejecutar todos los tests y comprobar que funcionan correctamente. Se debe entregar, además, un fichero de texto, situado en la raíz del proyecto, con el resultado de ejecutar la utilidad `coverage` sobre todos los tests implementados, indicando una cobertura máxima.

## 4.3. Gestión de variables de entorno

A la hora de trabajar en este tipo de proyectos, es conveniente exportar todas las variables de entorno, junto con otras de naturaleza más sensible, para que no sean visibles desde los ficheros de código y configuración. Esto incluye las variables de entorno necesarias para ejecutar la aplicación y algunas de las variables utilizadas en `settings.py`, como por ejemplo `SECRET_KEY, POSTGRESQL_URL, NEON_URL`, etc., que muestran en el código programado nombres de usuarios y contraseñas. Para ello, se puede utilizar la librería `pyhton-dotenv`, que permite leer desde `settings.py` las variables contenidas en un fichero `.env`:

```
import os
from dotenv import load_dotenv
load_dotenv() # Se puede parametrizar la ruta del fichero .env
...
SECRET_KEY = os.getenv('SECRET_KEY')
```

Sitúa y documenta, en un fichero `.env`, ubicado en la raíz del proyecto, todas las variables sensibles y necesarias para la ejecución del proyecto, incluyendo las de entorno y aquellas relacionadas con el despliegue de la aplicación, y léelas desde el código mediante la librería `pyhton-dotenv`.

## 4.4. Listado de endpoints

Finalmente, y como ayuda a la implementación y corrección, se instalarán lo que se denominan las *extensiones de Django* (*Django Extensions*) que permiten, entre otras funciones, tener acceso mediante el comando `show_urls` a las distintas URLs o endpoints definidos en el proyecto. La documentación de ayuda se encuentra en las siguientes direcciones web:

```
https://django-extensions.readthedocs.io/en/latest/
https://django-extensions.readthedocs.io/en/latest/command_extensions.html
```

El primer paso sería instalar las extensiones mediante el comando `pip3`. Sin embargo, esto ya viene definido en el fichero de requisitos suministrado, y se entiende que ya se habrá realizado la instalación en el entorno virtual al comienzo.

Seguidamente, hay que añadir la siguiente línea en `INSTALLED_APPS` dentro del fichero de configuración de Django (`settings.py`):

```
INSTALLED_APPS = (
...
'django_extensions',
...
)
```

Esto permitirá visualizar los endpoints definidos en el proyecto ejecutando directamente el comando:

```
> python3 manage.py show_urls
```

Realiza esta instalación y comprueba que se visualizan correctamente todas las URLs del proyecto junto con sus vistas correspondientes.

# 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. Incluir también la carpeta `.git` del proyecto realizado.

3. Calcular la cobertura de los tests 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. Se debe intentar llegar a una cobertura máxima (100 %), lo que implica añadir nuevos tests es un fichero `test_additional.py` si fuera el caso.

4. Se debe indicar claramente, y en el momento de la entrega en *Moodle*, la URL donde está desplegado el proyecto en Render. La URL debe tener el formato indicado en el enunciado. Esta información debe aparecer correctamente configurada en al 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`).

5. Comprobar que todas las variables sensibles y de entorno están gestionadas correctamente mediante `python-dotenv`, y almacenadas en un fichero `.env`.

6. El proyecto desarrollado debe permitir su ejecución tanto en local como en remoto, usando para ello las variables de entorno apropiadas y comentadas en el enunciado, las cuales serás accesibles desde el fichero `.env` situado en la raíz del proyecto.

7. Comprobar que se han instalado las extensiones de Django, y que se pueden visualizar los endponits mediante el comando correspondiente.

8. Asegurase de que **TODOS** los tests, tanto los de *Moodle* como los desarrollados en el tutorial, se satisfacen con el código implementado. Se recuerda que no se puede modificar el código de los tests proporcionados.

9. 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, aquellos con extensión `*.pyc`, ni tampoco otros ficheros pesados e innecesarios.

# 6. Criterios de evaluación

Esta práctica se calificará con APTO o NO APTO.

Aunque se tendrá en cuenta la calidad y completitud de cada uno de los requisitos requeridos en el enunciado y en el tutorial, a modo de recomendación se indica que **para aprobar y obtener la calificación de APTO en esta práctica** es necesario satisfacer los siguientes criterios:

- El código creado debe ejecutarse correctamente usando Python (versión igual o superior a la 3.11) y Django 5.2.2. Además, la programación debe realizarse en equipo y el código se debe gestionar mediante un repositorio privado `git`. Debe haber constancia, por tanto, del trabajo realizado por los miembros del equipo en `git` (se mirarán los *commits* realizados por cada estudiante) a través de la carpeta `.git` que debe entregarse obligatoriamente.
- El proyecto debe funcionar correctamente y sin errores, y se deben realizar todas las implementaciones descritas tanto en el enunciado de esta práctica como en el tutorial, permitiendo realizar todas las operaciones exigidas con los libros de la biblioteca. Se deben incluir además todas las implementaciones adicionales (*challenges*) correspondientes a cada capítulo.
- Además, la interfaz de administración debe ser completa, pudiendo visualizar y cambiar los datos a conveniencia mediante el superusuario creado (`alumnodb`).
- Los datos de la aplicación deben persistir en una base de datos PostgreSQL en *neon.tech*, según las especificaciones dadas.
- La aplicación debe funcionar correctamente en el entorno de producción de Render.com, siguiendo las especificaciones, composición del nombre de la URL y persistencia de datos exigidos.
- Las variables de entorno deben gestionarse a través de `python-dotenv`, entregando un fichero `.env` donde se almacenarán estas variables, incluidas las URLs de Render y *neon.tech*. Estas variables serán importadas en settings.py adecuadamente. En Render, tanto `DEBUG=False` como `SECRET_KEY` deben leerse de una variable de entorno.
- Los tests no deben devolver ningún error. Además, se debe entregar el fichero de texto generado con `coverage`, llegando a una cobertura cercana al 100 %.

Al margen de estos criterios, se recomienda a los estudiantes que alcancen el máximo nivel de completitud y comprensión posibles para una mejor adquisición de conocimientos sobre Django, lo cual repercutirá positivamente en el examen a realizar y en la confección de las siguientes prácticas, que utilizan Django como tecnología base.

NOTA: En caso de realizar la entrega fuera de plazo, se substraerá un punto de la nota del examen por cada día (o fracción) de retraso en la entrega.