Tutorial Python + Django

1. Lenguaje y framework

El lenguaje de programación a utilizar es Python en conjunto con el Framework Django que suele ser muy utilizado para el desarrollo web.

En primer lugar es necesario instalar Python, ya sea desde el sitio web https://www.python.org/downloads/, o en las tiendas de aplicaciones como la Windows Store.

Una vez instalado Python, procedemos a instalar las librerías correspondientes y el Framework Django. Para efectos de este tutorial la base de datos a utilizar es MySQL.

Para la instalación de las librerías, hay que abrir la terminal o el shell del sistema operativo y digitar el siguiente comando:

```shell
# Instalación Framework Django
pip install Django

# Instalación cliente de MySQL
pip install mysqlclient

# Instalación libreria para leer .env
pip install python-dotenv
```

Una vez instaladas las librerías correspondientes, podemos generar nuestro proyecto Django de la siguiente manera:

1. Primero creamos una carpeta donde estará almacenado nuestro proyecto (Esto es opcional puesto que se puede hacer directamente en el escritorio).
2. Dentro de esta carpeta abrimos el terminal, o en cuyo caso se abre el shell del sistema operativo y con comandos cd dirigirse hasta la carpeta donde estará el proyecto.
3. Una vez aquí se ejecuta el siguiente comando:

```shell
# En este caso CookShare es el nombre del proyecto, este puede ser
modificado:
django-admin startproject CookShare

# En caso de algún error se puede hacer de esta manera:
python -m django startproject CookShare
```

```
# Posteriormente se ejecuta:
cd CookShare
```

4. En este punto, podemos hacer diferentes modificaciones para el proyecto o también se puede correr el proyecto para ver si todo ha salido bien:

```
Shell
python manage.py runserver
```

5. Después de correr el comando anterior, obtendremos un enlace para ver el proyecto.

Con estos simples pasos, tenemos un proyecto Django listo para modificar y utilizar, sin embargo, para efectos de este tutorial tenemos que realizar ciertas modificaciones para ver la base de datos en vistas personalizadas.

2. Librerías ORM utilizadas

Nota: Django por defecto utiliza MVT (Modelo, vista, template), por lo cual utilizaremos este patrón de diseño para efectos de este tutorial. Por otro lado, cabe aclarar que Django ya nos provee de librerías ORM muy completas, por lo cual no es necesario instalar más dependencias.

En primera instancia, antes de utilizar cualquier librería ORM, crearemos una aplicación donde pondremos los modelos de las tablas y por consiguiente utilizaremos las librerías ORM.

En la misma terminal que teníamos anteriormente realizaremos el siguiente comando:.

```
Shell
# core es el nombre de nuestra aplicación
python manage.py startapp core
```

Una vez que tenemos creada nuestra aplicación, dentro de esta carpeta tendremos un archivo que se denomina **Models.py** dentro de este archivo pondremos una estructura de la siguiente manera:

```
Python
from django.db import models # Librerias ORM
```

```python
class Usuario(models.Model):
    correo = models.EmailField(max_length=255, primary_key=True)
    nombre = models.CharField(max_length=255)
    contraseña = models.CharField(max_length=255, db_column='contrasena')

    class Meta:
        managed = False      # si la tabla ya existe
        db_table = 'usuario'
```

Nosotros partimos que ya tenemos una tabla usuario en la base de datos, por este motivo se utiliza manage = False, para evitar que se sobreescriba o se cree otra tabla cuando se realicen las migraciones en caso de ser necesarias.

En este caso, la clase Usuario está modelando la tabla usuario que tenemos en nuestra base de datos con la siguiente estructura:

```sql
CREATE TABLE `usuario` (
  `correo` varchar(255) PRIMARY KEY,
  `nombre` varchar(255) NOT NULL,
  `contrasena` varchar(255) NOT NULL
);
```

Una vez que tenemos nuestro modelo, debemos hacer la configuración pertinente de nuestra base de datos, puesto que por defecto Django utiliza sqlite3 en lugar de MySQL, por consiguiente, tendremos que reemplazar en el archivo **/CookShare/settings.py** lo siguiente:

```python
"""Remplazamos"""
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

"""Por:"""
import os
from pathlib import Path
from dotenv import load_dotenv

# BASE_DIR apunta a la carpeta donde está manage.py
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
# Cargar el archivo .env
env_path = BASE_DIR.parent / '.env'
load_dotenv(dotenv_path=env_path)
```

La variable env path hay que modificarla acorde al sitio donde se encuentra nuestro .env normalmente se encuentra en `BASE_DIR / '.env`. Esta modificación es solo para cargar el archivo .env, dado que es importante para nuestra estructura.

Una vez hecho esto, tenemos que incluir nuestra aplicación, por consiguiente realizamos lo siguiente:

Python
```
"""Remplazamos"""
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

"""Por: """
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'core',
]
```

Nota: core, es nuestra aplicación donde configuramos nuestro modelo usuario.

3. Configuración y levantamiento de la base de datos

Como lo mencionamos anteriormente, utilizaremos MySQL, lo haremos utilizando contenedores docker y variables de entorno (.env).

Para realizar esto, tenemos que modificar nuevamente nuestro archivo **Settings.py**, realizando lo siguiente:

```python
"""Remplazamos"""
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

"""Por: """
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': os.getenv('DB_NAME'),
        'USER': os.getenv('DB_USER'),
        'PASSWORD': os.getenv('DB_PASSWORD'),
        'HOST': os.getenv('DB_HOST'),
        'PORT': os.getenv('DB_PORT'),
    }
}
```

Con esto ya tendremos configurada nuestra base de datos en nuestro proyecto django, sin embargo, no se establecerá ninguna conexión puesto que hasta el momento no hemos levantado nuestra base de datos.

Esto lo realizaremos con docker utilizando el siguiente comando en el sitio donde se encuentre nuestro archivo docker_compose.yml, mismo sitio donde debería estar nuestro archivo .env

```shell
docker compose up -d
```

Siendo el archivos docker compose.yml el siguiente:

```
None
services:
  db:
    container_name: mysql-demo-compose
    image: mysql:8.0
    ports:
      - "${DB_PORT}:3306"
    environment:
      MYSQL_ROOT_PASSWORD: ${DB_PASSWORD_ROOT}
      MYSQL_DATABASE: ${DB_NAME}
      MYSQL_USER: ${DB_USER}
      MYSQL_PASSWORD: ${DB_PASSWORD}
    volumes:
      - mysqldata:/var/lib/mysql

volumes:
  mysqldata:
```

El archivo .env debería tener un contenido como el siguiente:

```
None
# Database
DB_USER=testuser
DB_PASSWORD=password
DB_PASSWORD_ROOT=rootpassword
DB_NAME=testdb
DB_HOST=localhost
DB_PORT=3306
```

Estos parámetros deberían cambiar dada la configuración que se requiera para la base de datos.

Por último si se tiene una archivo de inicialización para la base de datos se debería ejecutar un comando como el siguiente:

```
Shell
docker exec -i mysql-demo-compose mysql -u testuser -ppassword testdb <
./ubicacion_del_archvo_init/init.sql
```

Con esto ya es posible establecer una coneccion con la base de datos desde nuestro proyecto Django

4. Ejecución Final

En este punto, ya tenemos la conexión con la base de datos, una instalación mínima de entidades como objetos de nuestra base de datos.

Con esto solo nos falta utilizar nuestro objeto usuario, para realizar operaciones CRUD y un componente visual para ver los datos.

1. Creamos los templates donde se verá nuestra información, para este caso, crearemos una carpeta dentro de nuestra aplicación core llamada templates/usuarios/ donde tendremos boton_usuario.html y lista_usuarios.html que se verán de la siguiente forma:

HTML

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <title>Botón Usuarios</title>
    <style>
        .btn {
            display: inline-block;
            padding: .5rem 1rem;
            background: #007bff;
            color: #fff;
            border: none;
            border-radius: .25rem;
            text-decoration: none;
            font-size: 1rem;
            cursor: pointer;
        }
        .btn:focus { outline: 3px solid rgba(0,123,255,.5); outline-offset:
2px; }
    </style>
</head>
<body>
    <!-- Botón que redirige a /usuarios/ -->
    <button type="button" class="btn"
onclick="window.location.href='/usuarios/'" aria-label="Ir a usuarios">
        Ir a Usuarios
    </button>

    <!-- En caso de que JavaScript esté deshabilitado -->
    <noscript>
        <a class="btn" href="/usuarios/">Ir a Usuarios</a>
    </noscript>
</body>
```

```
</html>
```

HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Catálogo de Productos</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f6f6f6;
            padding: 20px;
        }
        .usuarios {
            display: flex;
            flex-wrap: wrap;
            gap: 20px;
        }
        .usuario {
            background-color: white;
            border-radius: 10px;
            width: 250px;
            box-shadow: 0 2px 5px rgba(0,0,0,0.1);
            overflow: hidden;
            transition: 0.3s;
        }
        .producto:hover {
            transform: translateY(-5px);
        }
        .producto img {
            width: 100%;
            height: 180px;
            object-fit: cover;
        }
        .producto .info {
            padding: 10px 15px;
        }
        .producto h3 {
            font-size: 18px;
            margin-bottom: 5px;
        }
        .producto p {
            color: #555;
        }
        .correo {
            font-weight: bold;
```

```
            color: #007b00;
            font-size: 16px;
        }
        a {
            text-decoration: none;
            color: inherit;
        }
    </style>
</head>
<body>
    <h1>Lista de usuarios registrados</h1>
    <div class="usuarios">
        {% for usuario in usuarios %}
        <div class="usuario">
            <div class="info">
                <h3>{{ usuario.nombre }}</h3>
                <p class="correo">Correo: {{ usuario.correo }}</p>
                <p>Contraseña: {{ usuario.contraseña }}</p>
            </div>
        </div>
        {% endfor %}
    </div>
</body>
</html>
```

2. Crear un view para para visualizar nuestros datos, en este caso, tenemos que modificar el archivo **views.py** realizando lo siguiente:

Python

```python
"""Remplazar"""
from django.shortcuts import render

# Create your views here.

"""Por: """
from django.shortcuts import render
from .models import Usuario

def lista_usuarios_view(request):
    usuarios = Usuario.objects.all()
    return render(request, 'usuarios/lista_usuarios.html', {'usuarios':
usuarios})

def boton_usuario_view(request):
```

```
    return render(request, 'usuarios/boton_usuario.html')
```

3. Creamos un archivo **urls.py** dentro de core, el cual tendrá el siguiente contenido:

```python
from django.urls import path
from .views import lista_usuarios_view, boton_usuario_view

urlpatterns = [
    path('usuarios/', lista_usuarios_view, name='lista_usuarios'),
    path('', boton_usuario_view, name='boton_usuario'),
]
```

Nota: Esto simplemente se utiliza para mapear las vistas con la url que se verá en nuestra aplicación web.

4. Por ultimo, solo tenemos que modificar el archivo **urls.py** de la carpeta CookShare, realizando lo siguiente:

```python
"""Remplazar"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]

"""Por: """
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('core.urls')), # Incluye las rutas de nuestra
aplicación core
]
```

5. Una vez realizado esto, simplemente podemos levantar el servidor y ver nuestro programa funcionando utilizando la url que nos proporcionan al momento de ejecutar desde nuestra terminal en la carpeta donde se encuentra **manage.py** el siguiente comando en el shell del sistema operativo:

```shell
python manage.py runserver
```

Esto nos levantará un servidor para acceder a nuestro programa desde el siguiente enlace local: http://127.0.0.1:8000/

Para casos practicos el siguiente enlace es para ver nuestra implementacion con los paso realizados aqui: https://github.com/jbeltrano/social_group/tree/Hola_Mundo

Nota final: Este es un tutorial para la iniciación de un proyecto en Django, utilizando MVT, puesto que por practicidad y para fines del tutorial lo indicado es utilizar este patrón de diseño en lugar de MVC, este lo utilizaremos si o si para el proyecto en general.