Biblioteca - Mariza Valdez

Versiones de Herramientas utilizadas

```
Package                       Version
----------------------------- -----------
asgiref                       3.9.1
contourpy                     1.3.2
cycler                        0.12.1
Django                        5.2.4
djangorestframework           3.16.0
djangorestframework_simplejwt 5.5.0
fonttools                     4.58.5
kiwisolver                    1.4.8
matplotlib                    3.10.3
numpy                         2.3.1
packaging                     25.0
pandas                        2.3.1
pillow                        11.3.0
pip                           24.3.1
psycopg2                      2.9.10
PyJWT                         2.9.0
pyparsing                     3.2.3
python-dateutil               2.9.0.post0
pytz                          2025.2
seaborn                       0.13.2
six                           1.17.0
sqlparse                      0.5.3
tzdata                        2025.2
```

Guía de instalación y configuración

Instalación de Python

Asegúrate de tener Python 3.8+ instalado. Puedes descargarlo desde python.org. Para verificar la instalación:

```
python --version
```

Creación de entorno virtual

Se recomienda usar un entorno virtual para aislar las dependencias del proyecto. En este ejemplo, el entorno se llama ent.

```
# Abre una terminal en la carpeta del proyecto
python -m venv ent
```

Para activar el entorno virtual:

En Windows:

```
ent\Scripts\activate
```

En Linux/Mac:

```
source ent/bin/activate
```

Instalación de Django y psycopg2

Con el entorno activado, instala Django y el conector para PostgreSQL:

```
pip install django psycopg2-binary
```

Creación del proyecto Django

Si aún no creaste tu proyecto, hazlo así (ejemplo: se llamará biblioteca):

```
django-admin startproject biblioteca
cd biblioteca
```

Creación de una aplicación

Ejemplo, crea una app llamada libros:

```
python manage.py startapp libros
```

Configuración de la base de datos PostgreSQL

Edita el archivo biblioteca/settings.py y busca la sección DATABASES. Cámbiala así (usa tus propios datos de usuario, password, etc):

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'biblioteca',
        'USER': 'postgres',
        'PASSWORD': '123',          # Cambia por tu password real
```

```
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Migraciones iniciales

Aplica las migraciones para crear las tablas base:

```
python manage.py makemigrations
python manage.py migrate
```

Descripción y Funcionamiento del Programa

Este proyecto es un sistema de gestión y análisis de libros desarrollado en Django, utilizando PostgreSQL como base de datos. Permite registrar libros, autores, géneros, usuarios y calificaciones, así como generar reportes gráficos y recomendaciones personalizadas.

¿Cómo funciona el programa?

Modelo de datos: El programa utiliza el framework Django para estructurar los datos en modelos relacionales:

- Autor: Almacena nombre y nacionalidad del autor.
- Género: Define el género literario.
- Libro: Incluye título, autor, género, año de lanzamiento, ISBN y enlace.
- Calificación: Relaciona usuarios con libros, almacenando una nota del 1 al 5.

Usuarios y autenticación: Se utiliza el sistema de usuarios de Django, permitiendo el registro, inicio de sesión y asignación de calificaciones por cada usuario, Se utiliza JWT para autenticacion.

Administración y reportes: El programa permite gestionar la información desde la interfaz administrativa de Django. Además, incluye comandos personalizados para la consola que generan reportes automáticos en forma de gráficos (por ejemplo: top autores, libros mejor calificados, etc.) y para recomendaciones de libros.

Comandos personalizados: Los comandos escritos con BaseCommand permiten analizar los datos de la base y producir resultados útiles, por ejemplo:

- Generar gráficos que resumen visualmente la información de la biblioteca (cantidad de libros por género, evolución anual, mejores autores, etc).
- Recomendar libros a partir del género seleccionado, mostrando los títulos con mejores calificaciones.

Citación del funcionamiento

```
"El usuario puede ejecutar comandos desde la consola como python manage.py
reportes_libros para generar gráficos estadísticos en la carpeta /graficos, o
python manage.py recomienda_libros --genero=Fantasía para obtener sugerencias de
lectura personalizadas por género.
```

```
Todo el procesamiento se realiza sobre los datos almacenados en la base
PostgreSQL, utilizando las capacidades ORM de Django para consultas eficientes y
seguras."
```

El programa está diseñado para ser fácil de instalar, configurar y expandir, siguiendo las buenas prácticas de desarrollo con Django y asegurando compatibilidad multiplataforma.

Prueba de la Aplicacion

Registrarse!

Url de la api registro:

```
POST http://127.0.0.1:8000/api/register/
```

Cargamos los datos en Json:

```
{
  "username": "mariza",
  "email": "marizavaldez.alta@gmail.com",
  "password": "12345678!",
  "password2": "12345678!",
  "first_name": "Mariza",
  "last_name": "Valdez"
}
```

La respuesta en postman:

Codigo para el registro:

```
#serializers.py
class UserRegisterSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True, required=True, min_length=8)
    password2 = serializers.CharField(write_only=True, required=True,
label="Confirmar contraseña")

    class Meta:
        model = User
        fields = ['username', 'email', 'password', 'password2', 'first_name',
'last_name']

    def validate(self, data):
        if data['password'] != data['password2']:
            raise serializers.ValidationError("Las contraseñas no coinciden.")
        if User.objects.filter(username=data['username']).exists():
            raise serializers.ValidationError("El nombre de usuario ya está en
uso.")
```

```
        if 'email' in data and User.objects.filter(email=data['email']).exists():
            raise serializers.ValidationError("El email ya está registrado.")
        return data

    def create(self, validated_data):
        validated_data.pop('password2')
        password = validated_data.pop('password')
        user = User(**validated_data)
        user.set_password(password)
        user.save()
        return user
#views.py
class RegisterView(generics.CreateAPIView):
    serializer_class = UserRegisterSerializer
    permission_classes = [AllowAny]  # Permitir acceso sin autenticación
```

Inicio de Sesion

Url de la api login:

```
POST http://127.0.0.1:8000/api/login/
```

Cargamos los datos en Json:

```
{
  "username": "mariza",
  "password": "12345678!"
}
```

La respuesta en postman:

El Login genera un access token, que luego va a servir para autenticar las demas aplicaciones.