Asignación/proyecto práctico diseñado para integrar **Django**, **Flask** y **SQLite3** en una arquitectura de microservicios o sistemas distribuidos, ideal para consolidar habilidades de desarrollo backend.

Proyecto: Sistema Híbrido de Gestión de Tareas y Analítica (Django + Flask + SQLite3)

🎯 Objetivo General

Desarrollar una aplicación web compuesta por dos servicios independientes que compartan la misma base de datos en SQLite3:

1. **Servicio Django (Core / Panel de Administración):** Gestionará la autenticación de usuarios, la lógica principal de negocio (creación y administración de tareas/proyectos) y ofrecerá un panel de administración completo.
2. **Servicio Flask (API de Analítica / Reportes):** Leerá directamente de la base de datos para generar métricas, estadísticas y reportes en tiempo real, operando como un servicio aligerado y rápido.

🛠️ Requisitos Técnicos

- **Lenguaje:** Python 3.10+
- **Framework 1:** Django (para el CRUD principal y autenticación).
- **Framework 2:** Flask (para el microservicio de analítica/reportes).
- **Base de Datos:** SQLite3 (ambos servicios deben apuntar al mismo archivo .sqlite3).

💼 Requerimientos Funcionales por Módulo

1. Módulo Django (Core & Admin)

- **Modelos (**models.py**):**
    - Usuario (puedes usar el modelo por defecto de Django).
    - Proyecto: Campos como nombre, descripcion, fecha_creacion.
    - Tarea: Campos como titulo, descripcion, estado (*Pendiente, En Progreso, Completada*), prioridad (*Baja, Media, Alta*), fecha_limite, y una relación ForeignKey con Proyecto.

- **Funcionalidades:**
    - Panel de administración de Django configurado para gestionar Proyectos y Tareas con filtros personalizados (por estado y prioridad).
    - Vistas protegidas (requiriendo login) donde los usuarios puedan registrarse, iniciar sesión y ver sus propios proyectos y tareas.

2. Módulo Flask (Microservicio de Analítica)

- **Funcionalidades:**
    - Conectarse a la misma base de datos SQLite generada por Django utilizando SQLAlchemy (modo reflectivo/solo lectura) o la librería nativa sqlite3.
    - **Endpoints REST obligatorios:**
        - GET /api/stats/general: Devuelve un JSON con el conteo total de tareas agrupadas por estado (cuántas pendientes, en progreso y completadas).
        - GET /api/stats/priority: Devuelve estadísticas de tareas agrupadas por nivel de prioridad.
        - GET /api/report/export: Genera una vista HTML simple o un archivo CSV resumido con el rendimiento general de los proyectos.

🚀 **Fases Sugeridas para el Desarrollo**

Fase 1: Configuración del Core (Django)

- Inicializa el proyecto Django y configura la base de datos SQLite3.
- Crea las migraciones para los modelos Proyecto y Tarea.
- Registra los modelos en admin.py y crea algunos registros de prueba desde el panel.

Fase 2: Conexión y Microservicio (Flask)

- Configura una app de Flask en una carpeta independiente.
- Establece la conexión a la base de datos SQLite ubicada en el proyecto Django (../django_project/db.sqlite3).
- Desarrolla los endpoints de estadísticas y verifica que devuelvan la información correcta creada desde Django.

2. **Fase 3: Integración y Pruebas**
    - Asegúrate de que ambos servicios puedan correr simultáneamente (Django en el puerto 8000 y Flask en el puerto 5000).
    - Opcional: Consume los endpoints de Flask desde una plantilla o vista en Django mediante peticiones HTTP (por ejemplo, usando requests) para mostrar gráficos o resúmenes estadísticos en el panel del usuario.

📌 Criterios de Evaluación / Entrega

- **Estructura de Código:** Separación limpia de responsabilidades (código modular, uso de variables de entorno si es necesario).
- **Persistencia Compartida:** Demostración de que ambos frameworks leen y/o escriben correctamente en la misma base de datos SQLite sin corromper el archivo.
- **Documentación:** Un archivo README.md que explique paso a paso cómo clonar, instalar dependencias, ejecutar las migraciones y levantar ambos servidores (Django y Flask).