Prueba Técnica: Sistema de Gestión de Tareas con Django y Consumo de Servicio Externo (DRF)
Objetivo: Evaluar el dominio de Django (modelos, vistas, templates) y la capacidad para consumir servicios externos usando Django REST Framework (DRF).

Descripción del Proyecto

Desarrollar una aplicación web (task_manager) para gestionar tareas con las siguientes funcionalidades:

1. Crear, editar, marcar como completada y desactivar tareas.
2. Listar tareas con filtros (completadas/no completadas).
3. Consumir un servicio externo para obtener datos adicionales (ej: clima) y mostrarlos en la vista de detalle de una tarea.

Requisitos Técnicos

1. Configuración Inicial

- Proyecto Django (task_manager_project) y app (task_manager).
- Base de datos: SQLite.

2. Modelo de Datos

- Crear el modelo Task:

```
class Task(models.Model):
   title = models.CharField(max_length=200)
   description = models.TextField(blank=True)
   completed = models.BooleanField(default=False)
   created_at = models.DateTimeField(auto_now_add=True)
   due_date = models.DateField(null=True, blank=True)  # Plus si se implementa

   def __str__(self):
      return self.title
```

3. Vistas y Templates

- **Vistas basadas en clases:**
    - TaskListView: Lista todas las tareas, con filtro por completed (usar GET parameters).
    - TaskDetailView: Muestra detalles de una tarea.
- **Vistas basadas en funciones:**
    - create_task: Formulario para crear tareas (usar django.forms.ModelForm).
    - mark_task_completed: Cambia el estado completed de una tarea (vía POST).
- **Templates:**

- tasks/list.html: Tabla con lista de tareas, enlaces a detalles, formulario de filtro.
- tasks/detail.html: Detalle de la tarea + botones para editar, eliminar o marcar como completada.

4. Formularios

- Crear TaskForm (basado en ModelForm) para crear/editar tareas.
- Validación:
    - title no puede estar vacío.
    - due_date debe ser mayor o igual a created_at (plus si se implementa).

5. Consumo de Servicio Externo con DRF

- **Servicio Externo:** Usar OpenWeatherMap para obtener el clima de una ciudad.
- **Requerimientos:**
    - Añadir campo city en el modelo Task.
    - En la vista TaskDetailView, consumir la API de OpenWeatherMap para mostrar el clima actual de la ciudad asociada a la tarea.
    - Endpoint de la API: https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}.
    - Mostrar en el template: temperatura, humedad y descripción del clima.
    - Manejar errores: ciudad no encontrada, API no disponible.

6. URLs

- Mapear URLs lógicas:
    - /tasks/: Listado.
    - /tasks/<int:pk>/: Detalle.
    - /tasks/create/: Crear.
    - /tasks/<int:pk>/complete/: Marcar como completada.

7. Testing

- Escribir tests para:
    - Creación de tareas (validar que title no esté vacío).
    - Cambio de estado completed.
    - Consumo de la API externa (usar unittest.mock para simular respuestas).

8. Admin de Django

- Registrar Task en el admin.

- Permitir filtrar por completed y buscar por title en el panel.

Entregables

1. Repositorio Git con código.
2. README.md con instrucciones de instalación y explicación de decisiones.