Índice

Manual Técnico — Sistema De Eventos (django)

Introducción Técnica

Este documento describe técnicamente la aplicación **Sistema de Eventos**, construída con **Python + Django** (backend) y **HTML/CSS/JS + Bootstrap** (frontend). Incluye la arquitectura, requisitos, modelado, endpoints REST, integración con APIs externas (Open-Meteo y SMTP), pruebas, seguridad y manual de instalación.

Breve descripción del sistema y sus componentes

Frontend: Plantillas Django (HTML) y assets (CSS/JS/Bootstrap). Interfaz para registro, login, gestión de eventos (crear/editar/eliminar/listar), y vistas de calendario.

Backend: Django tradicional (MVT). Maneja lógica de negocio, autenticación, APIs REST y comunicación con servicios externos.

APIs:

Internas: API REST para CRUD de eventos (endpoints JSON, soportan GET/POST/PUT/DELETE).

Externas: Open-Meteo para obtener pronóstico del clima; servicio SMTP (SMTP estándar o django.core.mail) para enviar correos de notificación.

Requisitos del Sistema

Software

- Editor: **VS Code** (recomendado)
- Lenguaje: **Python 3.10+**
- Framework: **Django 4.x** (o compatible)
- Librerías adicionales: `djangorestframework`, `requests` (para llamadas a Open-Meteo), `python-dotenv` (opcional para variables de entorno)
- Navegador moderno (Chrome, Firefox, Edge)
- Sistema de control de versiones: **git**

Hardware (mínimo para desarrollo)

- CPU: cualquier CPU moderna (2 cores OK)
- RAM: 4 GB mínimo (8 GB recomendado)
- Disco: 2 GB libres (dependiendo del proyecto)

Arquitectura del Proyecto

Estructura de carpetas (ejemplo)

```
eventos_Django/
├─ .env
├─ manage.py
├─ requirements.txt
├─ README.md
├─ eventos_app/            # settings del proyecto
│   ├─ __init__.py
│   ├─ settings.py
│   ├─ urls.py
│   └─ wsgi.py
├─ eventos/       # app principal
│   ├─ migrations/
│   ├─ templates/
│   ├─ static/
│   ├─ models.py
│   ├─ views.py
│   ├─ serializers.py
│   ├─ urls.py
│   └─ tests.py
└─ db.sqlite3
```

Patrón MVT (Model-View-Template)

- **Model**: define la estructura de datos (modelos Django → tablas DB).
- **View**: función o clase que procesa peticiones y devuelve respuestas (HTML o JSON).
- **Template**: archivos HTML que renderizan el contenido al usuario.

Django separa la capa de datos (Model), la lógica de presentación (View) y la representación (Template).

Tecnologías Utilizadas

- **Backend:** Python + Django
- **Frontend:** HTML, CSS, JavaScript, Bootstrap
- **Base de datos:** SQLite (para desarrollo). Se puede cambiar a PostgreSQL en producción.
- **APIs externas:** Open-Meteo (clima), SMTP (envío de correos)

Modelado de Datos

Modelos clave — `Eventos`

```
# eventos/models.py
from django.db import models
from django.contrib.auth.models import User
```

```python
class Evento(models.Model):
    TIPO_PRIVACIDAD = [("public", "Público"), ("private", "Privado")]

    titulo = models.CharField(max_length=200)
    descripcion = models.TextField(blank=True)
    fecha = models.DateTimeField()
    ubicacion = models.CharField(max_length=255, blank=True)
    creado_por = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='eventos')
    privado = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    def __str__(self):


      return f"{self.titulo} ({self.fecha_inicio.date()})"
```

Relación entre usuarios y eventos

- `User` (Django) tiene relación *uno a muchos* con `Evento` mediante `creado_por`.
- Si se requiere compartir eventos entre usuarios se puede agregar una relación ManyToMany llamada `invitados`.

Endpoints REST (API)

A continuación un listado de endpoints típicos (asumiendo `rest_framework` y `eventos/urls.py`):

- `GET /api/eventos/` — Listar eventos (filtrado por usuario, rango de fechas, público/privado).
- `POST /api/eventos/` — Crear evento.
- `GET /api/eventos/{id}/` — Recuperar detalle de un evento.
- `PUT /api/eventos/{id}/` — Actualizar evento.
- `DELETE /api/eventos/{id}/` — Eliminar evento.

SERIALIZADOR

```python
# eventos/serializers.py
from rest_framework import serializers
from .models import Evento

class EventoSerializer(serializers.ModelSerializer):
    class Meta:
        model = Evento
        fields = '__all__'
        read_only_fields = ('created_at','updated_at','creado_por')
```

VISTAS (VIEWSETS)

```python
# eventos/views.py
from rest_framework import viewsets, permissions
from .models import Evento
from .serializers import EventoSerializer

class EventoViewSet(viewsets.ModelViewSet):
    queryset = Evento.objects.all()
    serializer_class = EventoSerializer
    permission_classes = [permissions.IsAuthenticated]

    def perform_create(self, serializer):
        serializer.save(creado_por=self.request.user)

    def get_queryset(self):
        user = self.request.user
        return Evento.objects.filter(models.Q(privado=False) |
models.Q(creado_por=user))
```

RUTAS

```python
# eventos_app/urls.py
from django.urls import path, include
from rest_framework import routers
from eventos_app.views import EventoViewSet

router = routers.DefaultRouter()
router.register(r'eventos', EventoViewSet)

urlpatterns = [
    path('api/', include(router.urls)),
    path('', include('eventos_app.urls')),
]
```

USO (CURL)

- Listar:

```
curl -H "Authorization: Token <TOKEN>" http://localhost:8000/api/eventos/
```

- Crear:

```
curl -X POST -H "Content-Type: application/json" -H "Authorization: Token
<TOKEN>" \
  -d '{"titulo":"Reunión","fecha_inicio":"2025-10-
10T10:00:00Z","fecha_fin":"2025-10-10T11:00:00Z"}' \
  http://localhost:8000/api/eventos/
```

Integración De Apis Externas

OPEN-METEO (CLIMA)

Flujo

Cuando el usuario crea un evento con `ubicacion` (lat,long o ciudad) se llama a Open-Meteo para predecir el clima en la fecha del evento.

Los datos se muestran en la vista del evento y se pueden guardar como metadato.

LLAMADA (REQUESTS)

```python
import requests

def obtener_clima(lat, lon, fecha_iso):
    url = 'https://api.open-meteo.com/v1/forecast'
    params = {
        'latitude': lat,
        'longitude': lon,
        'daily':
'temperature_2m_max,temperature_2m_min,precipitation_sum',
        'start_date': fecha_iso.split('T')[0],
        'end_date': fecha_iso.split('T')[0],
        'timezone': 'UTC'
    }
    r = requests.get(url, params=params, timeout=10)
    r.raise_for_status()
    return r.json()
```

Notas: Open-Meteo no requiere API key para niveles básicos. Ajustar parámetros según la doc.

SMTP (envío de correos)

Configuración (settings.py)

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = os.getenv('jfortiz@tes.edu.ec')
EMAIL_HOST_PASSWORD = os.getenv('xxxxxxxxxxx')
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

ENVÍO DE NOTIFICACIÓN AL CREAR EVENTO

```
from django.core.mail import send_mail

def notificar_creacion_evento(evento):
    subject = f'Nuevo evento: {evento.titulo}'
    message = f'Has creado un evento para {evento.fecha_inicio}.'
    send_mail(subject, message, None, [evento.creado_por.email])
```

CAPTURAS DE CÓDIGO RELEVANTES

(Se incluyeron fragmentos claves: `models.py`, `serializers.py`, `views.py`, `urls.py`, `obtener_clima()` y settings de SMTP.)

Pruebas Realizadas

Pruebas Funcionales Del Crud

- Test manual en frontend: crear → listar → editar → eliminar.
- Test automático con `django.test.TestCase` y `APIClient` de `rest_framework`.

Ejemplo de test básico:

```python
# eventos/url.py
from django.test import TestCase
from django.contrib.auth.models import User
from rest_framework.test import APIClient
from .models import Evento

class EventoCRUDTest(TestCase):
    def setUp(self):
        self.user = User.objects.create_user('test', 't@example.com',
'pass')
        self.client = APIClient()
        self.client.force_authenticate(user=self.user)

    def test_create_evento(self):
        data = {
            'titulo': 'Prueba',
            'fecha_inicio': '2025-10-15T10:00:00Z',
            'fecha_fin': '2025-10-15T11:00:00Z'
        }
        r = self.client.post('/api/eventos/', data, format='json')
        self.assertEqual(r.status_code, 201)
        self.assertEqual(Evento.objects.count(), 1)
```

VALIDACIÓN DE APIS Y AUTENTICACIÓN

Se probó acceso anónimo → debe ser rechazado (401).

Se verificó que usuarios sólo vean eventos públicos o propios.

Seguridad

Autenticación

- Uso de `rest_framework` con `TokenAuthentication` o `JWT` (recomendado para SPA).
- Endpoints protegidos: todos los endpoints CRUD requieren autenticación.
- Validaciones y protección de datos
- Validar datos entrantes en `serializers` (longitudes, fechas coherentes: `fecha_inicio < fecha_fin`).
- Escapar y sanitizar texto mostrado en templates.
- Usar HTTPS en producción y variables de entorno para credenciales.
- Limitar intentos de login (rate-limit) con `django-axes` o similar.

Manual Técnico De Instalación Y Uso

1) Clonar el repositorio

```
git clone https://github.com/narutojohn/eventos_django.git
```

2) Activar entorno virtual (Linux/Mac)

```
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

En Windows (PowerShell)

```
python -m venv venv
venv\Scripts\Activate.ps1
pip install -r requirements.txt
```

3) Variables de entorno

- Crear `.env` con `SECRET_KEY`, `EMAIL_USER`, `EMAIL_PASS`, etc. O configurar en settings.

4) Migraciones y runserver

```
python manage.py migrate
python manage.py createsuperuser  # opcional
python manage.py runserver
```

Acceder en `http://localhost:8000/`.

5) Cómo usar la app (flujo)

- **Registro**: crear cuenta desde `/accounts/register/` o con admin.
- **Login**: `/accounts/login/`.
- **Crear evento**: formulario → agregar título, fechas, ubicacion → guardar.
- **Listar/Editar/Eliminar**: interfaz en `/eventos/`.

Conclusiones Técnicas

- Lo aprendido: implementación de APIs REST con Django REST Framework, integración de servicios externos (Open-Meteo y SMTP), mejores prácticas de seguridad y testing.

- Resultados: CRUD funcional, autenticación protegida, integración básica de clima y notificaciones por correo.

- Mejoras futuras: migrar a PostgreSQL para producción, añadir JWT, mejorar UI con SPA (React/Vue), añadir pruebas de integración y despliegue automático (CI/CD).