Blog con Roles (Django + DRF + React)

0) Qué vamos a lograr

"Haremos un blog con roles:

- **Anónimo**: ve solo **títulos**.
- **User (registrado)**: ve el **detalle**.
- **Premium**: ve además una **sección premium** del post.
- **Admin**: **crea/edita/borra** posts.

Cada **Post**:

- pertenece a **1 Categoría (1–N)**
- puede tener **varias Subcategorías (N–M)**
- puede tener **varios Hashtags (N–M)**."

🅰️ PARTE A — BACKEND (Django + DRF)

1) Crear proyecto, app y entorno

```
mkdir blog-roles && cd blog-roles
python -m venv venv
# Windows: venv\Scripts\activate
# macOS/Linux:
source venv/bin/activate

pip install django djangorestframework django-filter python-decouple
djangorestframework-simplejwt django-cors-headers faker
```

```
django-admin startproject blogproj .
python manage.py startapp core
```

2) Crear archivo `.env` (variables de entorno)

En la raíz del proyecto (junto a `manage.py`) crea `.env`:

```
# .env
DEBUG=True
SECRET_KEY=super-secreto-cambia-esto
ALLOWED_HOSTS=127.0.0.1,localhost

# Base de datos: por defecto usaremos SQLite. Si quieres Postgres:
# DATABASE_URL=postgres://user:pass@localhost:5432/blogdb

# CORS (para poder llamar al API desde React)
CORS_ALLOW_ORIGINS=http://localhost:3000

# JWT (opcional ajustar tiempos)
ACCESS_TOKEN_LIFETIME_MIN=15
REFRESH_TOKEN_LIFETIME_DAYS=7
```

> "Guardamos aquí datos sensibles y configurables. Nunca los subas a Git."

3) Configurar `settings.py` completo (con .env)

Edita `blogproj/settings.py`:

```
import os
from pathlib import Path
from decouple import config, Csv
from datetime import timedelta

BASE_DIR = Path(__file__).resolve().parent.parent

# === ENV ===
DEBUG = config("DEBUG", default=False, cast=bool)
SECRET_KEY = config("SECRET_KEY")
```

```
ALLOWED_HOSTS = config("ALLOWED_HOSTS",
default="127.0.0.1,localhost", cast=Csv())

INSTALLED_APPS = [

"django.contrib.admin","django.contrib.auth","django.contrib.content
types",

"django.contrib.sessions","django.contrib.messages","django.contrib.
staticfiles",
    # Terceros
    "rest_framework","django_filters","corsheaders",
    # App propia
    "core",
]

MIDDLEWARE = [
    "corsheaders.middleware.CorsMiddleware",  # CORS primero
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]

ROOT_URLCONF = "blogproj.urls"

TEMPLATES = [
    {
        "BACKEND":
"django.template.backends.django.DjangoTemplates",
        "DIRS": [BASE_DIR / "templates"],   # Plantillas globales
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
```

```
"django.contrib.messages.context_processors.messages",
            ],
        },
    },
]

WSGI_APPLICATION = "blogproj.wsgi.application"

# === Base de datos ===
# Por simplicidad SQLite; si quieres Postgres, usa dj-database-url o
similar.
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }
}

# === Idioma y zona horaria ===
LANGUAGE_CODE = "es"
TIME_ZONE = "Europe/Madrid"
USE_I18N = True
USE_TZ = True

# === Archivos estáticos ===
STATIC_URL = "static/"
STATIC_ROOT = BASE_DIR / "staticfiles"
STATICFILES_DIRS = [BASE_DIR / "static"]

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

# === DRF ===
REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": (
        "rest_framework_simplejwt.authentication.JWTAuthentication",
    ),
    "DEFAULT_FILTER_BACKENDS": (
        "django_filters.rest_framework.DjangoFilterBackend",
        "rest_framework.filters.SearchFilter",
        "rest_framework.filters.OrderingFilter",
```

```
    ),
    "DEFAULT_PAGINATION_CLASS":
"rest_framework.pagination.PageNumberPagination",
    "PAGE_SIZE": 10,
}

# === JWT ===
ACCESS_MIN = config("ACCESS_TOKEN_LIFETIME_MIN", default=15,
cast=int)
REFRESH_DAYS = config("REFRESH_TOKEN_LIFETIME_DAYS", default=7,
cast=int)
SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=ACCESS_MIN),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=REFRESH_DAYS),
}

# === CORS ===
CORS_ALLOW_CREDENTIALS = True
CORS_ALLOWED_ORIGINS = config("CORS_ALLOW_ORIGINS",
default="http://localhost:3000", cast=Csv())
```

“Aquí leemos variables del `.env`, configuramos DRF, JWT y CORS para React.”

4) Modelos (POO: abstracción, herencia, encapsulamiento, polimorfismo)

`core/models.py`:

```
from django.db import models
from django.contrib.auth.models import User

# Base abstracta (abstracción + herencia)
class BaseEntidad(models.Model):
    titulo = models.CharField(max_length=200)
    fecha_creacion = models.DateTimeField(auto_now_add=True)
    fecha_modificacion = models.DateTimeField(auto_now=True)

    class Meta:
        abstract = True

    # Polimorfismo: cada hija puede redefinir
```

```
    def resumen(self):
        return f"{self.titulo} ({self.fecha_creacion.date()})"

class ProfileUser(models.Model):
    ROLE_CHOICES =
(("user","Usuario"),("premium","Premium"),("admin","Admin"))
    user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name="profile")
    role = models.CharField(max_length=10, choices=ROLE_CHOICES,
default="user")
    def __str__(self): return f"{self.user.username} [{self.role}]"

class Category(BaseEntidad):
    def __str__(self): return self.titulo

class Subcategory(BaseEntidad):
    def __str__(self): return self.titulo

class Hashtag(BaseEntidad):
    def __str__(self): return f"#{self.titulo.lower()}"

class Post(BaseEntidad):
    categoria = models.ForeignKey(Category,
on_delete=models.PROTECT, related_name="posts")
    subcategorias = models.ManyToManyField(Subcategory, blank=True,
related_name="posts")
    hashtags = models.ManyToManyField(Hashtag, blank=True,
related_name="posts")
    autor = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="posts")

    # Encapsulamiento
    _contenido = models.TextField(db_column="contenido")
    _contenido_premium =
models.TextField(db_column="contenido_premium", blank=True,
default="")

    @property
    def contenido(self): return self._contenido
    @contenido.setter
    def contenido(self, v): self._contenido = (v or "").strip()
```

```
    @property
    def contenido_premium(self): return self._contenido_premium
    @contenido_premium.setter
    def contenido_premium(self, v): self._contenido_premium = (v or
"").strip()

    # Polimorfismo
    def resumen(self): return f"{self.titulo} —
{self.categoria.titulo}"
```

Aplica migraciones y crea superusuario:

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

5) Admin

`core/admin.py`:

```
from django.contrib import admin
from .models import ProfileUser, Category, Subcategory, Hashtag,
Post

@admin.register(Category, Subcategory, Hashtag)
class TaxAdmin(admin.ModelAdmin):
    list_display = ("titulo","fecha_creacion")
    search_fields = ("titulo",)
    list_filter = ("fecha_creacion",)

@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ("titulo","categoria","autor","fecha_creacion")
    search_fields = ("titulo","_contenido")
    list_filter = ("categoria","fecha_creacion")

admin.site.register(ProfileUser)
```

6) Formularios (para CBV admin)

core/forms.py:

```
from django import forms
from .models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields =
("titulo","categoria","subcategorias","hashtags","contenido","conten
ido_premium")
```

7) Permisos por rol (ayudantes)

core/permissions.py:

```
from django.core.exceptions import PermissionDenied

def is_admin(user):
    return user.is_authenticated and
getattr(getattr(user,"profile",None),"role",None)=="admin"
```

8) Vistas — FBV (lista pública) y CBV (detalle/CRUD)

core/views.py:

```
from django.shortcuts import render
from django.core.exceptions import PermissionDenied
from django.views.generic import DetailView, CreateView, UpdateView,
DeleteView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from .models import Post, Category
from .forms import PostForm
from .permissions import is_admin

# FBV: listado público (anónimo ve títulos)
def post_list(request):
```

```
    posts =
Post.objects.select_related("categoria","autor").all().order_by("-fe
cha_creacion")
    return render(request, "core/post_list.html", {"posts": posts,
"categorias": Category.objects.all()})

# Detalle: requiere login (user/premium/admin)
class PostDetailView(LoginRequiredMixin, DetailView):
    model = Post
    template_name = "core/post_detail.html"

# Solo admin puede crear/editar/borrar
class AdminRequiredMixin(LoginRequiredMixin):
    def dispatch(self, request, *args, **kwargs):
        if not is_admin(request.user):
            raise PermissionDenied
        return super().dispatch(request, *args, **kwargs)

class PostCreateView(AdminRequiredMixin, CreateView):
    model = Post
    form_class = PostForm
    template_name = "core/post_form.html"
    success_url = reverse_lazy("post_list")
    def form_valid(self, form):
        form.instance.autor = self.request.user
        return super().form_valid(form)

class PostUpdateView(AdminRequiredMixin, UpdateView):
    model = Post
    form_class = PostForm
    template_name = "core/post_form.html"
    success_url = reverse_lazy("post_list")

class PostDeleteView(AdminRequiredMixin, DeleteView):
    model = Post
    success_url = reverse_lazy("post_list")
```

9) Plantillas (mínimas)

Crea `templates/base.html`, `templates/core/post_list.html`, `templates/core/post_detail.html`, `templates/core/post_form.html`:

base.html

```
<!doctype html>
<html lang="es">
<head><meta charset="utf-8"><title>{% block title %}Blog{% endblock
%}</title></head>
<body>
<header>
  {% if user.is_authenticated %}
    Hola, {{ user.username }} ({{ user.profile.role }}) | <a
href="/admin/">Admin</a>
  {% else %}
    <a href="/admin/login/">Entrar</a>
  {% endif %}
  <hr>
</header>
<main>{% block content %}{% endblock %}</main>
</body>
</html>
```

post_list.html

```
{% extends 'base.html' %}
{% block content %}
<h2>Posts</h2>
<ul>
  {% for p in posts %}
    <li><a href="{% url 'post_detail' p.id %}">{{ p.titulo
}}</a></li>
  {% endfor %}
</ul>
{% endblock %}
```

post_detail.html

```
{% extends 'base.html' %}
{% block content %}
<h2>{{ object.titulo }}</h2>
<p>{{ object.contenido }}</p>
```

```
{% if user.profile.role == 'premium' or user.profile.role == 'admin'
%}
  <hr><h4>Sección Premium</h4>
  <p>{{ object.contenido_premium }}</p>
{% endif %}
{% endblock %}
```

post_form.html

```
{% extends 'base.html' %}
{% block content %}
<h2>Post</h2>
<form method="post">{% csrf_token %}
  {{ form.as_p }}
  <button>Guardar</button>
</form>
{% endblock %}
```

10) Serializers y API (DRF + JWT)

`core/serializers.py`:

```
from rest_framework import serializers
from .models import Post

class PostListSerializer(serializers.ModelSerializer):
    categoria = serializers.StringRelatedField()
    subcategorias = serializers.StringRelatedField(many=True)
    hashtags = serializers.StringRelatedField(many=True)
    class Meta:
        model = Post
        fields =
("id","titulo","categoria","subcategorias","hashtags","fecha_creacio
n")

class PostDetailSerializer(PostListSerializer):
    class Meta(PostListSerializer.Meta):
        fields = PostListSerializer.Meta.fields +
("contenido","contenido_premium",)
```

core/api_views.py:

```
from rest_framework import generics, permissions, filters
from django_filters.rest_framework import DjangoFilterBackend
from .models import Post
from .serializers import PostListSerializer, PostDetailSerializer

class PostListCreateAPI(generics.ListCreateAPIView):
    queryset =
Post.objects.select_related("categoria","autor").all().order_by("-fe
cha_creacion")
    serializer_class = PostListSerializer
    filter_backends = [DjangoFilterBackend, filters.SearchFilter,
filters.OrderingFilter]
    filterset_fields = {"categoria":["exact"]}
    search_fields = ["titulo","categoria__titulo"]
    ordering_fields = ["fecha_creacion","titulo"]

    def get_permissions(self):
        if self.request.method == "POST":
            return [permissions.IsAuthenticated()]
        return [permissions.AllowAny()]

    def perform_create(self, serializer):
        user = self.request.user
        role = getattr(getattr(user,"profile",None),"role",None)
        if role != "admin":
            from rest_framework.exceptions import PermissionDenied
            raise PermissionDenied("Solo admin puede crear posts.")
        serializer.save(autor=user)

class PostRetrieveAPI(generics.RetrieveAPIView):
    queryset = Post.objects.all()
    def get_serializer_class(self):
        if self.request.user.is_authenticated:
            return PostDetailSerializer
        return PostListSerializer
```

11) URLs

blogproj/urls.py:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("core.urls")),
    path("api/", include("core.api_urls")),
]
```

core/urls.py:

```
from django.urls import path
from .views import post_list, PostDetailView, PostCreateView,
PostUpdateView, PostDeleteView

urlpatterns = [
    path("", post_list, name="post_list"),
    path("post/<int:pk>/", PostDetailView.as_view(),
name="post_detail"),
    path("post/new/", PostCreateView.as_view(), name="post_create"),
    path("post/<int:pk>/edit/", PostUpdateView.as_view(),
name="post_edit"),
    path("post/<int:pk>/delete/", PostDeleteView.as_view(),
name="post_delete"),
]
```

core/api_urls.py:

```
from django.urls import path
from rest_framework_simplejwt.views import TokenObtainPairView,
TokenRefreshView
from .api_views import PostListCreateAPI, PostRetrieveAPI

urlpatterns = [
    path("token/", TokenObtainPairView.as_view(),
name="token_obtain_pair"),
    path("token/refresh/", TokenRefreshView.as_view(),
name="token_refresh"),
    path("posts/", PostListCreateAPI.as_view(), name="api_posts"),
    path("posts/<int:pk>/", PostRetrieveAPI.as_view(),
name="api_post_detail"),
```

```
]
```

“JWT listo: `/api/token/` y `/api/token/refresh/`.”

12) Seed de datos (management command)

Estructura:

`core/management/commands/seed_blog.py`

Código:

```
from django.core.management.base import BaseCommand
from django.contrib.auth.models import User
from core.models import ProfileUser, Category, Subcategory, Hashtag,
Post
from faker import Faker
import random

class Command(BaseCommand):
    help = "Genera datos de ejemplo para Blog (usuarios, categorías,
posts)."

    def handle(self, *args, **options):
        fake = Faker("es_ES")

        # Usuarios + perfiles
        users =
[("admin","admin123","admin"),("user","user123","user"),("premium","
premium123","premium")]
        for u, p, r in users:
            usr, created = User.objects.get_or_create(username=u)
            if created:
                usr.set_password(p); usr.save()
            ProfileUser.objects.get_or_create(user=usr,
defaults={"role": r})

        # Taxonomía
        categorias = [Category.objects.create(titulo=fake.word())
for _ in range(3)]
```

```
        subcats = [Subcategory.objects.create(titulo=fake.word())
for _ in range(4)]
        tags = [Hashtag.objects.create(titulo=fake.word()) for _ in
range(6)]

        # Posts
        admin = User.objects.get(username="admin")
        for _ in range(10):
            post = Post.objects.create(
                titulo=fake.sentence(nb_words=5),
                categoria=random.choice(categorias),
                autor=admin,
                _contenido=fake.paragraph(nb_sentences=4),
                _contenido_premium=fake.text(180),
            )
            post.subcategorias.set(random.sample(subcats, k=2))
            post.hashtags.set(random.sample(tags, k=3))
            post.save()

        self.stdout.write(self.style.SUCCESS("Datos de ejemplo
creados ✅"))
```

Ejecuta:

```
python manage.py seed_blog
```

13) Probar el backend

```
python manage.py runserver
```

- Web:
    - `http://127.0.0.1:8000/` → títulos
    - Detalle requiere login (usa /admin para loguearte)
- API:
    - `POST http://127.0.0.1:8000/api/token/` con `{"username":"user","password":"user123"}`

- Usa el `access` en `Authorization: Bearer <token>` para llamar a `/api/posts/` y `/api/posts/<id>/`.
