REFERENCIA RÁPIDA DE DJANGO

Modelos, vistas, plantillas, ORM, formularios, admin, auth

Configuración del proyecto

Crear proyecto y app

```
pip install django
django-admin startproject mysite
cd mysite
python manage.py startapp blog
```

Comandos comunes

`runserver`	Iniciar servidor de desarrollo en el puerto 8000
`makemigrations`	Generar archivos de migración desde cambios en modelos
`migrate`	Aplicar migraciones a la base de datos
`createsuperuser`	Crear superusuario de administración
`shell`	Shell interactivo de Python con Django
`test`	Ejecutar suite de pruebas

Estructura del proyecto

`manage.py`	Punto de entrada de la CLI
`settings.py`	Configuración del proyecto
`urls.py`	Configuración de URLs raíz
`wsgi.py / _asgi.py`	Puntos de entrada del servidor
`apps/models.py`	Modelos de base de datos
`apps/views.py`	Manejadores de solicitudes

Modelos

Definir un modelo

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=200)
    body = models.TextField()
    created = models.DateTimeField(auto_now_add=True)
    published = models.BooleanField(default=False)
```

Tipos de campo

`CharField(max_length=N)`	Texto corto (max_length requerido)
`TextField()`	Texto largo (sin límite)
`IntegerField()`	Valor entero
`FloatField()`	Número de punto flotante
`BooleanField()`	Verdadero / Falso
`DateTimeField()`	Fecha y hora
`EmailField()`	Email con validación
`FileField(upload_to='')`	Carga de archivos

Relaciones

```
author = models.ForeignKey(
    User, on_delete=models.CASCADE
)
tags = models.ManyToManyField(Tag, blank=True)
profile = models.OneToOneField(User, on_delete=models.CASCADE)
```

Meta y métodos

```
class Meta:
    ordering = ['-created']
    verbose_name_plural = 'posts'

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

Vistas

Vista basada en función

```
from django.shortcuts import render, get_object_or_404

def post_list(request):
    posts = Post.objects.filter(published=True)
    return render(request, 'blog/list.html', {'posts': posts})
```

Vistas basadas en clases

```
from django.views.generic import ListView, DetailView

class PostListView(ListView):
    model = Post
    template_name = 'blog/list.html'
    context_object_name = 'posts'
    paginate_by = 10
```

CBVs comunes

`ListView`	Mostrar lista de objetos
`DetailView`	Mostrar un solo objeto
`CreateView`	Formulario para crear objeto
`UpdateView`	Formulario para editar objeto
`DeleteView`	Confirmar y eliminar objeto
`TemplateView`	Renderizar plantilla (sin modelo)

Respuesta JSON

```
from django.http import JsonResponse

def api_posts(request):
    data = list(Post.objects.values('id', 'title'))
    return JsonResponse(data, safe=False)
```

Plantillas

Sintaxis de plantilla

```
{{ variable }}
{{ post.title|truncatewords:30 }}
{% if user.is_authenticated %}
  <p>Welcome, {{ user.username }}!</p>
{% endif %}
```

Bucles y condiciones

```
{% for post in posts %}
  <h2>{{ post.title }}</h2>
  {% if forloop.last %}<hr>{% endif %}
{% empty %}
  <p>No posts yet.</p>
{% endfor %}
```

Herencia de plantillas

```
{# base.html #}
<html>
<body>{% block content %}{% endblock %}</body>
</html>

{# child.html #}
{% extends "base.html" %}
{% block content %}<h1>Hello</h1>{% endblock %}
```

Filtros comunes

`|date:"Y-m-d"`	Formatear fecha
`|default:"N/A"`	Valor por defecto para vacíos
`|length`	Contar elementos en lista
`|truncatewords:N`	Limitar a N palabras
`|safe`	Marcar como HTML seguro (sin escapado)
`|slugify`	Cadena en minúsculas apta para URL

URLs

Patrones de URL

```
from django.urls import path, include

urlpatterns = [
    path('', views.index, name='index'),
    path('post/<int:pk>/', views.detail, name='detail'),
    path('blog/', include('blog.urls')),
]
```

Conversores de path

`<int:pk>`	Entero (ej. 42)
`<str:slug>`	Cadena sin barras
`<slug:slug>`	Slug (letras, números, guiones)
`<uuid:id>`	Formato UUID
`<path:rest>`	Ruta completa incluyendo barras

URLs inversas

```
from django.urls import reverse
url = reverse('detail', kwargs={'pk': 1})
# En plantillas: {% url 'detail' pk=post.pk %}
```

Formularios

ModelForm

```
from django import forms

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'body', 'published']
```

Procesar formulario en vista

```
def create_post(request):
    form = PostForm(request.POST or None)
    if form.is_valid():
        post = form.save(commit=False)
        post.author = request.user
        post.save()
        return redirect('detail', pk=post.pk)
    return render(request, 'blog/form.html', {'form': form})
```

Formulario en plantilla

```
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Save</button>
</form>
```

Validación

```
def clean_title(self):
    title = self.cleaned_data['title']
    if len(title) < 5:
        raise forms.ValidationError("Title too short.")
    return title
```

Admin

Registrar modelo

```
from django.contrib import admin
from .models import Post

@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ['title', 'author', 'created', 'published']
    list_filter = ['published', 'created']
    search_fields = ['title', 'body']
```

Opciones de admin

`list_display`	Columnas en la vista de lista
`list_filter`	Opciones de filtro en la barra lateral
`search_fields`	Campos buscables
`prepopulated_fields`	Autocompletar (ej. slug desde título)
`readonly_fields`	No editables en el admin
`ordering`	Orden de clasificación por defecto

Consultas ORM

Consultas básicas

```
Post.objects.all()                # todos los registros
Post.objects.get(pk=1)            # único (lanza error si falta)
Post.objects.filter(published=True) # queryset
Post.objects.exclude(draft=True)  # excluir coincidencias
Post.objects.count()              # total de registros
```

Lookups de campo

`field__exact`	Coincidencia exacta (por defecto)
`field__icontains`	Contiene sin distinguir mayúsculas
`field__gt / __lt`	Mayor / menor que
`field__gte / __lte`	Mayor/menor o igual que
`field__in=[1,2,3]`	Valor en la lista
`field__isnull=True`	Es NULL
`field__startswith`	Empieza con cadena
`field__range=(a,b)`	Entre a y b inclusive

Encadenamiento y agregación

```
from django.db.models import Q, Count, Avg

Post.objects.filter(
    Q(title__icontains='django') | Q(body__icontains='django')
).order_by('-created')[:10]

Post.objects.aggregate(avg_views=Avg('views'))
```

Crear, actualizar, eliminar

```
post = Post.objects.create(title='New', body='...')
post.title = 'Updated'
post.save()
Post.objects.filter(draft=True).update(published=False)
post.delete()
```

Autenticación

Login / Logout

```
from django.contrib.auth import authenticate, login, logout

user = authenticate(request, username='admin', password='pw')
if user is not None:
    login(request, user)
```

Proteger vistas

```
from django.contrib.auth.decorators import login_required

@login_required
def dashboard(request):
    return render(request, 'dashboard.html')
```

URLs de auth

```
# urls.py
path('accounts/', include('django.contrib.auth.urls'))
# Provee: login, logout, password_change, password_reset
```

Auth en plantilla

```
{% if user.is_authenticated %}
  <p>Hi, {{ user.username }}</p>
  <a href="{% url 'logout' %}">Logout</a>
{% else %}
  <a href="{% url 'login' %}">Login</a>
{% endif %}
```

Configuración

Ajustes clave

`DEBUG`	`True` para desarrollo, `False` para producción
`ALLOWED_HOSTS`	Lista de nombres de host válidos
`SECRET_KEY`	Clave de firma criptográfica (mantener en secreto)
`DATABASES`	Motor, nombre, host y credenciales de BD
`INSTALLED_APPS`	Lista de apps registradas
`STATIC_URL`	Prefijo de URL para archivos estáticos
`MEDIA_URL / MEDIA_ROOT`	Rutas para archivos subidos por usuarios

Configuración de base de datos

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydb',
        'USER': 'dbuser',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Archivos estáticos

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / 'static']
# En plantillas: {% load static %}
# <link href="{% static 'css/style.css' %}">
```