Riferimento Rapido Django

Modelli, view, template, ORM, form, admin, autenticazione

Configurazione Progetto

Creare Progetto e App

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

Comandi Comuni

`runserver`	Avvia server di sviluppo sulla porta 8000
`makemigrations`	Genera file di migrazione dalle modifiche al modello
`migrate`	Applica le migrazioni al database
`createsuperuser`	Crea superutente admin
`shell`	Shell Python interattiva con Django
`test`	Esegui la suite di test

Struttura del Progetto

`manage.py`	Punto di ingresso CLI
`settings.py`	Configurazione del progetto
`urls.py`	Configurazione URL root
`wsgi.py / asgi.py`	Punti di ingresso server
`apps/models.py`	Modelli database
`apps/views.py`	Gestori delle richieste

Modelli

Definire un Modello

```
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)
```

Tipi di Campo

`CharField(max_length=N)`	Testo breve (max_length obbligatorio)
`TextField()`	Testo lungo (nessun limite)
`IntegerField()`	Valore intero
`FloatField()`	Numero in virgola mobile
`BooleanField()`	True / False
`DateTimeField()`	Data e ora
`EmailField()`	Email con validazione
`FileField(upload_to='')`	Upload file

Relazioni

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

Meta e Metodi

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

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

View

View Basata su Funzione

```
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})
```

View Basate su Classi

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

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

CBV Comuni

`ListView`	Mostra lista di oggetti
`DetailView`	Mostra singolo oggetto
`CreateView`	Form per creare oggetto
`UpdateView`	Form per modificare oggetto
`DeleteView`	Conferma ed elimina oggetto
`TemplateView`	Renderizza un template (nessun modello)

Risposta JSON

```
from django.http import JsonResponse

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

Template

Sintassi Template

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

Cicli e Condizioni

```
{% for post in posts %}
  <h2>{{ post.title }}</h2>
  {% if forloop.last %}<hr>{% endif %}
{% empty %}
  <p>Nessun post ancora.</p>
{% endfor %}
```

Ereditarietà Template

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

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

Filtri Comuni

`|date:"Y-m-d"`	Formatta la data
`|default:"N/A"`	Fallback per valori vuoti
`|length`	Conta elementi nella lista
`|truncatewords:N`	Limita a N parole
`|safe`	Marca come HTML sicuro (nessun escaping)
`|slugify`	Stringa URL-safe in minuscolo

URL

Pattern 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')),
]
```

Convertitori Path

`<int:pk>`	Intero (es. 42)
`<str:slug>`	Stringa senza slash
`<slug:slug>`	Slug (lettere, numeri, trattini)
`<uuid:id>`	Formato UUID
`<path:rest>`	Percorso completo inclusi gli slash

Reverse URL

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

Form

Model Form

```
from django import forms

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

Elaborare il Form nella View

```
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})
```

Form nel Template

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

Validazione

```
def clean_title(self):
    title = self.cleaned_data['title']
    if len(title) < 5:
        raise forms.ValidationError("Titolo troppo corto.")
    return title
```

Admin

Registrare il Modello

```
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']
```

Riferimento Rapido Django

Opzioni Admin

`list_display`	Colonne nella vista lista
`list_filter`	Opzioni filtro nella barra laterale
`search_fields`	Campi ricercabili
`prepopulated_fields`	Compilazione automatica (es. slug dal titolo)
`readonly_fields`	Non modificabili nell'admin
`ordering`	Ordinamento predefinito

Query ORM

Query Base

```
Post.objects.all()               # tutti i record
Post.objects.get(pk=1)           # singolo (errore se mancante)
Post.objects.filter(published=True) # queryset
Post.objects.exclude(draft=True)  # esclude corrispondenze
Post.objects.count()             # conteggio totale
```

Lookup dei Campi

`field__exact`	Corrispondenza esatta (predefinita)
`field__icontains`	Contiene (case-insensitive)
`field__gt / __lt`	Maggiore / minore di
`field__gte / __lte`	Maggiore/minore o uguale a
`field__in=[1,2,3]`	Valore nella lista
`field__isnull=True`	È NULL
`field__startswith`	Inizia con la stringa
`field__range=(a,b)`	Tra a e b inclusi

Concatenamento e Aggregazione

```
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'))
```

Crea, Aggiorna, Elimina

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

Autenticazione

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)
```

Proteggere le View

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

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

URL Auth

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

Auth nel Template

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

Impostazioni

Impostazioni Principali

`DEBUG`	`True` per sviluppo, `False` per produzione
`ALLOWED_HOSTS`	Lista degli hostname validi
`SECRET_KEY`	Chiave di firma crittografica (mantieni segreta)
`DATABASES`	Engine DB, nome, host, credenziali
`INSTALLED_APPS`	Lista delle app registrate
`STATIC_URL`	Prefisso URL per i file statici
`MEDIA_URL / MEDIA_ROOT`	Percorsi per i file caricati dagli utenti

Configurazione Database

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

File Statici

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