**Documento de Projeto Django — Guia Completo, Didático e Prático**

Autor: M365 Copilot | Para: Silvio Luiz Puzzovio

Este documento explica com profundidade a anatomia de um projeto Django, desde conceito (projeto vs. apps) até código (models, views, templates), segurança, testes e deploy. Inclui trechos de código hipercomentados e \*checklists\* práticos.

# Sumário

* 1. Visão geral: Projeto (condomínio) x App (prédios)
* 2. Estrutura mínima gerada pelo startproject
* 3. Ambiente e primeiros passos (venv, startproject, startapp, runserver, migrate)
* 4. settings.py em detalhes (INSTALLED\_APPS, MIDDLEWARE, TEMPLATES, DATABASES, STATIC/MEDIA, i18n)
* 5. URLs e roteamento (path, include, namespaces)
* 6. Views (FBV x CBV) e ciclo de request/response
* 7. Templates (DTL), herança e context processors
* 8. Models & ORM (relacionamentos, QuerySets, otimizações)
* 9. Forms & ModelForms (validação e segurança CSRF)
* 10. Admin (ModelAdmin, list\_display, filtros, busca)
* 11. Autenticação e autorização (LoginRequiredMixin, permissões)
* 12. Arquivos estáticos e mídia (collectstatic, produção)
* 13. Internacionalização, localização e fuso horário
* 14. Segurança (checklist, ALLOWED\_HOSTS, CSRF, XSS, cookies seguros)
* 15. Logging e configurações por ambiente (.env, settings modularizados)
* 16. Assíncrono, tasks e tempo real (ASGI, async views, Channels, Celery)
* 17. Cache e performance (select\_related/prefetch\_related, cache backends)
* 18. Testes (TestCase, Client, fixtures) e qualidade
* 19. Deploy (WSGI/ASGI, Gunicorn/Uvicorn, Nginx, banco, static/media)
* 20. Estrutura sugerida para projetos grandes
* 21. Comandos úteis do manage.py (cheatsheet)
* Anexo A — App exemplo completo (CRUD) com comentários linha a linha
* Anexo B — Checklist de produção

# 1) Visão geral: Projeto (condomínio) x App (prédios)

• Projeto = condomínio: guarda regras gerais (settings), portaria (manage.py), mapa de endereços (urls) e pontos de entrada (wsgi/asgi).  
• App = prédio funcional: encapsula um domínio (usuários, empresas, indicadores, relatórios). Pode ser reutilizável em outros projetos.  
• Dentro do app: models (banco), views (lógica por rota), templates (HTML), urls (endereços), admin (painel), tests e arquivos estáticos. (Ref.: docs “Applications” e tutorial oficial).

# 2) Estrutura mínima gerada pelo startproject

O comando cria um pacote do projeto com: manage.py, settings.py, urls.py, asgi.py, wsgi.py. (Ref.: tutorial oficial).

|  |
| --- |
| django-admin startproject mysite  mysite/ ├─ manage.py └─ mysite/  ├─ \_\_init\_\_.py  ├─ settings.py  ├─ urls.py  ├─ asgi.py  └─ wsgi.py |

# 3) Ambiente e primeiros passos

Crie venv, instale Django, gere projeto e app, rode migrações e servidor.

|  |
| --- |
| python -m venv .venv # Windows: .\.venv\Scripts\Activate.ps1 # macOS/Linux: source .venv/bin/activate python -m pip install --upgrade pip pip install django  # criar projeto e app django-admin startproject config . python manage.py startapp core  # ativar app em INSTALLED\_APPS e rodar python manage.py migrate python manage.py runserver |

# 4) settings.py em detalhes

Principais blocos:

* INSTALLED\_APPS: registra apps (core e contrib).
* MIDDLEWARE: pipeline de processamento de request/response.
* TEMPLATES: backend, dirs e context processors.
* DATABASES: engine (sqlite3, postgresql, etc.).
* LANGUAGE\_CODE/TIME\_ZONE: i18n/l10n e TZ.
* STATIC\_URL/STATICFILES\_DIRS/STATIC\_ROOT: estáticos.
* MEDIA\_URL/MEDIA\_ROOT: uploads.
* ALLOWED\_HOSTS, SECRET\_KEY, DEBUG: segurança e ambiente.

|  |
| --- |
| # config/settings.py (exemplo didático) INSTALLED\_APPS = [  'django.contrib.admin','django.contrib.auth','django.contrib.contenttypes',  'django.contrib.sessions','django.contrib.messages','django.contrib.staticfiles',  'core', ] MIDDLEWARE = [  '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', ] TEMPLATES = [{  'BACKEND':'django.template.backends.django.DjangoTemplates',  'DIRS':[BASE\_DIR/'templates'], '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',  ]} }] DATABASES = {'default': {  'ENGINE':'django.db.backends.sqlite3', 'NAME': BASE\_DIR/'db.sqlite3'} } LANGUAGE\_CODE = 'pt-br' TIME\_ZONE = 'America/Sao\_Paulo' STATIC\_URL = 'static/' MEDIA\_URL = 'media/' |

# 5) URLs e roteamento

O URL dispatcher mapeia padrões para views (funções ou classes). Use path(), include(), nomes e namespaces. (Ref.: URL dispatcher).

|  |
| --- |
| # config/urls.py from django.contrib import admin from django.urls import path, include urlpatterns = [  path('admin/', admin.site.urls),  path('', include('core.urls', namespace='core')), ]  # core/urls.py from django.urls import path from . import views app\_name = 'core' urlpatterns = [  path('', views.HomeView.as\_view(), name='home'),  path('empresa/<int:pk>/', views.CompanyDetailView.as\_view(), name='company\_detail'), ] |

# 6) Views (FBV x CBV) e ciclo de request/response

Funções (FBV) são simples; Classes (CBV) permitem reuso via herança/mixins. Genéricas aceleram CRUD.

|  |
| --- |
| # FBV simples from django.http import HttpResponse  def ping(request):  return HttpResponse('pong')  # CBV genéricas (lista, detalhe, criar, atualizar, deletar) from django.views.generic import ListView, DetailView, CreateView, UpdateView, DeleteView from django.urls import reverse\_lazy from .models import Company  class CompanyListView(ListView):  model = Company  paginate\_by = 10  class CompanyDetailView(DetailView):  model = Company  class CompanyCreateView(CreateView):  model = Company  fields = ['name','ticker']  success\_url = reverse\_lazy('core:home')  class CompanyUpdateView(UpdateView):  model = Company  fields = ['name','ticker']  success\_url = reverse\_lazy('core:home')  class CompanyDeleteView(DeleteView):  model = Company  success\_url = reverse\_lazy('core:home') |

# 7) Templates (DTL), herança e context processors

O Django Template Language (DTL) isola apresentação de lógica. Use herança de templates e tags/filters. (Ref.: Templates e Template Language).

|  |
| --- |
| <!-- templates/base.html --> {% load static %} <!doctype html> <html lang="pt-br">  <head>  <meta charset="utf-8">  <title>{% block title %}Minha App{% endblock %}</title>  <link rel="stylesheet" href="{% static 'css/base.css' %}">  </head>  <body>  <main class="container">{% block content %}{% endblock %}</main>  </body> </html>  <!-- templates/core/company\_list.html --> {% extends 'base.html' %} {% block title %}Empresas{% endblock %} {% block content %}  <h1>Empresas</h1>  <ul>  {% for obj in object\_list %}  <li><a href="{% url 'core:company\_detail' obj.pk %}">{{ obj.name }}</a></li>  {% empty %}  <li>Nenhuma empresa.</li>  {% endfor %}  </ul> {% endblock %} |

# 8) Models & ORM (relacionamentos, QuerySets, otimizações)

Defina dados como classes Python. Use a API de QuerySets. Otimize com select\_related (FK/OneToOne) e prefetch\_related (M2M/reverse). (Ref.: QuerySet API).

|  |
| --- |
| # models.py from django.db import models  class Sector(models.Model):  name = models.CharField(max\_length=80)  class Company(models.Model):  name = models.CharField(max\_length=120)  ticker = models.CharField(max\_length=20, unique=True)  sector = models.ForeignKey(Sector, on\_delete=models.PROTECT, related\_name='companies')  # consultas qs = Company.objects.select\_related('sector').all() # JOIN no banco (ótimo p/ FK) # Para reverse/M2M: Sector.objects.prefetch\_related('companies').all() |

# 9) Forms & ModelForms (validação e segurança CSRF)

ModelForm evita repetição ao mapear campos do model para o form; valida e salva com form.save(). (Ref.: ModelForms).

|  |
| --- |
| # forms.py from django import forms from .models import Company  class CompanyForm(forms.ModelForm):  class Meta:  model = Company  fields = ['name','ticker','sector']  # view usando ModelForm from django.views.generic import CreateView class CompanyCreateView(CreateView):  form\_class = CompanyForm  template\_name = 'core/company\_form.html' |

# 10) Admin (ModelAdmin, list\_display, filtros, busca)

Customize colunas, filtros, busca e inlines para produtividade no /admin. (Ref.: Admin docs).

|  |
| --- |
| # admin.py from django.contrib import admin from .models import Company, Sector  @admin.register(Company) class CompanyAdmin(admin.ModelAdmin):  list\_display = ('id','name','ticker','sector')  list\_filter = ('sector',)  search\_fields = ('name','ticker','sector\_\_name')  admin.site.register(Sector) |

# 11) Autenticação e autorização (LoginRequiredMixin, permissões)

Use mixins para proteger views e permissões por usuário/grupo. (Ref.: Auth e mixins).

|  |
| --- |
| from django.contrib.auth.mixins import LoginRequiredMixin, PermissionRequiredMixin  class SecureListView(LoginRequiredMixin, PermissionRequiredMixin, ListView):  permission\_required = 'core.view\_company'  model = Company |

# 12) Arquivos estáticos e mídia (collectstatic, produção)

Durante dev, runserver serve estáticos. Em produção, use collectstatic e sirva via servidor web/CDN. (Ref.: staticfiles).

|  |
| --- |
| # settings.py (produção) STATIC\_URL = 'static/' STATIC\_ROOT = BASE\_DIR/'staticfiles' # destino do collectstatic MEDIA\_URL = 'media/' MEDIA\_ROOT = BASE\_DIR/'media'  # coletar python manage.py collectstatic |

# 13) Internacionalização, localização e fuso horário

Configure LANGUAGE\_CODE, TIME\_ZONE e USE\_TZ. Ative tradução e formatação local quando necessário.

# 14) Segurança (checklist)

* DEBUG = False em produção; ALLOWED\_HOSTS configurado.
* SECRET\_KEY fora do código (variáveis de ambiente).
* CSRF habilitado; cookies seguros: SESSION\_COOKIE\_SECURE/CSRF\_COOKIE\_SECURE.
* SECURE\_SSL\_REDIRECT=True atrás de HTTPS.
* X-Frame-Options (Clickjacking) e outras recomendações do checklist.

# 15) Logging e configurações por ambiente (.env, settings modularizados)

Separe settings base/dev/prod. Carregue segredos de variáveis de ambiente. Ex.: django-environ.

|  |
| --- |
| # config/settings/base.py -> usado por dev.py e prod.py # config/settings/dev.py -> DEBUG=True # config/settings/prod.py -> DEBUG=False, ALLOWED\_HOSTS, banco/segurança  # .env (exemplo) DJANGO\_SECRET\_KEY='...' DJANGO\_DEBUG='0' DJANGO\_ALLOWED\_HOSTS='meusite.com,.meusite.com' |

# 16) Assíncrono, tasks e tempo real

Django suporta ASGI e views async; para websockets use Django Channels; para tarefas agendadas/assíncronas use Celery + broker (Redis).

# 17) Cache e performance

* Use select\_related (FK/OneToOne) e prefetch\_related (M2M/reverse) para evitar N+1.
* Configure cache (locmem, memcached, Redis) para páginas, fragmentos ou low-level API.
* Mensure com Django Debug Toolbar (terceiro) e logging de queries.

# 18) Testes (TestCase, Client, fixtures) e qualidade

Escreva testes de models, views e forms. Use TestCase, Client e fixtures. Integre com CI. (Ref.: tutorial e docs de testes).

|  |
| --- |
| from django.test import TestCase from .models import Company  class CompanyModelTests(TestCase):  def test\_str(self):  obj = Company.objects.create(name='Acme', ticker='ACME3')  self.assertEqual(str(obj), 'Acme') |

# 19) Deploy (WSGI/ASGI, Gunicorn/Uvicorn, Nginx, banco, static/media)

* Empacote app e rode via WSGI (Gunicorn) ou ASGI (Uvicorn) atrás de Nginx.
* Aplique migrações e collectstatic na pipeline.
* Configure Postgres, variáveis de ambiente e observabilidade (logs/monitoramento).

# 20) Estrutura sugerida para projetos grandes

|  |
| --- |
| project/ ├─ manage.py ├─ config/ # pacote do projeto (settings/urls/asgi/wsgi) │ ├─ settings/ │ │ ├─ base.py │ │ ├─ dev.py │ │ └─ prod.py │ ├─ urls.py │ ├─ asgi.py │ └─ wsgi.py ├─ apps/ │ ├─ accounts/ │ ├─ companies/ │ ├─ indicators/ │ └─ reports/ ├─ templates/ ├─ static/ ├─ media/ ├─ requirements/ │ ├─ base.txt │ ├─ dev.txt │ └─ prod.txt └─ docs/ |

# 21) Comandos úteis do manage.py (cheatsheet)

* runserver, migrate, makemigrations, shell, createsuperuser, collectstatic, test, dumpdata/loaddata
* startapp/startproject, showmigrations/sqlmigrate, check

# Anexo A — App exemplo completo (CRUD) com comentários

Exemplo mínimo de CRUD usando CBVs e ModelForm (comentários linha a linha).

|  |
| --- |
| # models.py from django.db import models class Company(models.Model):  name = models.CharField(max\_length=120) # nome da empresa  ticker = models.CharField(max\_length=20) # código de negociação  def \_\_str\_\_(self): # string amigável no admin/listas  return self.name  # forms.py from django import forms from .models import Company class CompanyForm(forms.ModelForm):  class Meta:  model = Company  fields = ['name','ticker'] # evita expor campos não desejados  # views.py from django.views.generic import ListView, CreateView, UpdateView, DeleteView from django.urls import reverse\_lazy from .models import Company from .forms import CompanyForm  class CompanyListView(ListView):  model = Company  template\_name = 'core/company\_list.html' # herda base.html  paginate\_by = 10  class CompanyCreateView(CreateView):  form\_class = CompanyForm  template\_name = 'core/company\_form.html'  success\_url = reverse\_lazy('core:company\_list')  class CompanyUpdateView(UpdateView):  model = Company  form\_class = CompanyForm  template\_name = 'core/company\_form.html'  success\_url = reverse\_lazy('core:company\_list')  class CompanyDeleteView(DeleteView):  model = Company  template\_name = 'core/company\_confirm\_delete.html'  success\_url = reverse\_lazy('core:company\_list')  # urls.py from django.urls import path from .views import CompanyListView, CompanyCreateView, CompanyUpdateView, CompanyDeleteView app\_name = 'core' urlpatterns = [  path('companies/', CompanyListView.as\_view(), name='company\_list'),  path('companies/new/', CompanyCreateView.as\_view(), name='company\_new'),  path('companies/<int:pk>/edit/', CompanyUpdateView.as\_view(), name='company\_edit'),  path('companies/<int:pk>/delete/', CompanyDeleteView.as\_view(), name='company\_delete'), ]  # templates/core/company\_list.html (trecho) {% extends 'base.html' %} {% block content %} <h1>Empresas</h1> <p><a href="{% url 'core:company\_new' %}" class="btn btn-primary">Nova</a></p> <ul>  {% for obj in object\_list %}  <li>{{ obj.name }} ({{ obj.ticker }}) —  <a href="{% url 'core:company\_edit' obj.pk %}">Editar</a> |  <a href="{% url 'core:company\_delete' obj.pk %}">Excluir</a></li>  {% empty %}  <li>Nenhum registro.</li>  {% endfor %} </ul> {% endblock %} |

# Anexo B — Checklist de produção

* DEBUG=False; ALLOWED\_HOSTS definido; SECRET\_KEY seguro (env).
* Banco de produção (PostgreSQL), migrations aplicadas.
* collectstatic executado e estáticos servidos por Nginx/CDN/WhiteNoise.
* SECURE\_SSL\_REDIRECT, SESSION\_COOKIE\_SECURE, CSRF\_COOKIE\_SECURE, HSTS (se aplicável).
* Logs configurados; monitoramento; backups e migrations versionadas.

# Referências (documentação recomendada)

* Django Docs (tutorial, models, views, templates, settings, URLs, admin, auth, staticfiles, caching, testing).
* MDN (tutoriais Django) e guias práticos de CBVs e forms.