Scripit para montar a estrutura de um aplicativo em Django

1) Estrutura de pastas recomendada

```
projeto_loja/
├─ venv/                              # (opcional) ambiente virtual
├─ manage.py
├─ projeto_loja/                      # pacote do projeto (core)
│   ├─ __init__.py
│   ├─ asgi.py
│   ├─ settings/
│   │   ├─ __init__.py
│   │   ├─ base.py                    # configs comuns
│   │   ├─ dev.py                     # overrides p/ desenvolvimento
│   │   └─ prod.py                    # overrides p/ produção
│   ├─ urls.py                        # roteamento raiz
│   └─ wsgi.py
├─ apps/
│   ├─ clientes/
│   │   ├─ __init__.py
│   │   ├─ admin.py
│   │   ├─ apps.py
│   │   ├─ migrations/…
│   │   ├─ models.py
│   │   ├─ urls.py
│   │   ├─ views.py
│   │   ├─ forms.py
│   │   └─ templates/clientes/…       # templates deste app
│   ├─ produtos/
│   │   ├─ … (mesma ideia)
│   └─ vendas/
│       ├─ … (mesma ideia)
├─ templates/                         # templates “globais” (base.html,
etc.)
│   └─ base.html
└─ static/                            # css/js/imagens globais
    ├─ css/
    ├─ js/
    └─ img/
```

Por que separar `settings` em `base/dev/prod`?
Fica profissional: você herda de `base.py` e troca apenas o necessário por ambiente (debug, banco, caches, allowed hosts, etc.).

2) Comandos iniciais (Windows)

```
# 1) criar e ativar venv
python -m venv venv
.\venv\Scripts\activate

# 2) instalar django
pip install django
```

```
# 3) criar projeto
django-admin startproject projeto_loja .

# 4) transformar settings em pacote
mkdir projeto_loja\settings
move projeto_loja\settings.py projeto_loja\settings\base.py
echo. > projeto_loja\settings\__init__.py

# 5) criar "dev.py" e "prod.py"
# dev.py (aponta para base e liga DEBUG)
```

projeto_loja/settings/dev.py

```
from .base import *
DEBUG = True
ALLOWED_HOSTS = []
```

projeto_loja/settings/prod.py

```
from .base import *
DEBUG = False
ALLOWED_HOSTS = ["sua-url.com.br"]
# configurar banco, staticfiles, segurança etc.
```

Para rodar usando o settings correto:

```
# Desenvolvimento
set DJANGO_SETTINGS_MODULE=projeto_loja.settings.dev
python manage.py migrate
python manage.py createsuperuser
python manage.py runserver
```

3) Criar os apps funcionais

```
python manage.py startapp clientes apps\clientes
python manage.py startapp produtos apps\produtos
python manage.py startapp vendas   apps\vendas
```

No **`projeto_loja/settings/base.py`**, registre os apps e os diretórios de templates/static:

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    # apps do projeto
    "apps.clientes",
    "apps.produtos",
    "apps.vendas",
]

TEMPLATES = [
    {
```

```
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [BASE_DIR / "templates"],  # templates globais
        "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",
        ]},
    }
]

STATIC_URL = "/static/"
STATICFILES_DIRS = [BASE_DIR / "static"]
```

4) Models mínimos para CRUD + Admin

apps/clientes/models.py

```
from django.db import models

class Cliente(models.Model):
    nome = models.CharField(max_length=120)
    email = models.EmailField(unique=True, blank=True, null=True)
    telefone = models.CharField(max_length=20, blank=True)
    criado_em = models.DateTimeField(auto_now_add=True)

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

apps/produtos/models.py

```
from django.db import models

class Produto(models.Model):
    nome = models.CharField(max_length=120)
    sku = models.CharField(max_length=40, unique=True)
    preco = models.DecimalField(max_digits=10, decimal_places=2)
    estoque = models.PositiveIntegerField(default=0)
    ativo = models.BooleanField(default=True)

    def __str__(self):
        return f"{self.nome} ({self.sku})"
```

apps/vendas/models.py

```
from django.db import models
from django.conf import settings
from apps.clientes.models import Cliente
from apps.produtos.models import Produto

class Venda(models.Model):
    cliente = models.ForeignKey(Cliente, on_delete=models.PROTECT)
    data = models.DateTimeField(auto_now_add=True)
    vendedor = models.ForeignKey(
        settings.AUTH_USER_MODEL, on_delete=models.PROTECT,
blank=True, null=True
    )
```

```
    def total(self):
        return sum(item.subtotal() for item in self.itens.all())

    def __str__(self):
        return f"Venda #{self.pk} - {self.cliente}"

class ItemVenda(models.Model):
    venda = models.ForeignKey(Venda, related_name="itens",
on_delete=models.CASCADE)
    produto = models.ForeignKey(Produto, on_delete=models.PROTECT)
    quantidade = models.PositiveIntegerField(default=1)
    preco_unit = models.DecimalField(max_digits=10, decimal_places=2)

    def subtotal(self):
        return self.quantidade * self.preco_unit
```

Registrar no Admin (CRUD imediato no admin):

```
# apps/clientes/admin.py
from django.contrib import admin
from .models import Cliente
@admin.register(Cliente)
class ClienteAdmin(admin.ModelAdmin):
    list_display = ("nome", "email", "telefone", "criado_em")
    search_fields = ("nome", "email")

# apps/produtos/admin.py
from django.contrib import admin
from .models import Produto
@admin.register(Produto)
class ProdutoAdmin(admin.ModelAdmin):
    list_display = ("nome", "sku", "preco", "estoque", "ativo")
    search_fields = ("nome", "sku")
    list_filter = ("ativo",)

# apps/vendas/admin.py
from django.contrib import admin
from .models import Venda, ItemVenda

class ItemVendaInline(admin.TabularInline):
    model = ItemVenda
    extra = 1

@admin.register(Venda)
class VendaAdmin(admin.ModelAdmin):
    inlines = [ItemVendaInline]
    list_display = ("id", "cliente", "data")
    search_fields = ("cliente__nome",)
```

5) URLs (roteamento)

projeto_loja/urls.py

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

urlpatterns = [
```

```
    path("admin/", admin.site.urls),
    path("clientes/", include("apps.clientes.urls")),
    path("produtos/", include("apps.produtos.urls")),
    path("vendas/", include("apps.vendas.urls")),
    path("", include("apps.clientes.urls")),  # página inicial
provisória
]
```

apps/clientes/urls.py (e repetir o padrão em `produtos` e `vendas`)

```
from django.urls import path
from . import views

app_name = "clientes"
urlpatterns = [
    path("", views.ClienteList.as_view(), name="lista"),
    path("novo/", views.ClienteCreate.as_view(), name="criar"),
    path("<int:pk>/editar/", views.ClienteUpdate.as_view(),
name="editar"),
    path("<int:pk>/excluir/", views.ClienteDelete.as_view(),
name="excluir"),
]
```

6) Views com Class-Based Generic Views (CRUD rápido e padrão)

apps/clientes/views.py

```
from django.urls import reverse_lazy
from django.views.generic import ListView, CreateView, UpdateView,
DeleteView
from .models import Cliente

class ClienteList(ListView):
    model = Cliente
    paginate_by = 20
    template_name = "clientes/lista.html"

class ClienteCreate(CreateView):
    model = Cliente
    fields = ["nome", "email", "telefone"]
    success_url = reverse_lazy("clientes:lista")
    template_name = "clientes/form.html"

class ClienteUpdate(UpdateView):
    model = Cliente
    fields = ["nome", "email", "telefone"]
    success_url = reverse_lazy("clientes:lista")
    template_name = "clientes/form.html"

class ClienteDelete(DeleteView):
    model = Cliente
    success_url = reverse_lazy("clientes:lista")
    template_name = "clientes/confirma_exclusao.html"
```

Repita a mesma lógica para `produtos` e, em `vendas`, você pode criar uma `VendaCreate` com `inline formset` para itens (fica um passo além, mas é o caminho correto).

7) Templates mínimos (herdando de `base.html`)

templates/base.html (exemplo simples)

```
<!doctype html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8">
    <title>{% block title %}Loja{% endblock %}</title>
    <link rel="stylesheet" href="{% static 'css/main.css' %}">
  </head>
  <body>
    <nav>
      <a href="{% url 'clientes:lista' %}">Clientes</a> |
      <a href="{% url 'produtos:lista' %}">Produtos</a> |
      <a href="{% url 'vendas:lista' %}">Vendas</a> |
      <a href="/admin/">Admin</a>
    </nav>
    <main>
      {% block content %}{% endblock %}
    </main>
  </body>
</html>
```

apps/clientes/templates/clientes/lista.html

```
{% extends "base.html" %}
{% block title %}Clientes{% endblock %}
{% block content %}
<h1>Clientes</h1>
<a href="{% url 'clientes:criar' %}">+ Novo cliente</a>
<table>
  <tr><th>Nome</th><th>Email</th><th>Telefone</th><th>Ações</th></tr>
  {% for obj in object_list %}
  <tr>
    <td>{{ obj.nome }}</td>
    <td>{{ obj.email }}</td>
    <td>{{ obj.telefone }}</td>
    <td>
      <a href="{% url 'clientes:editar' obj.pk %}">Editar</a> |
      <a href="{% url 'clientes:excluir' obj.pk %}">Excluir</a>
    </td>
  </tr>
  {% empty %}
  <tr><td colspan="4">Nenhum cliente.</td></tr>
  {% endfor %}
</table>
{% endblock %}
```

8) Fluxo de desenvolvimento (passo a passo)

1. **Criar venv** e **instalar dependências** (`Django`).
2. **Criar projeto** e **separar settings** (`base/dev/prod`).
3. **Criar apps** em `apps/…` e registrá-los em `INSTALLED_APPS`.
4. **Modelar** `models.py` (clientes, produtos, vendas + itens).
5. **Migrations**:
6. `python manage.py makemigrations`
7. `python manage.py migrate`
8. **Admin**: registrar modelos e criar superusuário:
9. `python manage.py createsuperuser`
10. **URLs**: incluir `include()` dos apps no `urls.py` do projeto.
11. **Views**: usar **Generic Class-Based Views** para CRUD rápido.
12. **Templates**: `base.html` + páginas por app.
13. **Arquivos estáticos**: use `static/` e `{% load static %}` nos templates.
14. **Rodar**:
15. `set DJANGO_SETTINGS_MODULE=projeto_loja.settings.dev`
16. `python manage.py runserver`

9) Boas práticas rápidas

- **Nomeie apps por domínio** (clientes, produtos, vendas), e arquivos por responsabilidade (`forms.py`, `services.py`, `selectors.py` quando a lógica crescer).
- **Separar settings** por ambiente e **usar variáveis de ambiente** (ex.: `django-environ`).
- **Admin é seu aliado** no começo; depois, evolua para **views** e **templates** próprios, ou **DRF** para APIs.
- **Tests**: crie `tests/` em cada app.
- **Migrations versionadas** no Git; nunca edite migrations antigas já aplicadas em produção.