Perfeito, Silvio! Vou te explicar **toda a estrutura de um projeto Django** de forma **bem didática e minuciosa**, como se você nunca tivesse visto nada disso antes. A ideia é que você consiga **visualizar cada peça**, entender **por que ela existe**, **como se encaixa** no todo e como isso te ajuda a construir (por exemplo) o seu **site de análise fundamentalista de ações**.

💡 **Metáfora base**: pense no Django como um “condomínio” de software.

* O **projeto** é o condomínio inteiro (regras gerais, portaria, administração).
* Cada **app** é um “prédio” dentro do condomínio (com sua função específica: usuários, empresas, indicadores, relatórios, etc.).
* Dentro de cada prédio (app) você tem **modelos** (o banco de dados), **visões** (o que acontece quando alguém entra num endereço), **templates** (as páginas HTML), **URLs** (endereços que levam às visões), **admin** (o painel de controle), **testes** e mais.

**1) Estrutura típica de pastas (visão geral)**

Quando você cria um projeto com:

python -m venv .venv          # cria ambiente virtual (isolado)

source .venv/bin/activate     # (Linux/macOS) ativa o venv

# .venv\Scripts\activate      # (Windows)

pip install django            # instala o Django

django-admin startproject config .

Você normalmente terá algo assim:

seu\_projeto/

├── .venv/ ← ambiente virtual (bibliotecas instaladas)

├── manage.py ← "controle remoto" do Django (comandos)

├── config/ ← pacote do projeto (o "condomínio")

│ ├── \_\_init\_\_.py

│ ├── settings.py ← configurações globais (regras do condomínio)

│ ├── urls.py ← mapa de endereços do projeto (portaria geral)

│ ├── asgi.py ← entrada assíncrona (tempo real, websockets)

│ └── wsgi.py ← entrada síncrona (produção tradicional)

├── requirements.txt / pyproject.toml ← dependências do projeto

├── apps/ ← pasta (opcional, mas recomendada) para seus apps

│ ├── \_\_init\_\_.py

│ └── core/ ← exemplo de um app (um “prédio”)

│ ├── \_\_init\_\_.py

│ ├── admin.py ← registro no painel admin

│ ├── apps.py ← config do app

│ ├── migrations/ ← histórico de mudanças no banco

│ │ └── \_\_init\_\_.py

│ ├── models.py ← modelos (tabelas e regras de dados)

│ ├── urls.py ← endereços desse app

│ ├── views.py ← o que cada endereço faz

│ ├── forms.py ← formulários (HTML/validação)

│ ├── templates/ ← HTML do app (interface)

│ │ └── core/

│ │ └── \*.html

│ ├── static/ ← CSS/JS/imagens específicos do app

│ │ └── core/

│ │ ├── css/

│ │ ├── js/

│ │ └── img/

│ ├── tests/ ← testes automatizados

│ │ └── test\_\*.py

│ └── signals.py ← gatilhos automáticos (opcional)

├── templates/ ← templates compartilhados (base.html, etc.)

├── static/ ← estáticos compartilhados (CSS/JS/img)

└── media/ ← uploads de usuários (logo de empresas, PDFs)

✅ Dica: Colocar seus apps dentro de uma pasta apps/ mantém tudo organizado.

**2) O núcleo do projeto (pasta config/)**

**2.1 manage.py (o “controle remoto”)**

* É o **comando central** do Django. Você roda assim:
  + python manage.py runserver → sobe o servidor local.
  + python manage.py makemigrations → cria “scripts” de mudança do banco.
  + python manage.py migrate → aplica essas mudanças no banco.
  + python manage.py createsuperuser → cria um administrador do painel.
  + python manage.py startapp nome → cria um novo app.
  + python manage.py shell → abre console Python com Django carregado.

**2.2 settings.py (regras do condomínio)**

As configurações mais importantes:

* SECRET\_KEY → **senha-mestra** do projeto (nunca publique!).
* DEBUG → True no desenvolvimento, **False em produção**.
* ALLOWED\_HOSTS → domínios permitidos em produção (ex.: ["suadomena.com"]).
* INSTALLED\_APPS → lista de apps ativos (do Django e os seus).
* MIDDLEWARE → “camadas” que processam a requisição/resposta (segurança, sessões etc.).
* TEMPLATES → onde estão os templates e como renderizar.
* DATABASES → banco de dados (SQLite por padrão; Postgres/MySQL em produção).
* STATIC\_URL, STATICFILES\_DIRS, STATIC\_ROOT → arquivos estáticos (CSS/JS/imagens).
* MEDIA\_URL, MEDIA\_ROOT → arquivos enviados pelos usuários (uploads).
* LANGUAGE\_CODE, TIME\_ZONE, USE\_TZ → idioma e fuso.
* ROOT\_URLCONF → aponta para config.urls.
* WSGI\_APPLICATION/ASGI\_APPLICATION → pontos de entrada para servidores web.

🧩 **Boas práticas**: em projetos profissionais, dividimos configurações em **módulos**:

config/settings/

├── \_\_init\_\_.py

├── base.py

├── dev.py # DEBUG=True, DB local

└── prod.py # DEBUG=False, segurança, cache, etc.

E usamos variáveis de ambiente (arquivo .env) para SECRET\_KEY, credenciais do banco, etc.

**2.3 urls.py (portaria geral)**

* Centraliza **rotas** do projeto.
* Normalmente delega rotas para cada app.

Exemplo **bem comentado**:

# config/urls.py

from django.contrib import admin

from django.urls import path, include  # include = repassar rotas para apps

urlpatterns = [

    path("admin/", admin.site.urls),       # painel admin padrão

    path("", include("apps.core.urls")),   # rotas do app core na raiz do site

]

``

**2.4 wsgi.py e asgi.py**

* **WSGI**: entrada tradicional (sincrona) — usado por servidores como Gunicorn/uwsgi.
* **ASGI**: entrada moderna (assíncrona) — permite WebSockets, tarefas em tempo real.

**3) Ciclo de vida de uma requisição (o “caminho do visitante”)**

1. Navegador acessa https://seudominio.com/empresas/
2. Servidor web (Nginx/Apache) passa para o **WSGI/ASGI** do Django.
3. Django passa pelas **camadas de middleware** (segurança, sessões, etc.).
4. O roteador (URLconf) encontra a **view** correspondente ao caminho.
5. A **view** pode interagir com os **models** (banco de dados via ORM).
6. A view responde com **HTML** (template), **JSON** (API) ou outro tipo de resposta.
7. Resposta volta pelas **middlewares** e chega ao navegador.

**4) Estrutura de um App (o “prédio”)**

Vamos supor um app companies para o seu site de análise fundamentalista.

**4.1 apps.py (identidade do app)**

# apps/companies/apps.py

from django.apps import AppConfig

class CompaniesConfig(AppConfig):

    default*auto*field = "django.db.models.BigAutoField"  # tipo padrão de PK

    name = "apps.companies"  # caminho do app

    verbose\_name = "Empresas" # nome legível no admin

Depois, **adicione** "apps.companies" em INSTALLED\_APPS no settings.py.

**4.2 models.py (dados e regras de negócio)**

Exemplo **totalmente comentado**:

# apps/companies/models.py

from django.db import models

class Company(models.Model):

    """

    Representa uma empresa listada em bolsa.

    """

    # Código de negociação (ex.: PETR4)

    ticker = models.CharField(

        max*length=10,         # tamanho máximo do texto*

*unique=True,           # não pode repetir no banco*

*help*text="Código de negociação. Ex.: PETR4"

    )

    # Nome completo da empresa

    name = models.CharField(

        max*length=150,*

*help*text="Nome oficial da empresa"

    )

    # Setor (ex.: Petróleo, Bancos)

    sector = models.CharField(

        max*length=100,*

*blank=True,            # permite vazio no formulário*

*null=True              # permite nulo no banco*

*)*

*# Data de cadastro automático*

*created*at = models.DateTimeField(

        auto*now*add=True      # preenche automaticamente na criação

    )

    # Data de atualização automática

    updated*at = models.DateTimeField(*

*auto*now=True          # atualiza a cada save()

    )

    class Meta:

        ordering = ["ticker"]  # ordenação padrão quando consultamos

        verbose*name = "Empresa"*

*verbose*name*plural = "Empresas"*

*def* ***str****(self):*

*# como o objeto aparece em representações textuais (admin, shell)*

*return f"{self.ticker} - {self.name}"*

*class FinancialIndicator(models.Model):*

*"""*

*Indicadores financeiros atrelados a uma empresa, por período.*

*Ex.: P/L, ROE, Margem Líquida, etc.*

*"""*

*company = models.ForeignKey(*

*Company,                 # relação com Company*

*on*delete=models.CASCADE,# se a empresa for deletada, apaga indicadores

        related*name="indicators"*

*)*

*period = models.DateField(   # podemos representar "ano" como 31/12/AAAA*

*help*text="Data de referência do período (ex.: 31/12/2024)"

    )

    # Exemplos de indicadores (valores reais com casas decimais)

    price*earnings = models.DecimalField(max*digits=10, decimal*places=2, null=True, blank=True)*

*roe = models.DecimalField(max*digits=10, decimal*places=2, null=True, blank=True)  # Return on Equity (%)*

*net*margin = models.DecimalField(max*digits=10, decimal*places=2, null=True, blank=True)  # %

    class Meta:

        unique*together = ("company", "period")  # não repetir indicador no mesmo período*

*ordering = ["-period"]*

*verbose*name = "Indicador Financeiro"

        verbose*name*plural = "Indicadores Financeiros"

    def **str**(self):

        return f"Indicadores {self.company.ticker} em {self.period:%d/%m/%Y}"

**Fluxo após criar/alterar models**:

1. python manage.py makemigrations → cria a “migração” (um script de mudança do banco).
2. python manage.py migrate → aplica a mudança no banco de dados.

**4.3 admin.py (painel administrativo)**

# apps/companies/admin.py

from django.contrib import admin

from .models import Company, FinancialIndicator

@admin.register(Company)

class CompanyAdmin(admin.ModelAdmin):

    list*display = ("ticker", "name", "sector", "created*at")

    search*fields = ("ticker", "name", "sector")*

*list*filter = ("sector",)

@admin.register(FinancialIndicator)

class FinancialIndicatorAdmin(admin.ModelAdmin):

    list*display = ("company", "period", "price*earnings", "roe", "net*margin")*

*list*filter = ("period", "company")

    search*fields = ("company*\_ticker",)

Acesse o admin com:

python manage.py createsuperuser

python manage.py runserver

# depois, abra http://127.0.0.1:8000/admin/

**4.4 views.py (o que acontece quando alguém visita uma URL)**

Duas abordagens: **FBV** (function-based) e **CBV** (class-based).

**FBV (função)** — simples e direto:

# apps/companies/views.py

from django.shortcuts import render, get*object*or*404*

*from .models import Company*

*def company*list(request):

    """

    Lista todas as empresas.

    """

    companies = Company.objects.all()  # ORM: busca todas as empresas

    context = {"companies": companies} # dados que vão para o template

    return render(request, "companies/company*list.html", context)*

*def company*detail(request, ticker):

    """

    Detalha uma empresa pelo ticker.

    """

    company = get*object*or*404(Company, ticker=ticker.upper())*

*context = {"company": company}*

*return render(request, "companies/company*detail.html", context)

**CBV (classe)** — mais organizada para CRUDs:

# apps/companies/views.py (exemplo com CBVs)

from django.views.generic import ListView, DetailView

from .models import Company

class CompanyListView(ListView):

    model = Company

    template*name = "companies/company*list.html"

    context*object*name = "companies"  # nome usado no template

class CompanyDetailView(DetailView):

    model = Company

    template*name = "companies/company*detail.html"

    slug*field = "ticker"    # campo usado como "slug"*

*slug*url\_kwarg = "ticker"

**4.5 urls.py do app (endereços internos)**

# apps/companies/urls.py

from django.urls import path

from . import views

urlpatterns = [

    # FBV

    path("empresas/", views.company*list, name="company*list"),

    path("empresas/<str:ticker>/", views.company*detail, name="company*detail"),

    # CBV (se usar as classes acima):

    # path("empresas/", views.CompanyListView.as*view(), name="company*list"),

    # path("empresas/<str:ticker>/", views.CompanyDetailView.as*view(), name="company*detail"),

]

E no config/urls.py você inclui como vimos:

path("", include("apps.companies.urls")),

**4.6 templates/ (HTML com herança)**

**Herança de templates** evita repetição (cabeçalho/rodapé).

templates/base.html (compartilhado):

<!-- templates/base.html -->

<!DOCTYPE html>

<html lang="pt-br">

<head>

  <meta charset="UTF-8" />

  <title>{% block title %}Meu Site{% endblock %}</title>

  {% static

</head>

<body>

  <header>

    <h1>Fundamentalista</h1>

    <nav>

      <a href="{% url 'company\_list' % </header>

  <main>

    {% block content %}{% endblock %}

  </main>

  <footer>

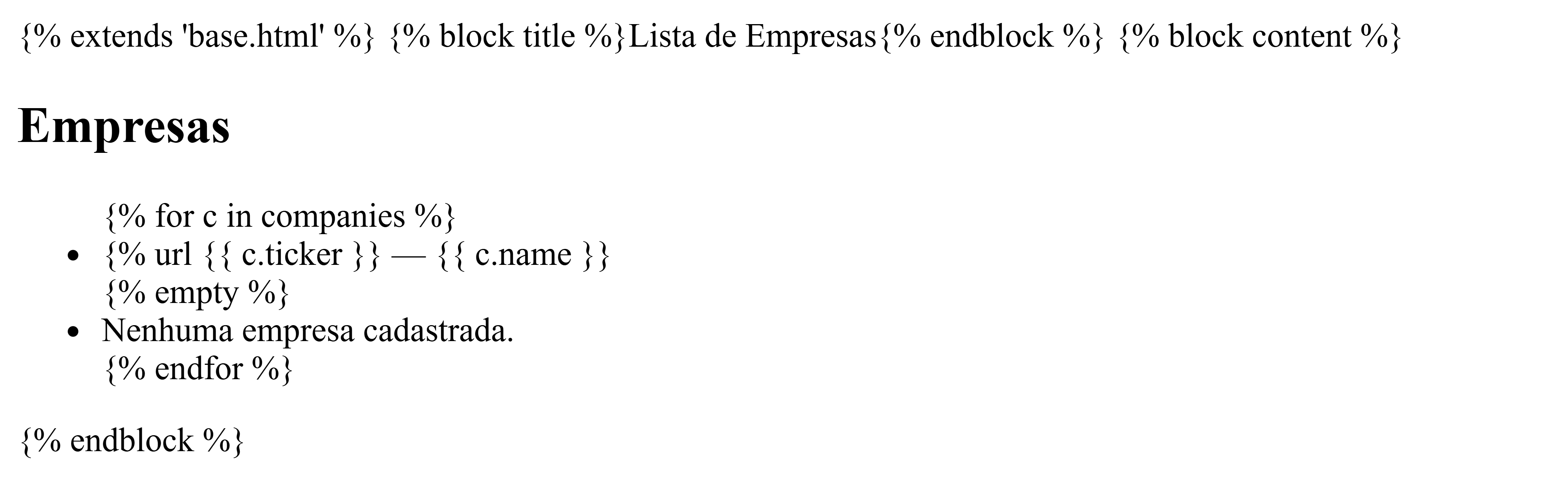
    <small>&copy; {{ now|date:"Y" }}</small>

  </footer>

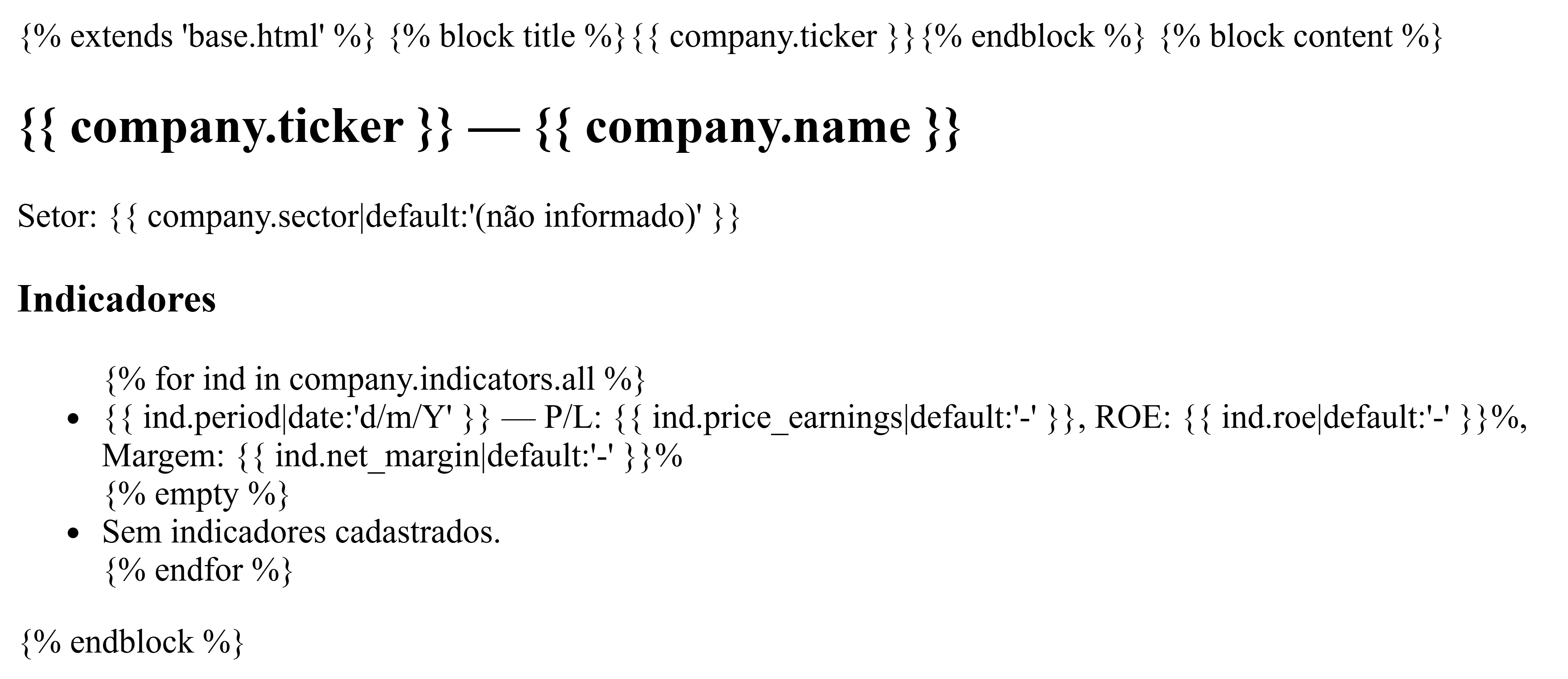
</body>

</html>

apps/companies/templates/companies/company\_list.html:



apps/companies/templates/companies/company\_detail.html:



🔧 Para usar {% static %} e {% url %} nos templates, garanta que django.contrib.staticfiles está em INSTALLED\_APPS e o APP\_DIRS está True em TEMPLATES.

**4.7 static/ (CSS/JS/imagens)**

* Em desenvolvimento, Django serve estáticos automaticamente.
* Em produção, você roda python manage.py collectstatic, que **junta tudo** em STATIC\_ROOT para o servidor web entregar.

Configuração comum no settings.py:

STATIC*URL = "/static/"*

*STATICFILES*DIRS = [BASE*DIR / "static"]  # estáticos compartilhados*

*STATIC*ROOT = BASE*DIR / "staticfiles"    # destino do collectstatic*

*MEDIA*URL = "/media/"

MEDIA*ROOT = BASE*DIR / "media"           # onde ficam os uploads

**4.8 forms.py (entrada de dados com validação)**

# apps/companies/forms.py

from django import forms

from .models import Company

class CompanyForm(forms.ModelForm):

    class Meta:

        model = Company

        fields = ["ticker", "name", "sector"]

    def clean*ticker(self):*

*# validação: garantir TICKER em maiúsculas sem espaços*

*t = self.cleaned*data["ticker"].strip().upper()

        if " " in t:

            raise forms.ValidationError("Ticker não pode conter espaços.")

        return t

**4.9 tests/ (testes automatizados)**

# apps/companies/tests/test*models.py*

*from django.test import TestCase*

*from apps.companies.models import Company*

*class CompanyModelTest(TestCase):*

*def test*str(self):

        c = Company.objects.create(ticker="TEST4", name="Teste S.A.")

        self.assertEqual(str(c), "TEST4 - Teste S.A.")

``

Dica: Rodar python manage.py test. Testes garantem que mudanças futuras não quebrem o que já funciona.

**4.10 migrations/ (histórico do banco)**

* Cada alteração nos models gera arquivos de migração.
* Nunca edite migrações antigas manualmente (salvo exceções). Crie novas.

**5) Middlewares e Context Processors (camadas “invisíveis”)**

* **Middleware**: intercepta requisições e respostas para adicionar segurança, sessões, CSRF, etc.\ Exemplo de **middleware customizado**:

# apps/core/middleware.py

import time

class RequestTimeLoggingMiddleware:

    def **init**(self, get*response):*

*self.get*response = get*response*

*def* ***call****(self, request):*

*start = time.time()*

*response = self.get*response(request)

        duration*ms = (time.time() - start) \* 1000*

*# Aqui você poderia logar o tempo*

*response["X-Request-Duration-ms"] = f"{duration*ms:.2f}"

        return response

Depois, adicione em MIDDLEWARE (em settings.py) o caminho apps.core.middleware.RequestTimeLoggingMiddleware.

* **Context Processors**: funções que **injetam variáveis** em todos os templates (ex.: o “ano atual”, dados do usuário).\ A lista fica em TEMPLATES[...]["OPTIONS"]["context\_processors"].

**6) Autenticação e Usuários (visão rápida)**

* Django já traz autenticação: login, logout, permissões, grupos.
* Você pode usar o User padrão ou **criar um CustomUser** (recomendado se precisar de campos extras).
* Templates do admin e permissões facilitam “quem pode ver/editar o quê”.

**7) Internacionalização e Fuso**

* LANGUAGE\_CODE = "pt-br"
* TIME\_ZONE = "America/Sao\_Paulo"
* USE\_I18N = True, USE\_TZ = True
* Você pode traduzir textos com gettext e gerar arquivos .po via makemessages.

**8) Ambientes: Desenvolvimento vs Produção**

* **Desenvolvimento**
  + DEBUG = True
  + Banco sqlite3 ou Postgres local
  + Servidor: python manage.py runserver
* **Produção**
  + DEBUG = False
  + ALLOWED\_HOSTS configurado
  + SECRET\_KEY e credenciais via **variáveis de ambiente**
  + Banco gerenciado (Postgres)
  + Servir **estáticos** a partir de STATIC\_ROOT (após collectstatic)
  + Usar WSGI (Gunicorn) ou ASGI (Uvicorn/Daphne)
  + HTTPS, cache, logs, backups

**9) Principais comandos do dia a dia**

* python manage.py startapp companies → cria um app
* python manage.py makemigrations → detecta mudanças nos models
* python manage.py migrate → aplica no banco
* python manage.py runserver → roda localmente
* python manage.py createsuperuser → cria admin
* python manage.py shell\_plus → (com django-extensions) shell turbinado
* python manage.py collectstatic → junta estáticos para produção
* python manage.py test → executa testes

**10) Resumo mental (mapa de onde cada coisa vive)**

* **Projeto (config/)**: configurações globais, URLs raiz, entradas WSGI/ASGI.
* **App (apps/companies/)**:
  + **models.py**: tabelas e regras de dados (ORM).
  + **migrations/**: histórico de mudanças do banco.
  + **admin.py**: painel administrativo.
  + **views.py**: lógica do que a página faz.
  + **urls.py**: caminhos deste app.
  + **templates/**: HTML (com herança e tags).
  + **static/**: CSS/JS/Imagens deste app.
  + **forms.py**: formulários com validação.
  + **tests/**: testes automatizados.
  + **signals.py**: reações a eventos (salvar, deletar etc.).
  + **apps.py**: identidade/configuração do app.

**11) Próximos passos práticos (sugeridos para você)**

1. **Criar o projeto** com a estrutura sugerida (config + apps).
2. Criar o app companies e **registrar** em INSTALLED\_APPS.
3. Implementar Company e FinancialIndicator (como acima).
4. Rodar migrações, criar superusuário e **ver no admin**.
5. Criar views/urls/templates para **listar e detalhar** empresas.
6. Adicionar **forms** para cadastrar/editar dados (se quiser).
7. Configurar STATIC e MEDIA (para logos de empresas, por exemplo).
8. Escrever **testes** básicos (garantir qualidade).
9. (Opcional) Separar settings em base/dev/prod e usar .env.

**Quer que eu gere um “mini-projeto” de exemplo?**

Se você quiser, eu posso **gerar um pacote zip** com:

* Estrutura de pastas pronta (config/, apps/companies/, templates/, static/)
* Arquivos **preenchidos** com os exemplos acima
* Comentários **linha por linha** (como você gosta) nos pontos principais

Eu não vou instalar o Django aqui (isso você faz localmente), mas o zip te entrega a **espinha dorsal** inteira já organizada. É só me dizer **o nome do projeto** e **o nome dos apps** que você quer (ex.: companies, reports, users) e eu crio!

Se preferir, posso também:

* Montar um **guia passo a passo** para você rodar tudo no seu PC (Windows/macOS/Linux).
* Explicar **classe por classe** (ListView, DetailView, CreateView, UpdateView, DeleteView) com exemplos hiper-comentados.
* Desenhar um **modelo de dados** para o seu site de análise fundamentalista (empresas, demonstrações financeiras, indicadores calculados, fontes de dados, etc.).

**Por onde você quer começar agora?** Quer o zip com a estrutura inicial comentada ou prefere que eu aprofunde mais em **Models + Migrations** ou **Views + Templates** com exemplos ainda mais detalhados?