**O que vem implementado**

**1) Comparador com Modal**

**Rota:** /fundamentus/compare/?a=ABEV3&b=AERI3

* Lista cada **indicador** uma vez, com **duas linhas de valor** (ticker A e ticker B).
* **Clique** no card abre um **modal** com: **valor + classificação** de A e B lado-a-lado, mais **fórmula, definição, faixa e referência** (puxa dos metadados do indicador).
* **Layout** e copy inspirados no seu *hfm02\_modal\_compare* (títulos, tom, interação). [[hfm02\_modal\_compare]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hfm02_modal_compare.html)

**2) Heatmap denso com Tooltip completo**

**Rota:** /fundamentus/heatmap/dense/?q=&class=

* Cartões compactos: **ticker · agrupador · fonte**, **valor** e **classificação**.
* **Hover** mostra **tooltip** detalhando **todos** os campos (fórmula, definição, faixa, referência e observação).
* Filtros por **busca** (ticker/indicador) e **classificação**.
* Em linha com seu *hm03\_dense\_tooltip* (estrutura e densidade de informação). [[hm03\_dense\_tooltip]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm03_dense_tooltip.html)

**3) Heatmap com filtro + ordenação**

**Rota:** /fundamentus/heatmap/smart/?class=&sort=label|ticker|valor&order=asc|desc

* **Chips** de classificação (inclusive “Todos”).
* **Ordenação** por **Indicador**, **Ticker** ou **Valor** (asc/desc).
* Visual respira o *hm05\_filter\_sort* (chips e controle de sort). [[hm05\_filter\_sort]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm05_filter_sort.html)

**Estrutura do ZIP**

fundamentus\_pack2/

└─ fundamentus/

├─ admin.py

├─ apps.py

├─ management/

│ └─ commands/

│ └─ load\_more\_metrics.py # novo seed desta trinca

├─ migrations/ # vazia (gere localmente)

├─ models.py # mesmo modelo (Company, IndicatorDefinition, IndicatorValue)

├─ templates/

│ └─ fundamentus/

│ ├─ hfm02\_modal\_compare.html # modal compare (novo)

│ ├─ hm03\_dense\_tooltip.html # heatmap denso + tooltip (novo)

│ ├─ hm05\_filter\_sort.html # heatmap filtro + sort (novo)

│ ├─ flip\_cards\_3d.html # placeholders p/ rotas anteriores

│ ├─ filter\_togglecard.html # placeholders p/ rotas anteriores

│ └─ quickstats.html # placeholders p/ rotas anteriores

├─ urls.py # inclui novas rotas

└─ views.py # 3 novas views + as anteriores

README.txt

**Obs.:** Mantive o **mesmo layout** visual base (tipografia, cartões, chips, grids, etc.) para todos os 3. O conteúdo e comportamento foram guiados pelos seus três HTMLs enviados (títulos, seção “Registros/Colunas”, chips, tooltip denso, modal de comparação). [[hfm02\_modal\_compare]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hfm02_modal_compare.html), [[hm03\_dense\_tooltip]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm03_dense_tooltip.html), [[hm05\_filter\_sort]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm05_filter_sort.html)

**Como rodar**

1. Adicione o app:

Python

# settings.py

INSTALLED\_APPS = [

# ...

"fundamentus",

]

Mostrar mais linhas

1. Inclua as rotas:

Python

# yourproject/urls.py

from django.urls import path, include

urlpatterns = [

path("fundamentus/", include("fundamentus.urls")),

]

``

Mostrar mais linhas

1. (Se ainda não tiver o schema) gere/migre:

Shell

python manage.py makemigrations fundamentus

python manage.py migrate

Mostrar mais linhas

1. **Seed** desta trinca (muitos indicadores ABEV3/AERI3 — ROIC, P/Cap. Giro, DY, PL, Giro Ativos, M. EBITDA, EV/EBITDA, ROE, LMD, P/Ativo, Liq. Corrente, P/VP, EV/EBIT, TAG ALONG, M. Bruta, M. Líquida, PL/Ativos, DL/EBIT, Dívida Líquida, Dívida Bruta, DL/EBITDA, ROA, VPA, P/EBIT, P/Ativo Circ. Líq., LPA, PSR):

Shell

python manage.py load\_more\_metrics

Mostrar mais linhas

1. Acesse:

* **Modal Compare:** /fundamentus/compare/?a=ABEV3&b=AERI3 [[hfm02\_modal\_compare]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hfm02_modal_compare.html)
* **Heatmap denso (tooltip):** /fundamentus/heatmap/dense/?q=ABEV3&class=Bom [[hm03\_dense\_tooltip]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm03_dense_tooltip.html)
* **Heatmap filtro+sort:** /fundamentus/heatmap/smart/?class=Ótimo&sort=valor&order=desc [[hm05\_filter\_sort]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm05_filter_sort.html)

**Notas técnicas rápidas**

* **Modal compare:** agrupa por (group, label, code) e monta um map[code] em JS para abrir o modal com **A** vs **B** sem roundtrip. Se quiser, trocamos para abrir com ?code= e recarregar server-side. [[hfm02\_modal\_compare]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hfm02_modal_compare.html)
* **Dense tooltip:** tooltip “on hover” com **todos os metadados** (fórmula, definição, faixa, referência, obs). Mantém busca e filtro de classe como no heatmap anterior. [[hm03\_dense\_tooltip]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm03_dense_tooltip.html)
* **Filtro+Sort:** class via chips; sort por label/ticker/valor; order asc/desc. Usa value\_raw para ordenar por **valor** quando disponível. [[hm05\_filter\_sort]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/hm05_filter_sort.html)

**M365 Copilot**

Chat

Agentes

Ícone

O conteúdo gerado por IA pode estar incorreto.Idea Coach

Ícone

O conteúdo gerado por IA pode estar incorreto.Prompt Coach

Todos os agentesCriar agente

Conversas

posso lhe passar um html e gostaria de gerar um co

quais o beneficios de comer amendoas e nozes todos

qual a % de diferença 12000 para 30594

melhore o texto : No último ano, apresentou evolu

melhorar o texto : Foi realizar o curso solicitad

Todas as conversas

GPT-5 Ativado

Saiba mais

Você está usando o Copilot Chat com **GPT-5** com acesso padrão. A qualidade e o desempenho podem variar dependendo da disponibilidade do serviço.[Saiba mais.](https://go.microsoft.com/fwlink/?linkid=2329839)

Hoje

**You said:**

posso lhe passar um html e gostaria de gerar um codigo python + django, onde os valores do html, devem estar em uma estrutura sdl3 do django, o layout do html, deve ser preservado, posso passar ?

**Copilot said:**

**Copilot**

**Raciocínio por 9s**

**Considerando possíveis interpretações**

**Definindo próximos passos**

**Preparando resposta detalhada**

Claro, Silvio! Pode enviar o HTML sim. 😊  
Se houver arquivos de **CSS/JS/imagens**, por favor compacte tudo em um **.zip** e envie junto — assim eu preservo o layout exatamente como está.

Antes de começar, só uma confirmação rápida sobre isto aqui: quando você diz **“estrutura sdl3 do Django”**, você quis dizer:

* **Arquitetura MTV do Django (Models, Templates, Views)**, **ou**
* **Django 3.x** com a estrutura padrão de projeto/app, **ou**
* Alguma estrutura interna específica da sua empresa chamada “sdl3”?

Se puder esclarecer, eu já gero o projeto/aplicativo no formato certo.

**O que eu vou fazer com seu HTML**

1. **Preservar o layout fielmente**
   * Mover CSS/JS/imagens para static/ e ajustar caminhos com {% load static %} + {% static '...' %}.
   * Converter seu HTML em **template Django** sem alterar a estrutura visual.
2. **Transformar valores do HTML em dados Django**
   * Identificar os trechos variáveis (textos, números, listas, cards, tabelas, etc.).
   * Mapear isso para **contexto** e, se fizer sentido, **models.py** (ORM) + **forms.py** (se houver formulários).
   * Onde houver listas/tabelas repetitivas, vou usar {% for %} e {{ ... }}.
3. **Entregar uma estrutura completa** (pronta para rodar):
4. projeto/
5. ├─ manage.py
6. ├─ projeto/ # settings, urls, wsgi/asgi
7. ├─ app/
8. │ ├─ templates/app/ # seus HTMLs convertidos p/ template
9. │ ├─ static/app/ # css, js, imagens preservados
10. │ ├─ models.py # campos com base nos valores do HTML
11. │ ├─ views.py # CBV ou FBV, você escolhe
12. │ ├─ urls.py
13. │ ├─ forms.py # se houver forms no HTML
14. │ └─ admin.py
15. └─ requirements.txt
16. **(Opcional)**
    * base.html com **template inheritance** se houver várias páginas.
    * **Fixtures** (seed) de dados para popular o banco.
    * **Tradução/i18n** se precisar.
    * **Comentários linha a linha e bem detalhados** no código (sei que você curte esse nível de detalhe 😉).

**Para eu gerar tudo na primeira passada, diga-me:**

* **Qual versão do Django** você prefere (ex.: 3.2 LTS ou 4.x).
* **Nome do projeto** e do **app** (posso sugerir padrões).
* **Banco de dados** (posso deixar **SQLite** por padrão).
* Se houver **formulários** no HTML: quais campos devem ir para **models** e quais são só de exibição?
* Você prefere **Function-Based Views** ou **Class-Based Views**?
* Quer que eu gere **CRUD completo** (listar/criar/editar/excluir) para os dados?

**Exemplo rápido do que vou gerar (só para ilustrar)**

**Template (trecho convertido):**

A interatividade nas pré-visualizações de código estará disponível em breve

**View (exemplo com contexto):**

**Python**

from django.shortcuts import render

def home(request):

contexto = {

"titulo\_pagina": "Catálogo",

"produtos": [

{"nome": "Notebook X", "descricao": "i7, 16GB", "preco": "4.999,90", "imagem": "app/img/notebook-x.png"},

{"nome": "Monitor 27\"", "descricao": "IPS, 144Hz", "preco": "1.599,90", "imagem": "app/img/monitor-27.png"},

]

}

return render(request, "app/home.html", contexto)

**Models (se quiser persistir):**

**Python**

from django.db import models

class Produto(models.Model):

nome = models.CharField(max\_length=150)

descricao = models.TextField(blank=True)

preco = models.DecimalField(max\_digits=10, decimal\_places=2)

imagem = models.CharField(max\_length=255) # ou ImageField, se quiser upload

def \_\_str\_\_(self):

return self.nome

``

Manda o **HTML (e o .zip com assets, se houver)** e me diga as preferências acima.  
A partir disso, eu **já te devolvo o projeto Django completo**, com **comentários minuciosos** no código conforme você gosta.

ex2\_cards\_grid\_from\_excel

**You said:**

segue o html

**Copilot said:**

**Copilot**

**Raciocínio por 173s**

**Gathering data**

**Convertendo para Django**

**Configurando o projeto**

**Montando o esqueleto**

**Codificando e executando...**

**Verificando a detecção de seções**

**Codificando e executando...**

**Desenvolvendo o parser**

**Codificando e executando...**

**Mapeando a estrutura HTML**

**Desenvolvendo a abordagem**

**Codificando e executando...**

**Atualizando o arquivo zip**

**Codificando e executando...**

**Organizando a apresentação**

**Extraindo e preservando**

**Codificando e executando...**

**Codificando e executando...**

**Validando resultados**

**Referenciando fontes**

**Propondo melhorias técnicamente**

Perfeito, Silvio — recebi seu HTML e já transformei tudo em um **projeto Django completo**, preservando o layout e convertendo o conteúdo em **models + templates**.  
O parser identificou **2 ativos (ABEV3, AERI3)** com **68 cards** no total, exatamente como no HTML.Favicon typeex2\_cards\_grid\_from\_excel

**✅ O que eu gerei para você**

**Download do projeto (ZIP):**  
[statuscards\_project.zip](blob:https://outlook.office.com/b2f33d04-a07b-48e7-99a0-5ff482fbddeb)

Projeto pronto para rodar: inclui **models**, **views**, **templates**, **static** (CSS extraído do seu HTML) e **fixtures** com todos os cards parseados a partir do arquivo que você enviou. Layout preservado via CSS original que estava dentro da <style> — eu movi esse CSS para static/indicadores/css/style.css e ajustei os templates para usá-lo.Favicon typeex2\_cards\_grid\_from\_excel

**📦 Estrutura gerada (resumo)**

statuscards/

├─ manage.py

├─ requirements.txt # Django 4.2 LTS por padrão

├─ statuscards/ # settings, urls, asgi, wsgi

└─ indicadores/

├─ apps.py

├─ models.py # Ativo, Indicador

├─ admin.py # admin pronto

├─ views.py # home (lista) e detalhe por ativo

├─ urls.py

├─ templates/indicadores/

│ ├─ base.html

│ ├─ home.html # lista de ativos

│ └─ ativo\_detail.html # grid de cards por ativo

├─ static/indicadores/css/

│ └─ style.css # CSS extraído do seu HTML original

└─ fixtures/

└─ indicadores.json # TODOS os 68 cards + 2 ativos

**🧠 Modelo de dados (mapeamento HTML → Django)**

Cada <article class="card"> virou um registro Indicador, ligado a um Ativo (ticker):

* **Ativo**
  + ticker: *ex.:* "ABEV3" / "AERI3" (extraído de <h2 class="section-title">Ativo: ...)Favicon typeex2\_cards\_grid\_from\_excel
* **Indicador**
  + status: texto do selo (ex.: “Crítico”, “Ótimo”) — <span class="chip ...">Favicon typeex2\_cards\_grid\_from\_excel
  + nome: título do card — <h3 class="card\_\_titulo">Favicon typeex2\_cards\_grid\_from\_excel
  + valor\_texto: valor exibido — <p class="card\_\_valor"> (mantido como string para preservar formatação, percentuais, traços etc.)Favicon typeex2\_cards\_grid\_from\_excel
  + categoria e fonte: parseadas de <p class="card\_\_nota"> (separadas pelo “• Fonte: ...”)Favicon typeex2\_cards\_grid\_from\_excel
  + faixa: texto após **Faixa:** — <p class="card\_\_faixa">Favicon typeex2\_cards\_grid\_from\_excel
  + analise: parágrafo do card — <p class="card\_\_desc">Favicon typeex2\_cards\_grid\_from\_excel
  + formula e definicao: extraídas do <details> → <strong>Fórmula:</strong> / <strong>Definição:</strong>Favicon typeex2\_cards\_grid\_from\_excel

Observação: como os formatos variam (percentuais, moeda, traço “—”), optei por **preservar o valor\_texto**. Se você quiser campos numéricos normalizados (ex.: valor\_num + unidade), eu ajusto o parser facilmente.

**🧩 Trechos principais (com comentários detalhados)**

**models.py**

**Python**

from django.db import models

*# Choices de status (iguais aos que aparecem nos chips do HTML)*

STATUS\_CHOICES = [

('Muito Crítico', 'Muito Crítico'),

('Crítico', 'Crítico'),

('Ruim', 'Ruim'),

('Moderado', 'Moderado'),

('Bom', 'Bom'),

('Muito Bom', 'Muito Bom'),

('Ótimo', 'Ótimo'),

]

class Ativo(models.Model):

*# Ticker do ativo — chave de agrupamento dos cards*

ticker = models.CharField(max\_length=12, unique=True)

class Meta:

ordering = ['ticker']

def \_\_str\_\_(self):

return self.ticker

class Indicador(models.Model):

*# FK para o ativo (1:N) — cada card pertence a um ativo*

ativo = models.ForeignKey(Ativo, on\_delete=models.CASCADE, related\_name='indicadores')

*# Selo/nota do card ("Crítico", "Bom", etc.)*

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES)

*# Título do card (ex.: "Divida bruta", "ROE", etc.)*

nome = models.CharField(max\_length=120)

*# Valor como aparece no HTML, preservando formatação*

valor\_texto = models.CharField(max\_length=50, blank=True)

*# Metadados do card*

categoria = models.CharField(max\_length=80, blank=True)

fonte = models.CharField(max\_length=120, blank=True)

faixa = models.CharField(max\_length=200, blank=True)

*# Textos explicativos*

analise = models.TextField(blank=True)

formula = models.TextField(blank=True)

definicao = models.TextField(blank=True)

*# Carimbos técnicos*

criado\_em = models.DateTimeField(auto\_now\_add=True)

atualizado\_em = models.DateTimeField(auto\_now=True)

class Meta:

ordering = ['ativo\_\_ticker', 'nome']

indexes = [models.Index(fields=['ativo', 'status'])]

def \_\_str\_\_(self):

return f"{self.ativo} · {self.nome}"

Mostrar mais linhas

**templates/indicadores/ativo\_detail.html**

**Django Template**

{% extends 'indicadores/base.html' %}

{% block title %}{{ ativo.ticker }} — Cards{% endblock %}

{% block content %}

<h2>Ativo: {{ ativo.ticker }}</h2>

<!-- Grid (o CSS é exatamente o do seu HTML, movido para static/) -->

<div class="grid">

{% for c in indicadores %}

<article class="card">

<span class="chip">{{ c.status }}</span>

<h3 class="card\_\_titulo">{{ c.nome }}</h3>

<p class="card\_\_valor">{{ c.valor\_texto }}</p>

<p class="card\_\_nota">

{{ c.categoria }} • Fonte: {{ c.fonte }}

</p>

<p class="card\_\_faixa"><strong>Faixa:</strong> {{ c.faixa }}</p>

{% if c.analise %}<p class="card\_\_desc">{{ c.analise }}</p>{% endif %}

{% if c.formula or c.definicao %}

<details style="margin-top:8px; color:var(--muted)">

<summary style="cursor:pointer;color:#c7d2fe">Detalhes</summary>

{% if c.formula %}<p><strong>Fórmula:</strong> {{ c.formula }}</p>{% endif %}

{% if c.definicao %}<p><strong>Definição:</strong> {{ c.definicao }}</p>{% endif %}

</details>

{% endif %}

</article>

{% empty %}

<p>Nenhum indicador encontrado para este ativo.</p>

{% endfor %}

</div>

{% endblock %}

``

Mostrar mais linhas

Repare que mantive **as mesmas classes CSS** do seu HTML (grid, card, chip, card\_\_titulo, card\_\_valor, etc.), e copiei seu CSS para static/indicadores/css/style.css. Assim, **o visual fica igual ao original**.Favicon typeex2\_cards\_grid\_from\_excel

**▶️ Rodando localmente**

No diretório do ZIP:

**Shell**

python -m venv .venv

# Windows: .venv\Scripts\activate

source .venv/bin/activate

pip install -r requirements.txt

python manage.py migrate

python manage.py loaddata indicadores/fixtures/indicadores.json

python manage.py runserver

``

Abra http://127.0.0.1:8000 para ver a lista de ativos e clique em cada ticker para ver o **grid de cards** correspondente.

**🔧 Observações técnicas e próximos passos**

* **Django 4.2 LTS** por padrão. Se preferir **Django 3.2 LTS** (ou **5.x**), eu ajusto requirements.txt e eventuais compatibilidades.
* **FBV** (Function-Based Views) para simplicidade. Se quiser **CBV** (ListView/DetailView) eu troco em 2 min.
* **Normalização de valores**: hoje valor\_texto preserva formatação (bom para exibir). Se quiser cálculos/ordenação, crio valor\_num (Decimal) + unidade (%, R$, etc.) e ajusto o loader.
* **Filtros/pesquisa**: dá para filtrar por status, categoria, faixa no template (ou via query params).
* **Admin** já preparado (crie um superuser e navegue): python manage.py createsuperuser.
* **Importar direto de Excel**: se você me passar o .xlsx (ou o dicionário de colunas), eu adiciono um *management command* (python manage.py importar\_indicadores caminho/arquivo.xlsx) que populará o banco com validações e conversões.

**O que usei do seu arquivo**

* Extraí **os ativos**, **todos os 68 cards** e **o CSS original** de dentro do <style> para o static/. Isso garante que o layout fique fiel ao HTML que você enviou.Favicon typeex2\_cards\_grid\_from\_excel

Se quiser, eu **aumento o nível de comentários linha a linha** (sei que você curte explicações minuciosas 😉) em **todos** os arquivos — é só me dizer quais priorizar (models, views, templates, parser/loader).  
Quer que eu **normalize os valores** (moedas/percentuais) e já entregue **filtros** por status/categoria na UI?

Fontes

Favicon type

**You said:**

se eu passar mais 8 htmls, poderia usar a mesma logica e gerar o codigo django + sql3, pois os dados são os mesmos, para compor as tabelas, manter o mesmo layout, pode ser ?

**Copilot said:**

**Copilot**

**Raciocínio por 50s**

**Respondendo à dúvida**

**Propondo plano de ação**

**Implementando instruções de importação**

**Codificando e executando...**

**Investigando erro de caminho**

**Codificando e executando...**

**Reexecutando o processo**

**Codificando e executando...**

Com certeza, Silvio — **pode mandar os 8 HTMLs** que eu uso a **mesma lógica** para importar tudo em **Django + SQLite** mantendo **o mesmo layout**. Já deixei o projeto preparado para **importar múltiplos HTMLs** de uma vez, com **deduplicação** e **upsert** (atualiza caso o indicador já exista).

**Projeto atualizado (ZIP):** [statuscards\_project.zip](blob:https://outlook.office.com/4b81f341-b658-45b2-ba30-917f65651eff)  
— agora inclui um **management command** importar\_html para processar quantos HTMLs você quiser.

**Como vai funcionar (fluxo)**

1. **Você me envia** os 8 HTMLs (idealmente em um **.zip**).
   * Se cada HTML tiver um <style> ... </style> igual ao do exemplo, **o visual fica idêntico** (eu já movo o CSS para static/ automaticamente).
2. **Eu rodo o importador** e **mantenho o layout** nos templates (mesmas classes grid, card, chip, card\_\_titulo, etc.).
3. **Os dados** (Ativo, Indicadores, status, valor, faixa, análise, fórmula, definição, etc.) são **persistidos** via ORM no **SQLite** e exibidos nos templates preservando a estrutura.

**O que já está pronto no projeto**

* **Models**
  + Ativo(ticker) e Indicador(...).
  + **Restrição de unicidade**: unique\_together = (ativo, nome) — assim, se você reimportar, o card é **atualizado** e não duplicado.
* **Templates** preservando o **mesmo layout** (CSS copiado do seu HTML para static/indicadores/css/style.css).
* **Admin** para gerenciar tudo.
* **Fixtures** (68 cards de ABEV3 e AERI3) — já incluídas para você ver funcionando.
* **Comando de importação múltipla**: python manage.py importar\_html ... (detalhes abaixo).

O ZIP acima contém tudo isso, pronto para rodar. [statuscards\_project.zip](blob:https://outlook.office.com/4b81f341-b658-45b2-ba30-917f65651eff)

**Rodando localmente**

**Shell**

python -m venv .venv

# Windows: .venv\Scripts\activate

source .venv/bin/activate

pip install -r requirements.txt

python manage.py migrate

# (Opcional) carregar os 68 cards do exemplo

python manage.py loaddata indicadores/fixtures/indicadores.json

python manage.py runserver

Acesse http://127.0.0.1:8000.

**Importando vários HTMLs (seus 8 arquivos)**

Você pode passar **arquivos individuais** ou uma **pasta**:

**Shell**

# Reimportar do zero (apenas Indicadores; Ativos são preservados)

python manage.py importar\_html /caminho/pasta\_com\_htmls --clear

# Ou passando arquivos soltos (UPsert por (ativo, nome))

python manage.py importar\_html a.html b.html c.html d.html e.html f.html g.html h.html

**Regras do importador:**

* Encontra cada **Ativo** por <h2 class="section-title">Ativo: TICKER ...</h2>.
* Dentro da <section class="grid"> ... </section>, lê cada <article class="card"> e extrai:
  + status (texto do <span class="chip">),
  + nome ( <h3 class="card\_\_titulo">),
  + valor\_texto ( <p class="card\_\_valor">),
  + categoria e fonte (de <p class="card\_\_nota">),
  + faixa ( <p class="card\_\_faixa">),
  + analise ( <p class="card\_\_desc">),
  + formula / definicao (de <details> com <strong>Fórmula:</strong> / <strong>Definição:</strong>).
* Faz **upsert** pelo par **(ativo, nome)** para evitar duplicidades em reimportações.

Tudo isso já está embutido no comando importar\_html dentro do projeto do ZIP. [statuscards\_project.zip](blob:https://outlook.office.com/4b81f341-b658-45b2-ba30-917f65651eff)

**Sobre o layout (CSS/estático)**

* Se **todos** os 8 HTMLs usam o **mesmo CSS** (estrutura idêntica), eu **centralizo** em static/indicadores/css/style.css (como já fiz) e o visual fica **idêntico** ao original.
* Se **algum** arquivo tiver **CSS diferente**, posso:
  1. **Unificar** e manter um único CSS “canônico”, **ou**
  2. Criar **templates alternativos** com CSS separados (ex.: style\_a.css, style\_b.css) e rotas específicas.

*(Se quiser a opção 2, me avise e eu já deixo os templates separados por conjunto.)*

**Próximos passos (sugestões)**

* **Normalizar valores**: hoje mantenho valor\_texto para preservar formatação. Se você quiser ordenar/filtrar por **numéricos**, crio valor\_num (Decimal) e separo unidade (%, R$, etc.).
* **Filtros na UI**: por **status**, **categoria**, **faixa** e **fonte** (via querystring ou dropdowns).
* **CBVs** e **paginação** se o número de cards crescer ainda mais.
* **Seed/Fixtures** por ativo: gerar fixtures separadas para facilitar migrações.

Se estiver ok, **me envia os 8 HTMLs** (preferencialmente em um **.zip**). Eu já **importo** e te devolvo o projeto com tudo populado — e, se quiser, acrescento **comentários detalhados linha a linha** nos arquivos que você preferir (models, views, management command, templates), do jeitinho que você curte estudar.

Fontes

Favicon type

**You said:**

vou mandar 11, que como zip ou mando 1 a 1 ?

**Copilot said:**

**Copilot**

Melhor mandar tudo em **um único arquivo ZIP** — assim eu consigo processar os 11 HTMLs de uma vez, aplicar a mesma lógica e gerar:

* **Projeto Django completo** (mantendo layout e CSS);
* **Banco SQLite** já populado com todos os indicadores;
* **Fixtures** atualizadas;
* **Comando de importação** funcionando para futuras cargas.

Se mandar um a um, eu teria que rodar o processo várias vezes, o que é menos prático.  
Então, compacta os 11 HTMLs em um .zip e me envia. Depois disso, posso:

✅ Atualizar o projeto com todos os dados;  
✅ Garantir que não haja duplicatas (uso upsert por (ativo, nome));  
✅ Te devolver o ZIP final pronto para rodar.

Quer que eu também normalize os valores (ex.: converter para número + unidade) ou mantenho tudo como texto por enquanto?

**You said:**

não ha suport, para zip

**Copilot said:**

**Copilot**

**Raciocínio por 1s**

**Oferecendo alternativas de envio**

Tranquilo! Se não dá para enviar **.zip**, podemos seguir assim:

**Opção A — Enviar um a um (mais simples)**

1. Me envie os **11 arquivos .html** individualmente, em sequência.
2. Eu vou **importando à medida que você enviar** (o projeto já tem o comando de importação preparado).
3. Ao final, eu te devolvo o **projeto Django atualizado** + **SQLite** populado e, se quiser, **fixtures** novas.

**Dica:** Se puder, mantenha um **padrão de nome** (ex.: cards\_ABEV3.html, cards\_AERI3.html, …). Facilita a auditoria depois.

**Opção B — Consolidar tudo em um único HTML**

Se seus arquivos seguem o mesmo padrão (cada um com Ativo: TICKER e uma <section class="grid">), você pode:

* **Colar** o conteúdo de todos **num único arquivo .html**, um após o outro.
* Me enviar esse único .html.  
  O importador lê **múltiplas seções de “Ativo:”** dentro do mesmo arquivo sem problemas.

**Opção C — Enviar links (se estiverem hospedados)**

Se os 11 HTMLs estiverem públicos (ou em um repositório), me passe as **URLs** e eu faço o download e importo.

**Como vou processar**

* Para cada arquivo, o importador faz **upsert** por (ativo, nome) → **sem duplicar** cards.
* O **layout** é mantido (mesmas classes e **CSS preservado** no static/).
* No final, te devolvo o **ZIP do projeto** atualizado e pronto para rodar.

Quer que eu **normalize os valores** (ex.: número + unidade) já nessa rodada, ou mantemos **valor\_texto** como está para preservar a formatação?  
Se preferir, já habilito **filtros por status/categoria** na página também.

**You said:**

vai 1 a 1

**Copilot said:**

**Copilot**

**Raciocínio por 1s**

**Fornecendo instruções de envio**

Perfeito, Silvio — vamos **1 a 1** então. 👍

**Como vamos trabalhar a cada arquivo .html**

1. **Você envia** o próximo HTML.
2. Eu **importo** e atualizo o **banco (SQLite)** + **fixtures** + **templates** mantendo o **mesmo layout**.
3. Eu te devolvo um **ZIP do projeto Django** atualizado (pronto para rodar).
4. Repetimos para o próximo arquivo até fechar os 11.

O importador faz **upsert** por (ativo, nome), então **não duplica** cards se vierem repetidos entre arquivos.  
Se algum arquivo tiver **CSS diferente**, eu preservo o visual criando a variação necessária (ou unifico se você preferir).

**O que eu preciso em cada envio**

* Apenas o **HTML bruto** (como você fez no primeiro).
* Se houver **imagens locais** referenciadas (não é comum nesse padrão), me diga para eu mover para static/.

**Sugestão de nome**: cards\_<TICKER>.html (ex.: cards\_PETR4.html). Isso facilita auditoria.

**O que eu te retorno após cada import**

* **Projeto Django** atualizado (requirements.txt, manage.py, statuscards/, indicadores/ etc.).
* **Banco db.sqlite3** já populado (se você quiser que eu inclua o arquivo do banco no ZIP, posso enviar).
* **indicadores/fixtures/indicadores.json** com todos os dados carregados até aquele momento.
* **Templates** e **CSS** preservando o layout do seu HTML.

**Rodar localmente (passo a passo)**

Após eu te devolver o ZIP (ou caso você já esteja com a versão anterior), para conferir o progresso:

**Shell**

python -m venv .venv

# Windows: .venv\Scripts\activate

source .venv/bin/activate

pip install -r requirements.txt

python manage.py migrate

python manage.py loaddata indicadores/fixtures/indicadores.json # opcional (se quiser reconstituir a partir das fixtures)

python manage.py runserver

Acesse: http://127.0.0.1:8000

Quando quiser, **manda o primeiro HTML** que eu já faço a importação inicial e te devolvo o projeto atualizado. Quer que eu já **normalize os valores** (número + unidade) ou mantemos **valor\_texto** por enquanto para preservar a formatação?

**You said:**

vou mandar de 3 em 3

**Copilot said:**

**Copilot**

Perfeito, Silvio! Pode mandar os **3 primeiros HTMLs** agora.  
Vou processar assim que receber, atualizar o projeto Django com:

* **Todos os dados importados** (sem duplicar);
* **Banco SQLite** atualizado;
* **Fixtures** completas;
* **Layout preservado** (mesmo CSS e estrutura).

Depois te devolvo o **ZIP atualizado** para você baixar e testar.  
Quando estiver pronto, manda os próximos 3, e assim por diante até os 11.

Pode enviar o primeiro lote! 🚀

ex3\_lista\_filtrosex9\_cards\_horizontais\_from\_excelexC\_mosaico\_classificacao

**You said:**

segue os 3 primeiros

**Copilot said:**

**Copilot**

**Raciocínio por 199s**

**Analisando arquivos HTML**

**Desenvolvendo lógica de importação**

**Implementando parsing e templates**

**Desenvolvendo regex para parsing**

**Codificando e executando...**

**Investigando falhas de leitura**

**Codificando e executando...**

**Investigando novo formato de arquivo**

**Codificando e executando...**

**Analisando markup de HTML**

**Codificando e executando...**

**Implementando extração de dados**

**Codificando e executando...**

**Codificando e executando...**

**Codificando e executando...**

**Codificando e executando...**

**Codificando e executando...**

**Atualizando dados e instruções**

**Ajustando extração de CSS**

**Codificando e executando...**

**Processando arquivos e preparando resposta**

**Compondo explicações detalhadas**

Show! Recebi os **3 primeiros HTMLs** e já integrei todos no **projeto Django + SQLite**, mantendo o **mesmo layout** (extraí o CSS dos arquivos e movi para static/). Aqui está o **ZIP atualizado**:

**▶ Download:** [statuscards\_project\_lote1.zip](blob:https://outlook.office.com/4a780c94-7cd6-4477-ab5d-e1337a4cb149)

**✅ O que foi processado agora**

* **Ex9 – Cards Horizontais**  
  Estrutura com **“Agrupador: …”** e cartões horizontais (<article class="hcard">). Extraí **nome**, **valor**, **status**, **ativo**, **fonte**, **faixa**, **análise**, **fórmula** e **definição**. Também captei os **agrupadores** e gravei em categoria.microsoft
* **Layout C – Mosaico por Classificação**  
  Seções por **status** (Ótimo, Muito Bom, etc.) com *pills* exibindo **indicador**, **valor** e **ativo** — usei isso para **preencher/confirmar** status e valor\_texto.Favicon typeex9\_cards\_horizontais\_from\_excel
* **Ex3 – Lista com Filtros**  
  Este HTML não traz dados; gerei a **página de lista com filtros** (via querystring) reaproveitando os campos que importamos.Favicon typeexC\_mosaico\_classificacao

**CSS preservado**: extraí o <style> … </style> do **Ex9** e do **Mosaico** e acrescentei em indicadores/static/indicadores/css/style.css. Assim o visual fica consistente com seus HTMLs.microsoft+1

**📊 Resultado da importação (lote 1)**

* **Ativos detectados:** ABEV3, AERI3.
* **Total de cards carregados:** **68** (deduplicados por (ativo, nome)).

Origem principal dos dados detalhados: **Ex9** (horizontal), complementados/validados com o **Mosaico** por status.microsoft+1

Esses dados estão no arquivo:  
indicadores/fixtures/indicadores.json (dentro do ZIP). [4](blob:https://outlook.office.com/4a780c94-7cd6-4477-ab5d-e1337a4cb149)

**🗂️ Estrutura e rotas disponíveis**

* **Home (lista de ativos):** /
* **Grid por ativo (Ex2-style):** /ativo/<TICKER>/
* **Cards Horizontais (Ex9):** /horizontais/ — agrupado por **categoria** (Agrupador).microsoft
* **Mosaico por Classificação (Layout C):** /mosaico/ — agrupado por **status**.Favicon typeex9\_cards\_horizontais\_from\_excel
* **Lista com Filtros (Ex3):** /lista/ — filtros por **status**, **categoria** e **ativo**.Favicon typeexC\_mosaico\_classificacao

**▶️ Como rodar**

Dentro do ZIP:

**Shell**

python -m venv .venv

# Windows: .venv\Scripts\activate

source .venv/bin/activate

pip install -r requirements.txt

python manage.py migrate

python manage.py loaddata indicadores/fixtures/indicadores.json

python manage.py runserver

Acesse: http://127.0.0.1:8000

Se preferir, posso te devolver também o db.sqlite3 já populado no próximo ZIP.

**🔧 Como importar mais HTMLs (local)**

O projeto inclui o comando:

**Shell**

# Reimportar do zero (mantém Ativos, zera Indicadores)

python manage.py importar\_html pasta\_ou\_arquivos --clear

# Ou upsert (sem duplicar), passando arquivos soltos

python manage.py importar\_html arq1.html arq2.html arq3.html

* **Deduplicação por (ativo, nome)**: se um indicador já existir, é **atualizado** (status/valor/faixa/…); se não existir, é criado.
* Suporta os **layouts** que você me enviou (**Ex9** e **Mosaico**), além do **grid Ex2**. [4](blob:https://outlook.office.com/4a780c94-7cd6-4477-ab5d-e1337a4cb149)

**🧠 Decisões técnicas (pra você auditar)**

* **categoria** = *Agrupador* (do Ex9).microsoft
* **Campos numéricos**: mantive **valor\_texto** (string) para preservar formatação de percentuais/moeda. Se quiser, no próximo lote eu já entrego **normalização** (valor\_num + unidade) e **filtros por faixa de valor**.
* **Layout**: reutilizei as **classes e semântica** dos seus HTMLs; a página “Lista com Filtros” é um *template* utilitário para exploração (o seu Ex3 não tinha dados, então criei filtros sobre o que importamos).Favicon typeexC\_mosaico\_classificacao

**Próximos passos**

Manda o **próximo lote de 3 HTMLs** quando quiser.  
Vou rodar o mesmo pipeline, manter tudo **sem duplicar**, e te devolvo outro **ZIP pronto**.

Quer que eu já habilite:

* **Normalização de valores** (número + unidade)
* **Ordenação e filtros avançados** (por valor/status/categoria)
* **Paginação** nas listas

? Posso aplicar no próximo retorno.

Fontes

Favicon type

exZ3\_flip\_cards\_3dhf01\_filter\_togglecardhf07\_filter\_quickstats

**You said:**

mais 3

**Copilot said:**

**Copilot**

**Raciocínio por 6s**

**Buscando clareza sobre "mais 3"**

**Propondo opções para "mais 3"**

**Propondo estrutura para cards**

Oi, Silvio! 👋  
Quando você disse **“mais 3”**, quer dizer **mais 3 *o quê*** exatamente?

Aqui estão algumas opções prováveis — me diga qual você quer que eu gere agora (ou outra coisa), que eu já trago pronto:

1. **Mais 3 cards 3D** no arquivo *Z3 – Flip 3D* (mesmo padrão “frente/verso”, puro CSS, sem JS).Favicon typeexZ3\_flip\_cards\_3d
2. **Mais 3 indicadores** para o *Heatmap (Filtro & Busca)* — adicionando blocos no mesmo estilo do *hf01* / *hf07*. (Vi que seus arquivos listam ABEV3 e AERI3 e seguem o layout de “Agrupador/Fórmula/Definição/Faixa/Descrição”.)Favicon typehf01\_filter\_togglecard+1
3. **Mais 3 ativos (tickers)** (ex.: PETR4, VALE3, ITUB4) com os mesmos componentes/indicadores já existentes.Favicon typehf01\_filter\_togglecard+1

Se preferir, diga o **tipo de indicador** (Valuation, Rentabilidade, Liquidez, Governança etc.) e eu já monto com *Fórmula*, *Definição*, *Faixa* e um texto de *Descrição* no mesmo tom dos seus arquivos.Favicon typehf01\_filter\_togglecard+1

**Quer ir adiantando?** Posso preparar imediatamente qualquer uma das opções acima e te devolver com o HTML pronto para colar no arquivo correspondente.

Fontes

**You said:**

o mesmo codigo que estou pendido do django, lembra que falei que iria mandar de 3 em 3

**Copilot said:**

**Copilot**

**Raciocínio por 8s**

**Interpretando mensagem curta**

**Propondo três componentes Django**

**Preparando código Django detalhado**

Perfeito, Silvio! 👊  
Seguindo o seu combinado de **mandar de 3 em 3**, aqui vão **os próximos 3 entregáveis em Django** (com **comentários linha a linha** no Python, como você curte):

1. **View + Template** para os **cards 3D “flip” por ativo**
2. **View + Template** para o **Heatmap (toggle + busca)**
3. **View + Template** para o **QuickStats (contagem por classificação)**

Incluí **models** e um **management command** para “seedar” rapidamente **ABEV3** e **AERI3** com parte dos indicadores que você mandou nos HTMLs, para você já ver tudo renderizando.

**0) Estrutura do app**

Supondo um app chamado fundamentus:

yourproject/

├─ yourproject/

│ ├─ settings.py

│ └─ urls.py

└─ fundamentus/

├─ migrations/

├─ management/

│ └─ commands/

│ └─ load\_sample\_metrics.py

├─ templates/

│ └─ fundamentus/

│ ├─ flip\_cards\_3d.html

│ ├─ filter\_togglecard.html

│ └─ quickstats.html

├─ admin.py

├─ apps.py

├─ models.py

├─ urls.py

└─ views.py

No settings.py, adicione fundamentus em INSTALLED\_APPS.

**1) models.py (com comentários detalhados)**

**Python**

from django.db import models

from django.utils import timezone

class Classification(models.TextChoices):

"""

Enum textual para manter as faixas visuais/semânticas de classificação

alinhadas ao que você usa nos HTMLs (Ótimo, Bom, etc.).

Usamos TextChoices para:

- ter validação de choices no banco,

- e manter legibilidade no código/admin.

"""

OTIMO = "Ótimo", "Ótimo"

MUITO\_BOM = "Muito Bom", "Muito Bom"

BOM = "Bom", "Bom"

MODERADO = "Moderado", "Moderado"

RUIM = "Ruim", "Ruim"

CRITICO = "Crítico", "Crítico"

MUITO\_CRITICO = "Muito Crítico", "Muito Crítico"

class Company(models.Model):

"""

Representa um ativo/ticker negociado (ex.: ABEV3, AERI3).

"""

ticker = models.CharField(

max\_length=16,

unique=True,

help\_text="Código do ticker, ex.: ABEV3"

)

name = models.CharField(

max\_length=128,

blank=True,

default="",

help\_text="Nome opcional da empresa (ex.: Ambev S.A.)"

)

def \_\_str\_\_(self) -> str:

*# Mostrar o ticker facilita debug e exibição em admin/listas.*

return self.ticker

class IndicatorGroup(models.TextChoices):

"""

Agrupadores conforme seus arquivos (Valuation, Rentabilidade, etc.).

Mantido como choices para padronização e filtros rápidos.

"""

VALUATION = "Valuation", "Valuation"

RENTABILIDADE = "Rentabilidade", "Rentabilidade"

LIQUIDEZ\_MERCADO = "Liquidez de Mercado", "Liquidez de Mercado"

LIQUIDEZ = "Liquidez", "Liquidez"

SOLVENCIA = "Solvência", "Solvência"

ESTRUTURA = "Estrutura de Capital", "Estrutura de Capital"

EFICIENCIA = "Eficiência Operacional", "Eficiência Operacional"

GOVERNANCA = "Governança Corporativa", "Governança Corporativa"

SAUDE = "Saúde Financeira", "Saúde Financeira"

OUTROS = "Outros", "Outros"

class IndicatorDefinition(models.Model):

"""

Metadados do indicador (fórmula, definição, faixa, referência, etc.).

Isso permite reuso da mesma definição para vários ativos.

"""

code = models.SlugField(

max\_length=64,

unique=True,

help\_text="Identificador curto para o indicador (ex.: 'dy', 'ev\_ebitda')."

)

label = models.CharField(

max\_length=64,

help\_text="Nome que aparece na UI (ex.: 'D.Y', 'EV/EBITDA')."

)

group = models.CharField(

max\_length=32,

choices=IndicatorGroup.choices,

default=IndicatorGroup.OUTROS,

help\_text="Agrupador/família do indicador."

)

formula = models.TextField(

help\_text="Fórmula exibida no verso do card/tooltip."

)

definition = models.TextField(

help\_text="Definição em linguagem natural do indicador."

)

band = models.CharField(

max\_length=128,

blank=True,

default="",

help\_text="Faixa/threshold textual (ex.: 'Dividend Yield > 6%')."

)

reference = models.CharField(

max\_length=128,

blank=True,

default="",

help\_text="Referência bibliográfica/fonte, se houver."

)

def \_\_str\_\_(self) -> str:

*# Ex.: "D.Y (dy)"*

return f"{self.label} ({self.code})"

class IndicatorValue(models.Model):

"""

Valor do indicador por empresa (snapshot).

Mantemos:

- 'value\_raw' como DecimalField para ordenação/filtragem numérica,

- 'value\_display' para preservar o formato original (ex.: '8,48%'),

- 'classification' para colorir/segmentar o heatmap.

"""

company = models.ForeignKey(

Company, on\_delete=models.CASCADE, related\_name="indicators"

)

indicator = models.ForeignKey(

IndicatorDefinition, on\_delete=models.CASCADE, related\_name="values"

)

value\_raw = models.DecimalField(

max\_digits=24,

decimal\_places=8,

null=True,

blank=True,

help\_text="Valor numérico bruto (para sort/filtro)."

)

value\_display = models.CharField(

max\_length=64,

blank=True,

default="",

help\_text="Valor formatado para exibição (ex.: '8,48%')."

)

classification = models.CharField(

max\_length=16,

choices=Classification.choices,

help\_text="Classificação do valor na faixa (Ótimo, Bom, ...)."

)

description = models.TextField(

blank=True,

default="",

help\_text="Texto descritivo da faixa/implicação (do verso do card)."

)

source = models.CharField(

max\_length=128,

blank=True,

default="StausInvest",

help\_text="Fonte do dado (ex.: StausInvest)."

)

updated\_at = models.DateTimeField(

default=timezone.now,

help\_text="Timestamp da coleta/cálculo."

)

class Meta:

*# Garante 1 valor por (empresa, indicador) no snapshot atual.*

unique\_together = ("company", "indicator")

def \_\_str\_\_(self) -> str:

*# Ex.: "ABEV3 - D.Y: 8.48 (Ótimo)"*

return f"{self.company} - {self.indicator.label}: {self.value\_display or self.value\_raw} ({self.classification})"

Mostrar mais linhas

**2) views.py (3 views, comentadas)**

**Python**

from django.db.models import Count

from django.shortcuts import get\_object\_or\_404, render

from django.http import HttpRequest, HttpResponse

from .models import Company, IndicatorValue, IndicatorDefinition, Classification

def flip\_cards\_3d(request: HttpRequest, ticker: str) -> HttpResponse:

"""

View 1/3: lista os indicadores de um ativo em cards 3D 'flip'.

- Recebe 'ticker' na URL (ex.: /fundamentus/flip/ABEV3/).

- Busca a empresa e todos os seus IndicatorValue + metadados (IndicatorDefinition).

- Renderiza um template com front/back (frente: valor + classificação; verso: fórmula, definição, faixa, ref e descrição).

"""

company = get\_object\_or\_404(Company, ticker\_\_iexact=ticker)

# Prefetch protege contra N+1 ao acessar .indicator.\* no template

values = (

IndicatorValue.objects.select\_related("indicator", "company")

.filter(company=company)

.order\_by("indicator\_\_group", "indicator\_\_label")

)

return render(

request,

"fundamentus/flip\_cards\_3d.html",

{

"company": company,

"values": values,

},

)

def filter\_togglecard(request: HttpRequest) -> HttpResponse:

"""

View 2/3: Heatmap com filtro por classificação e busca por ticker/indicador.

- Query params:

?q=texto (procura em ticker e label do indicador)

&class=Ótimo|Bom|... (filtra por classificação específica)

- Exibe cartões/tabelas no estilo do seu 'hf01\_filter\_togglecard.html'.

"""

q = (request.GET.get("q") or "").strip()

klass = (request.GET.get("class") or "").strip()

queryset = IndicatorValue.objects.select\_related("indicator", "company")

if q:

queryset = queryset.filter(

models.Q(company\_\_ticker\_\_icontains=q)

| models.Q(indicator\_\_label\_\_icontains=q)

| models.Q(indicator\_\_code\_\_icontains=q)

)

if klass in dict(Classification.choices):

queryset = queryset.filter(classification=klass)

# Uma ordenação que ajuda a leitura no heatmap

queryset = queryset.order\_by("company\_\_ticker", "indicator\_\_group", "indicator\_\_label")

# Para popular dropdown de classes no template

classes = [c[0] for c in Classification.choices]

return render(

request,

"fundamentus/filter\_togglecard.html",

{

"values": queryset,

"q": q,

"klass": klass,

"classes": classes,

},

)

def quickstats(request: HttpRequest) -> HttpResponse:

"""

View 3/3: QuickStats com contagem total de registros por classificação,

similar ao 'hf07\_filter\_quickstats.html'.

- Também mostra um top-N de indicadores por ativo (opcional), e

pode filtrar por ticker via ?ticker=ABEV3

"""

ticker = (request.GET.get("ticker") or "").strip()

base\_qs = IndicatorValue.objects.select\_related("company")

if ticker:

base\_qs = base\_qs.filter(company\_\_ticker\_\_iexact=ticker)

# Contagem por classificação para o cabeçalho/legenda

by\_class = (

base\_qs.values("classification")

.annotate(total=Count("id"))

.order\_by("classification")

)

# Exemplo opcional: contagem por empresa (para uma tabela rápida)

by\_company = (

base\_qs.values("company\_\_ticker")

.annotate(total=Count("id"))

.order\_by("company\_\_ticker")

)

return render(

request,

"fundamentus/quickstats.html",

{

"ticker": ticker,

"by\_class": by\_class,

"by\_company": by\_company,

"all\_classes": [c[0] for c in Classification.choices],

},

)

Mostrar mais linhas

**3) urls.py (do app)**

**Python**

from django.urls import path

from . import views

app\_name = "fundamentus"

urlpatterns = [

path("flip/<str:ticker>/", views.flip\_cards\_3d, name="flip"),

path("heatmap/", views.filter\_togglecard, name="heatmap"),

path("quickstats/", views.quickstats, name="quickstats"),

]

No yourproject/urls.py:

**Python**

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path("admin/", admin.site.urls),

path("fundamentus/", include("fundamentus.urls")),

]

**4) Templates**

**4.1) templates/fundamentus/flip\_cards\_3d.html**

**HTML**

{% load static %}

<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="utf-8"/>

<title>Z3 – Flip 3D · {{ company.ticker }}</title>

<style>

/\* Grid básico \*/

.grid { display: grid; grid-template-columns: repeat(auto-fill, minmax(260px, 1fr)); gap: 16px; }

/\* Card 3D container \*/

.card {

perspective: 1000px; /\* ativa profundidade 3D \*/

}

.inner {

position: relative;

width: 100%;

height: 220px;

transform-style: preserve-3d;

transition: transform .6s;

border-radius: 10px;

}

.card:hover .inner { transform: rotateY(180deg); }

/\* Faces \*/

.face {

position: absolute;

inset: 0;

backface-visibility: hidden;

border-radius: 10px;

padding: 14px;

color: #0f172a;

background: #ffffff;

border: 1px solid #e2e8f0;

display: flex;

flex-direction: column;

justify-content: space-between;

}

.back {

transform: rotateY(180deg);

background: #f8fafc;

}

/\* Cabeçalho e chips \*/

.header { font: 700 14px/1.2 system-ui, -apple-system, Segoe UI, Roboto, sans-serif; color: #0f172a; }

.sub { font: 12px/1.2 system-ui; color: #475569; }

.value { font: 600 22px/1.2 ui-monospace, SFMono-Regular, Menlo, Consolas, monospace; margin-top: 10px; }

/\* Cores por classificação \*/

.chip { display: inline-block; padding: 4px 8px; border-radius: 999px; font: 600 12px/1 system-ui; color: #0f172a; background: #e2e8f0; }

.Ótimo { background: #bef264; }

.Muito\.Bom { background: #d9f99d; }

.Bom { background: #a7f3d0; }

.Moderado { background: #fde68a; }

.Ruim { background: #fca5a5; }

.Crítico { background: #f87171; color: #111827; }

.Muito\.Crítico { background: #ef4444; color: #111827; }

.muted { color: #64748b; font-size: 12px; }

.k { font-weight: 600; color: #0f172a; }

.section-title { font: 700 12px/1.2 system-ui; color: #334155; margin: 8px 0 4px; }

</style>

</head>

<body>

<h1>Z3 – Flip 3D · {{ company.ticker }}</h1>

<p class="muted">Passe o mouse para ver fórmula, definição, faixa e referência.</p>

<div class="grid">

{% for iv in values %}

<div class="card">

<div class="inner">

<!-- Frente -->

<div class="face">

<div>

<div class="header">{{ iv.indicator.label }}</div>

<div class="sub">{{ iv.indicator.group }} · {{ iv.source }}</div>

<div class="value">

{{ iv.value\_display|default:iv.value\_raw }}

</div>

</div>

<div>

<span class="chip {{ iv.classification|slugify }}">{{ iv.classification }}</span>

</div>

</div>

<!-- Verso -->

<div class="face back">

<div>

<div class="section-title">Fórmula</div>

<div class="muted">{{ iv.indicator.formula }}</div>

<div class="section-title">Definição</div>

<div class="muted">{{ iv.indicator.definition }}</div>

{% if iv.indicator.band %}

<div class="section-title">Faixa</div>

<div class="muted">{{ iv.indicator.band }}</div>

{% endif %}

{% if iv.indicator.reference %}

<div class="section-title">Ref.</div>

<div class="muted">{{ iv.indicator.reference }}</div>

{% endif %}

{% if iv.description %}

<div class="section-title">Obs.</div>

<div class="muted">{{ iv.description }}</div>

{% endif %}

</div>

</div>

</div>

</div>

{% empty %}

<p>Nenhum indicador para {{ company.ticker }}.</p>

{% endfor %}

</div>

</body>

</html>

Mostrar mais linhas

**4.2) templates/fundamentus/filter\_togglecard.html**

A interatividade nas pré-visualizações de código estará disponível em breve

**4.3) templates/fundamentus/quickstats.html**

**HTML**

{% load static %}

<!DOCTYPE html>

<html lang="pt-BR">

<head>

<meta charset="utf-8"/>

<title>QuickStats</title>

<style>

body { font: 14px system-ui, -apple-system, Segoe UI, Roboto, sans-serif; color: #0f172a; }

.row { display:flex; gap:12px; align-items:center; margin-bottom:16px; }

input[type="text"] { padding: 8px 10px; border:1px solid #e2e8f0; border-radius:8px; min-width: 180px; }

.stats { display:grid; grid-template-columns: repeat(auto-fill, minmax(140px,1fr)); gap:10px; margin: 10px 0 20px; }

.tile { border:1px solid #e2e8f0; border-radius:10px; padding:10px; background:white; }

.count { font: 700 22px/1 ui-monospace, SFMono-Regular, Menlo, Consolas, monospace; }

.muted { color:#64748b; font-size:12px; }

table { border-collapse: collapse; width: 100%; }

th, td { border-bottom: 1px solid #e2e8f0; padding: 8px; text-align: left; }

th { background: #f8fafc; }

</style>

</head>

<body>

<h1>QuickStats</h1>

<form class="row" method="get">

<input type="text" name="ticker" value="{{ ticker }}" placeholder="Filtrar por ticker (ex.: ABEV3)"/>

<button type="submit">Aplicar</button>

</form>

<div class="stats">

{% for c in all\_classes %}

{% with found=None %}

{% for item in by\_class %}

{% if item.classification == c %}

<div class="tile">

<div class="count">{{ item.total }}</div>

<div class="muted">{{ c }}</div>

</div>

{% with found=1 %}{% endwith %}

{% endif %}

{% endfor %}

{% if not found %}

<div class="tile">

<div class="count">0</div>

<div class="muted">{{ c }}</div>

</div>

{% endif %}

{% endwith %}

{% endfor %}

</div>

<h3>Registros por empresa</h3>

<table>

<thead><tr><th>Ticker</th><th>Total de indicadores</th></tr></thead>

<tbody>

{% for row in by\_company %}

<tr><td>{{ row.company\_\_ticker }}</td><td>{{ row.total }}</td></tr>

{% empty %}

<tr><td colspan="2" class="muted">Sem dados.</td></tr>

{% endfor %}

</tbody>

</table>

</body>

</html>

Mostrar mais linhas

**5) management/commands/load\_sample\_metrics.py**

*(carrega rapidamente algumas métricas reais dos seus HTMLs para ABEV3 e AERI3)*

**Python**

from decimal import Decimal

from django.core.management.base import BaseCommand

from fundamentus.models import (

Company,

IndicatorDefinition,

IndicatorValue,

IndicatorGroup,

Classification,

)

class Command(BaseCommand):

help = "Carrega amostras de indicadores para ABEV3 e AERI3 (dados resumidos dos HTMLs)."

def handle(self, \*args, \*\*options):

# 1) Cria empresas

abev3, \_ = Company.objects.get\_or\_create(ticker="ABEV3", defaults={"name": "Ambev S.A."})

aeri3, \_ = Company.objects.get\_or\_create(ticker="AERI3", defaults={"name": "Aeris Energy"})

# 2) Define alguns indicadores (usei parte dos que veio nos arquivos)

defs = [

dict(

code="dy",

label="D.Y",

group=IndicatorGroup.RENTABILIDADE,

formula="Dividend Yield (%) = (Dividendos por Ação / Preço da Ação) \* 100",

definition="Mede o retorno anual dos dividendos em relação ao preço da ação.",

band="Dividend Yield > 6%",

reference="",

),

dict(

code="pl",

label="P/L",

group=IndicatorGroup.SAUDE,

formula="Preço/Lucro (P/L) = Preço da Ação / LPA",

definition="Relação entre preço e lucro por ação; sinaliza valuation vs. lucro.",

band="Faixas dependem do setor e ciclo",

reference="",

),

dict(

code="ev\_ebitda",

label="EV/EBITDA",

group=IndicatorGroup.VALUATION,

formula="EV/EBITDA = Enterprise Value / EBITDA",

definition="Valuation ajustado por dívida/caixa, relativo à geração operacional.",

band="4 < EV/EBITDA <= 8 (equilíbrio)",

reference="",

),

dict(

code="p\_ebitda",

label="P/EBITDA",

group=IndicatorGroup.VALUATION,

formula="P/EBITDA = Valor de Mercado / EBITDA",

definition="Preço relativo à geração de caixa operacional ajustada.",

band="0 <= P/EBITDA <= 6 (barato)",

reference="",

),

dict(

code="tag\_along",

label="TAG ALONG",

group=IndicatorGroup.GOVERNANCA,

formula="Tag Along (%) = Percentual do preço por ação pago ao controlador garantido aos minoritários",

definition="Mecanismo de proteção aos minoritários em venda de controle.",

band="80% <= Tag Along <= 100% (ou =100% para máximo)",

reference="",

),

]

created\_defs = {}

for d in defs:

obj, \_ = IndicatorDefinition.objects.get\_or\_create(code=d["code"], defaults=d)

# Atualiza metadados caso já exista (opcional)

obj.label = d["label"]

obj.group = d["group"]

obj.formula = d["formula"]

obj.definition = d["definition"]

obj.band = d["band"]

obj.reference = d["reference"]

obj.save()

created\_defs[d["code"]] = obj

# 3) Insere valores (recorte dos arquivos que você enviou)

samples = [

# ABEV3

dict(ticker="ABEV3", code="dy", value\_display="8,48%", value\_raw=Decimal("8.48"),

classification=Classification.OTIMO,

description="Dividend Yield bem alto, retorno via dividendos robusto.",

source="StausInvest"),

dict(ticker="ABEV3", code="pl", value\_display="13,25", value\_raw=Decimal("13.25"),

classification=Classification.OTIMO,

description="P/L confortável dentro do setor/consumo estável.",

source="StausInvest"),

dict(ticker="ABEV3", code="ev\_ebitda", value\_display="5,45", value\_raw=Decimal("5.45"),

classification=Classification.BOM,

description="Valuation equilibrado vs. EBITDA.",

source="StausInvest"),

dict(ticker="ABEV3", code="p\_ebitda", value\_display="5,88", value\_raw=Decimal("5.88"),

classification=Classification.OTIMO,

description="Faixa considerada barata para geração operacional.",

source="StausInvest"),

dict(ticker="ABEV3", code="tag\_along", value\_display="80%", value\_raw=Decimal("80"),

classification=Classification.MODERADO,

description="Proteção padrão (80%) aos minoritários.",

source="StausInvest"),

# AERI3

dict(ticker="AERI3", code="dy", value\_display="0,0%", value\_raw=Decimal("0"),

classification=Classification.CRITICO,

description="Sem distribuição de dividendos no período.",

source="StausInvest"),

dict(ticker="AERI3", code="pl", value\_display="-0,22", value\_raw=Decimal("-0.22"),

classification=Classification.CRITICO,

description="PL/Patrimônio negativo indica fragilidade.",

source="StausInvest"),

dict(ticker="AERI3", code="ev\_ebitda", value\_display="-2,35", value\_raw=Decimal("-2.35"),

classification=Classification.MUITO\_CRITICO,

description="EBITDA negativo; métrica perde sentido.",

source="StausInvest"),

dict(ticker="AERI3", code="p\_ebitda", value\_display="-0,32", value\_raw=Decimal("-0.32"),

classification=Classification.MUITO\_CRITICO,

description="EBITDA negativo; preço/EBITDA negativo é red flag.",

source="StausInvest"),

dict(ticker="AERI3", code="tag\_along", value\_display="100%", value\_raw=Decimal("100"),

classification=Classification.OTIMO,

description="Proteção total aos minoritários.",

source="StausInvest"),

]

for s in samples:

company = Company.objects.get(ticker=s["ticker"])

ind\_def = created\_defs[s["code"]]

IndicatorValue.objects.update\_or\_create(

company=company,

indicator=ind\_def,

defaults=dict(

value\_raw=s["value\_raw"],

value\_display=s["value\_display"],

classification=s["classification"],

description=s["description"],

source=s["source"],

),

)

self.stdout.write(self.style.SUCCESS("Amostras carregadas com sucesso."))

Mostrar mais linhas

**6) admin.py (opcional, rápido para inspeção)**

**Python**

from django.contrib import admin

from .models import Company, IndicatorDefinition, IndicatorValue

@admin.register(Company)

class CompanyAdmin(admin.ModelAdmin):

search\_fields = ["ticker", "name"]

list\_display = ["ticker", "name"]

@admin.register(IndicatorDefinition)

class IndicatorDefinitionAdmin(admin.ModelAdmin):

search\_fields = ["code", "label"]

list\_display = ["code", "label", "group"]

@admin.register(IndicatorValue)

class IndicatorValueAdmin(admin.ModelAdmin):

search\_fields = ["company\_\_ticker", "indicator\_\_label", "indicator\_\_code"]

list\_display = ["company", "indicator", "value\_display", "classification", "updated\_at"]

list\_filter = ["classification", "indicator\_\_group", "source"]

Mostrar mais linhas

**7) Como rodar**

**Shell**

python manage.py makemigrations fundamentus

python manage.py migrate

python manage.py load\_sample\_metrics

python manage.py runserver

Acesse:

* **Flip 3D (por ativo):** http://localhost:8000/fundamentus/flip/ABEV3/
* **Heatmap (filtro & busca):** http://localhost:8000/fundamentus/heatmap/?q=ABEV3&class=Ótimo
* **QuickStats:** http://localhost:8000/fundamentus/quickstats/?ticker=ABEV3

**Próximos 3?**

Quer que eu já mande os **próximos 3** (por exemplo: **endpoint JSON/REST**, **página de comparação ABEV3 vs AERI3 por indicador**, **gráfico de distribuição por classificação usando Chart.js**)?  
Se preferir outra tríade (ex.: **signals** para recalcular classificação, **perfis de setor**, **painel de upload CSV**), me fala que eu já preparo no mesmo padrão.

**You said:**

mas cade o codigo em zip

**Copilot said:**

**Copilot**

**Raciocínio por 34s**

**Preparando arquivos e zipando**

**Codificando e executando...**

Pronto, Silvio! Aqui está o **ZIP com o código Django** (models, views, urls, admin, templates e o management command para seed):

**➡️ Baixar:** [fundamentus\_django\_snippets.zip](blob:https://outlook.office.com/0f9b75f6-4f6e-4bd3-aa26-c3707a8855af)

**O que vem no pacote**

fundamentus\_app/

└─ fundamentus/

├─ admin.py

├─ apps.py

├─ management/

│ └─ commands/

│ └─ load\_sample\_metrics.py

├─ migrations/ # (vazia para você gerar localmente)

├─ models.py

├─ templates/

│ └─ fundamentus/

│ ├─ flip\_cards\_3d.html

│ ├─ filter\_togglecard.html

│ └─ quickstats.html

├─ urls.py

└─ views.py

README.txt

**Como rodar (resumo)**

1. Adicione fundamentus ao INSTALLED\_APPS.
2. Inclua as rotas:

**Python**

*# yourproject/urls.py*

from django.urls import path, include

urlpatterns = [

path("fundamentus/", include("fundamentus.urls")),

]

1. Migre e carregue amostras:

**Shell**

python manage.py makemigrations fundamentus

python manage.py migrate

python manage.py load\_sample\_metrics

1. Acesse:

* **Flip 3D:** /fundamentus/flip/ABEV3/
* **Heatmap (filtro & busca):** /fundamentus/heatmap/?q=ABEV3&class=Ótimo
* **QuickStats:** /fundamentus/quickstats/?ticker=ABEV3

Se quiser, já preparo o **próximo ZIP** com os **próximos 3** (ex.: endpoint REST/JSON, comparação ABEV3 vs AERI3, e gráfico Chart.js). Quer seguir nessa linha?

Fontes

Favicon type

**You said:**

vou passar mais 3

**Copilot said:**

**Copilot**

**Raciocínio por 1s**

**Preparando para receber mais tarefas**

**Fornecendo modelo e instruções**

Manda ver, Silvio! 🚀  
Pronto para receber **mais 3**. Assim que você enviar, eu já **gero o código e devolvo outro ZIP** prontinho.

Para agilizar, me passe cada item neste formato (copie e preencha):

**Item 1**

* **Tipo** (ex.: *endpoint REST/JSON*, *página Django/Template*, *view + template*, *job/command*, *model migration*, *gráfico/Chart.js*, etc.):
* **Rota/URL desejada** (se for view):  
  ex.: /fundamentus/comparar/<tickerA>/<tickerB>/
* **Campos/dados** (se for API, liste o JSON; se for tela, diga o que mostrar/ordenar/filtrar):
* **Model/DB** (precisa criar/alterar modelos? descreva campos):
* **Seed/Exemplo** (2–3 registros de exemplo para eu carregar no management command):
* **Observações** (regras de negócio, cálculos, faixas, estilos, etc.):

**Item 2**

*(mesmo esquema acima)*

**Item 3**

*(mesmo esquema acima)*

**Sugestões de próximos “3” (se quiser inspiração)**

1. **API REST** (DRF) para indicadores:
   * GET /api/companies/, GET /api/companies/<ticker>/indicators/, filtros (?group=Valuation&class=Ótimo), paginação.
2. **Comparador** de dois ativos (ABEV3 vs AERI3) por indicador, com barras lado-a-lado e destaque de quem “vence” por métrica.
3. **QuickCharts** com Chart.js: pizza por classificação e barras por agrupador (Valuation/Rentabilidade etc.), usando os mesmos dados.

Assim que você mandar os **3 itens** (mesmo que sejam diferentes dessas ideias), eu já **codifico e entrego o .zip** na sequência, no mesmo padrão dos anteriores.

hfm02\_modal\_comparehm03\_dense\_tooltiphm05\_filter\_sort

**You said:**

gere o codigo desses mais 3, sempre com o mesmo layout

**Copilot said:**

**Copilot**

**Raciocínio por 176s**

**Gerando o código Django**

**Integração e ajustes no modelo**

**Implementando visualizações e ajustes**

**Codificando e executando...**

Show de bola, Silvio! 💥  
Gerei o **código dos próximos 3** com **o mesmo layout base** dos anteriores, mapeando exatamente o que você enviou nos arquivos **hfm02\_modal\_compare.html**, **hm03\_dense\_tooltip.html** e **hm05\_filter\_sort.html** (usei seus textos/títulos/ideias de interação para guiar os templates e as views).Favicon typehfm02\_modal\_compare+2

**⬇️ Download do pacote (ZIP)**

**Baixar:** [fundamentus\_trinca2.zip](blob:https://outlook.office.com/9bdf0c17-3a8c-4d91-8190-6a7191fdf388)

**O que vem implementado**

**1) Comparador com Modal**

**Rota:** /fundamentus/compare/?a=ABEV3&b=AERI3

* Lista cada **indicador** uma vez, com **duas linhas de valor** (ticker A e ticker B).
* **Clique** no card abre um **modal** com: **valor + classificação** de A e B lado-a-lado, mais **fórmula, definição, faixa e referência** (puxa dos metadados do indicador).
* **Layout** e copy inspirados no seu *hfm02\_modal\_compare* (títulos, tom, interação).Favicon typehfm02\_modal\_compare

**2) Heatmap denso com Tooltip completo**

**Rota:** /fundamentus/heatmap/dense/?q=&class=

* Cartões compactos: **ticker · agrupador · fonte**, **valor** e **classificação**.
* **Hover** mostra **tooltip** detalhando **todos** os campos (fórmula, definição, faixa, referência e observação).
* Filtros por **busca** (ticker/indicador) e **classificação**.
* Em linha com seu *hm03\_dense\_tooltip* (estrutura e densidade de informação).Favicon typehm03\_dense\_tooltip

**3) Heatmap com filtro + ordenação**

**Rota:** /fundamentus/heatmap/smart/?class=&sort=label|ticker|valor&order=asc|desc

* **Chips** de classificação (inclusive “Todos”).
* **Ordenação** por **Indicador**, **Ticker** ou **Valor** (asc/desc).
* Visual respira o *hm05\_filter\_sort* (chips e controle de sort).Favicon typehm05\_filter\_sort

**Estrutura do ZIP**

fundamentus\_pack2/

└─ fundamentus/

├─ admin.py

├─ apps.py

├─ management/

│ └─ commands/

│ └─ load\_more\_metrics.py # novo seed desta trinca

├─ migrations/ # vazia (gere localmente)

├─ models.py # mesmo modelo (Company, IndicatorDefinition, IndicatorValue)

├─ templates/

│ └─ fundamentus/

│ ├─ hfm02\_modal\_compare.html # modal compare (novo)

│ ├─ hm03\_dense\_tooltip.html # heatmap denso + tooltip (novo)

│ ├─ hm05\_filter\_sort.html # heatmap filtro + sort (novo)

│ ├─ flip\_cards\_3d.html # placeholders p/ rotas anteriores

│ ├─ filter\_togglecard.html # placeholders p/ rotas anteriores

│ └─ quickstats.html # placeholders p/ rotas anteriores

├─ urls.py # inclui novas rotas

└─ views.py # 3 novas views + as anteriores

README.txt

**Obs.:** Mantive o **mesmo layout** visual base (tipografia, cartões, chips, grids, etc.) para todos os 3. O conteúdo e comportamento foram guiados pelos seus três HTMLs enviados (títulos, seção “Registros/Colunas”, chips, tooltip denso, modal de comparação).Favicon typehfm02\_modal\_compare+2

**Como rodar**

1. Adicione o app:

**Python**

*# settings.py*

INSTALLED\_APPS = [

*# ...*

"fundamentus",

]

1. Inclua as rotas:

**Python**

*# yourproject/urls.py*

from django.urls import path, include

urlpatterns = [

path("fundamentus/", include("fundamentus.urls")),

]

``

1. (Se ainda não tiver o schema) gere/migre:

**Shell**

python manage.py makemigrations fundamentus

python manage.py migrate

1. **Seed** desta trinca (muitos indicadores ABEV3/AERI3 — ROIC, P/Cap. Giro, DY, PL, Giro Ativos, M. EBITDA, EV/EBITDA, ROE, LMD, P/Ativo, Liq. Corrente, P/VP, EV/EBIT, TAG ALONG, M. Bruta, M. Líquida, PL/Ativos, DL/EBIT, Dívida Líquida, Dívida Bruta, DL/EBITDA, ROA, VPA, P/EBIT, P/Ativo Circ. Líq., LPA, PSR):

**Shell**

python manage.py load\_more\_metrics

1. Acesse:

* **Modal Compare:** /fundamentus/compare/?a=ABEV3&b=AERI3Favicon typehfm02\_modal\_compare
* **Heatmap denso (tooltip):** /fundamentus/heatmap/dense/?q=ABEV3&class=BomFavicon typehm03\_dense\_tooltip
* **Heatmap filtro+sort:** /fundamentus/heatmap/smart/?class=Ótimo&sort=valor&order=descFavicon typehm05\_filter\_sort

**Notas técnicas rápidas**

* **Modal compare:** agrupa por (group, label, code) e monta um map[code] em JS para abrir o modal com **A** vs **B** sem roundtrip. Se quiser, trocamos para abrir com ?code= e recarregar server-side.Favicon typehfm02\_modal\_compare
* **Dense tooltip:** tooltip “on hover” com **todos os metadados** (fórmula, definição, faixa, referência, obs). Mantém busca e filtro de classe como no heatmap anterior.Favicon typehm03\_dense\_tooltip
* **Filtro+Sort:** class via chips; sort por label/ticker/valor; order asc/desc. Usa value\_raw para ordenar por **valor** quando disponível.Favicon typehm05\_filter\_sort

**Quer a próxima trinca?**

Posso já deixar pronto **(3 de 3)**:

1. **/api/** (endpoints JSON)
2. **/fundamentus/compare///charts com barras lado-a-lado (Chart**

Fontes

Favicon type

**You said:**

vou mandar mas 2

**Copilot said:**

**Copilot**

**Raciocínio por 1s**

**Buscando clareza**

**Solicitando detalhes**

Manda, Silvio! 🚀  
Pronto para receber **mais 2**. Assim que você enviar, eu **gero o código e devolvo um ZIP** no mesmo layout que já estamos usando.

Para agilizar, manda cada um no formato abaixo (copie e preencha):

**Item 1**

* **Tipo** (ex.: *endpoint REST/JSON*, *view + template*, *command/cron*, *gráfico/Chart.js*, *upload CSV*, *signal/migration*, etc.):
* **Rota/URL desejada** (se for view ou API):  
  ex.: /fundamentus/api/companies/<ticker>/indicators/
* **Campos/dados** (se for API, descreva o JSON; se for tela, diga o que mostrar/ordenar/filtrar):
* **Model/DB** (precisa criar/alterar modelos? novos campos?):
* **Seed/Exemplo** (2–3 registros para eu incluir num management command):
* **Regras/Observações** (cálculos, thresholds, estilo, comportamento do UI, paginação, etc.):

**Item 2**

*(mesmo esquema acima)*

💡 **Se preferir, eu já assumo defaults e sigo com uma trinca reduzida de 2**:

1. **API REST mínima** (listagem de empresas, indicadores por ticker, filtros ?group=&class=&q=).
2. **Comparação com Chart.js** (barras lado-a-lado A vs B por indicador, usando as mesmas definições/valores).

É só responder: **“faz a API e o gráfico”** + os **endpoints/rotas** que quer, e eu já te entrego o **ZIP** prontinho.

v1\_tabelav2\_heatmap

**You said:**

segue mais, gere o codigo

**Copilot said:**

**Copilot**

**Raciocínio por 33s**

**Planejando a criação do código**

**Organizando a implementação**

**Detalhando a implementação do código**

**Codificando e executando...**

Top, Silvio! 💪  
Gerei o **código dos 2 itens** que você acabou de mandar, mantendo **o mesmo layout** e mapeando fielmente a ideia dos seus arquivos **v1\_tabela.html** (Tabela detalhada) e **v2\_heatmap.html** (Heatmap de status por categoria).Favicon typev1\_tabela+1

**⬇️ Download do pacote (ZIP)**

**Baixar:** [fundamentus\_dupla.zip](blob:https://outlook.office.com/35c3cea5-ca0a-488e-a78f-32bdcf513ee7)

**O que foi implementado**

**1) Tabela detalhada**

**Rota:** /fundamentus/table/?tickers=ABEV3,AERI3

* Cabeçalho no topo no estilo “**Ativos: … • Total de cartões: N**”, exatamente como no seu rascunho.
* Para cada ticker selecionado, renderiza uma **tabela** com as colunas:  
  **Indicador · Valor · Status · Categoria · Faixa · Resumo** (o “Resumo” vem de IndicatorValue.description, a “Faixa” de IndicatorDefinition.band).Favicon typev1\_tabela
* Ordem padrão: group → label (fica bem próximo do que você mostrou).

**2) Heatmap de status (por categoria)**

**Rota:** /fundamentus/heatmap/status/?tickers=ABEV3,AERI3

* Título e subtítulo: **“Heatmap de status — Cor por estado (Crítico → Ótimo)”**.
* Para cada ticker, lista **grupos/categorias** (Alavancagem, Eficiência Operacional, etc.) e os **indicadores** em cartões com **badge colorida** pela **classificação** (Ótimo, Bom, Moderado, …), exatamente como você descreveu.Favicon typev2\_heatmap
* As cores por status seguem a mesma paleta usada nos anteriores.

Obs.: As duas views pegam os **mesmos dados do seu modelo** (Company/IndicatorDefinition/IndicatorValue) e usam o value\_display quando vem formatado (ex.: “51,50%”, “358.014.350,52”), mantendo a estética dos seus HTMLs.Favicon typev1\_tabela+1

**Estrutura do ZIP**

fundamentus\_pack3/

└─ fundamentus/

├─ admin.py

├─ apps.py

├─ models.py

├─ templates/

│ └─ fundamentus/

│ ├─ v1\_tabela.html # Tabela detalhada (novo)

│ └─ v2\_heatmap.html # Heatmap de status (novo)

├─ urls.py # Rotas novas

└─ views.py # Views novas

README.txt

**Como usar**

1. Habilite o app:

**Python**

# settings.py

INSTALLED\_APPS = [

# ...

"fundamentus",

]

1. Inclua as rotas:

**Python**

# yourproject/urls.py

from django.urls import path, include

urlpatterns = [

path("fundamentus/", include("fundamentus.urls")),

]

1. (Se necessário) migre e **use seus seeds anteriores** (os mesmos que já te entreguei nas outras trincas) para popular **ABEV3** e **AERI3**.
2. Acesse:

* **Tabela detalhada:** /fundamentus/table/?tickers=ABEV3,AERI3Favicon typev1\_tabela
* **Heatmap de status:** /fundamentus/heatmap/status/?tickers=ABEV3,AERI3Favicon typev2\_heatmap

**Notas técnicas (rápidas)**

* **detailed\_table** agrupa por ticker e mostra **Faixa** a partir de IndicatorDefinition.band e **Resumo** a partir de IndicatorValue.description, como no seu v1. Contagem “Total de cartões” é o count() do queryset aplicado aos tickers.Favicon typev1\_tabela
* **status\_heatmap** monta um dicionário heatmap[ticker][group] = [IndicatorValue…] e renderiza cards por grupo com **badge** colorida por classificação, replicando o look & feel do v2.Favicon typev2\_heatmap

Se quiser, eu já **integro essas duas telas ao menu** das anteriores (Flip 3D, Heatmaps, Modal Compare, QuickStats) e te mando um **ZIP consolidado** com tudo. Ou seguimos com **mais 2** (por exemplo: **export CSV/Excel** da tabela/heatmap e **endpoint JSON** da mesma consulta). Como prefere?

Fontes

Favicon type

Adicione exportação para CSVCrie endpoint JSON para tabelaMonte gráfico de barras por grupo

Copilot de Mensagens

Ferramentas

O conteúdo gerado por IA pode estar incorreto

**O que vem pronto**

* **Projeto Django** z3site/ com manage.py, settings.py, urls.py, wsgi.py, asgi.py.
* **App** fundamentus/ com:
  + **Models:** Company, IndicatorDefinition, IndicatorValue.
  + **Migrations:** 0001\_initial.py (sobe o schema).
  + **Admin** para inspecionar/jogar dados.
  + **Seeds (management commands)**
    - load\_sample\_metrics (mínimo) e
    - load\_more\_metrics (pacotão ABEV3 + AERI3 com os indicadores que você usou nos HTMLs).
  + **Views + Templates** para **todas** as telas que entreguei nas etapas anteriores:
    - home.html: formulário para digitar o **ticker**;
    - overview.html: **página central** com links para todas as visões do ativo + **QuickStats** do ticker;
    - flip\_cards\_3d.html: Flip 3D por indicador;
    - filter\_togglecard.html: Heatmap com **Busca + Filtro** de classificação;
    - quickstats.html: contagem por **classificação** e por **empresa**;
    - hfm02\_modal\_compare.html: **Comparador** A×B com **modal**;
    - hm03\_dense\_tooltip.html: Heatmap **denso** com **tooltip** completo (fórmula/definição/faixa/ref);
    - hm05\_filter\_sort.html: Heatmap com **chips de classe** + **ordenação** por label/ticker/valor;
    - v1\_tabela.html: **Tabela detalhada** (Indicador/Valor/Status/Categoria/Faixa/Resumo);
    - v2\_heatmap.html: **Heatmap de status por categoria**.
  + **Roteamento** consolidado:
    - /fundamentus/ → **Home** (digita o ticker)
    - /fundamentus/overview/<ticker>/ → **Overview** com os links prontos
    - Demais rotas das visões já publicadas (flip/heatmaps/quickstats/compare/table).

As telas **Tabela detalhada** e **Heatmap de status** foram montadas a partir dos **v1** e **v2** que você enviou, mantendo a linguagem das colunas, categorias e estados.   
As telas **Modal Compare**, **Heatmap denso** e **Filtro+Sort** foram baseadas nos **hfm02**, **hm03** e **hm05** que você anexou; comportamento e layout seguem o mesmo padrão. [[eu-prod.as...rosoft.com]](https://eu-prod.asyncgw.teams.microsoft.com/v1/objects/0-weu-d11-3b0063ce4b75a188ad1156a540b9e237/views/original/z3site_fundamentus_full.zip), [[v1\_tabela]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/v1_tabela.html) [[eu-prod.as...rosoft.com]](https://eu-prod.asyncgw.teams.microsoft.com/v1/objects/0-weu-d11-3b0063ce4b75a188ad1156a540b9e237/views/original/z3site_fundamentus_full.zip), [[v2\_heatmap]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/v2_heatmap.html), [[v1\_tabela]](https://telefonicacorp-my.sharepoint.com/personal/silvio_puzzovio_telefonica_com/Documents/Microsoft%20Copilot%20Chat%20Files/v1_tabela.html)

**Como rodar**

Shell

# 1) Ambiente

python -m venv .venv

# Windows: .venv\Scripts\activate

# Linux/Mac:

source .venv/bin/activate

pip install django

# 2) Banco (SQLite) + schema

python manage.py makemigrations

python manage.py migrate

# 3) Popular dados

python manage.py load\_sample\_metrics # mínimo

# ou

python manage.py load\_more\_metrics # conjunto ampliado (ABEV3, AERI3 com muitos indicadores)

# 4) Subir

python manage.py runserver

Mostrar mais linhas

Acesse:

* **Home (digitar ticker):** http://localhost:8000/fundamentus/
* **Overview (links do ticker):** http://localhost:8000/fundamentus/overview/ABEV3/
* Exemplos de visões:
  + **Flip 3D:** /fundamentus/flip/ABEV3/
  + **Heatmap filtro+busca:** /fundamentus/heatmap/?q=ABEV3
  + **QuickStats:** /fundamentus/quickstats/?ticker=ABEV3
  + **Comparador (modal):** /fundamentus/compare/?a=ABEV3&b=AERI3
  + **Heatmap denso (tooltip):** /fundamentus/heatmap/dense/?q=ABEV3
  + **Heatmap filtro+sort:** /fundamentus/heatmap/smart/?sort=valor&order=desc
  + **Tabela detalhada:** /fundamentus/table/?tickers=ABEV3
  + **Heatmap status:** /fundamentus/heatmap/status/?tickers=ABEV3

**Como incluir novos HTMLs (pronto para escalar)**

1. **Crie** a view + template e **publique a rota** em fundamentus/urls.py.
2. **Registre** a tela em fundamentus/registry.py:

Python

# fundamentus/registry.py

from dataclasses import dataclass

from typing import Callable, Optional

@dataclass

class ViewEntry:

key: str

name: str

desc: str

build\_url: Callable[[str, Optional[str]], str]

ENTRIES = [

# ... já existem várias

ViewEntry(

key="minha\_nova\_tela",

name="Minha Nova Tela",

desc="Descrição amigável",

build\_url=lambda t, b: f"/fundamentus/minha-nova-tela/?q={t}",

),

]

Mostrar mais linhas

A **Overview** renderiza **automaticamente** os cartões/links da lista ENTRIES — sem precisar mexer na página.

**Observações técnicas**

* **Banco**: SQLite (db.sqlite3) é criado ao rodar migrate; os **management commands** populam os dados.
* **Overview**: além dos links, mostra **QuickStats** do ticker (contagem por classificação) – útil como header informativo.
* **Comparador**: na Overview, você escolhe o segundo ticker pelo **dropdown**; os links já levam com a=<ticker> e b=<outro>.
* **Ordenação/Busca**: preserva a mesma lógica dos seus HTMLs (chips de classe, sort=valor, q por ticker/indicador, etc.).
* **Estilo/Layout**: fontes, grid de cartões, chips e badges estão uniformes entre as páginas (fácil de trocar para Tailwind/Bootstrap depois).

Se quiser, posso te mandar uma **versão com Docker** (Dockerfile + docker-compose) para subir em 1 comando, ou consolidar tudo num **único ZIP** com **fixtures** (loaddata) além dos commands. Quer que eu gere já essa variante também?