Fase 1: Configuração e Coleta de Dados

Passo 1: Definir o Escopo Mínimo

Vamos começar com um objetivo simples e claro:

- **O quê?** Criar um painel que mostra a cotação atual e a variação diária de uma lista de ações brasileiras.
- **Como?** Usando Python para buscar os dados de uma API e exibi-los em um dashboard web local.
- **Ativos de exemplo:** `PETR4`, `VALE3`, `ITUB4`, `MGLU3`.

Passo 2: Preparar o Ambiente de Desenvolvimento

É uma boa prática usar um "ambiente virtual" para isolar as bibliotecas do seu projeto.

1. **Instale o Python:** Se ainda não tiver, baixe e instale o Python do site oficial (python.org).

Crie a Pasta do Projeto e o Ambiente Virtual: Abra seu terminal (Prompt de Comando, PowerShell ou Terminal do Linux/macOS) e digite os comandos abaixo:
Bash
```
# Cria e acessa a pasta do projeto
mkdir monitor-de-ativos
cd monitor-de-ativos

# Cria um ambiente virtual chamado 'venv'
python -m venv venv

# Ativa o ambiente virtual
# No Windows:
.\venv\Scripts\activate
# No macOS/Linux:
source venv/bin/activate
```

2. Seu terminal agora deve mostrar `(venv)` antes do caminho da pasta.

Instale as Bibliotecas Necessárias: Vamos usar `requests` para acessar a API, `pandas` para organizar os dados e `streamlit` para criar o dashboard web.
Bash
```
pip install requests pandas streamlit
```

3.

Passo 3: Escolher e Obter uma Chave de API

Para este guia, vamos usar a API da **brapi**, que possui um plano gratuito excelente e é muito simples de usar.

1. **Acesse o site:** Vá para brapi.dev.
2. **Cadastre-se:** Crie uma conta gratuita.
3. **Obtenha seu Token (Chave de API):** Após o login, você verá seu "Token de API" pessoal no painel principal. Copie este token, pois vamos usá-lo no nosso código. Ele será algo como `a1b2c3d4e5f6`.

Fase 2: O Coração do Código - Buscando os Dados

Nesta fase, criaremos o script Python que se conecta à API e busca as cotações.

Passo 4: Criando o Script e Buscando os Dados

Crie um arquivo na pasta do seu projeto chamado `app.py`. Este será o único arquivo que precisaremos.

Abra `app.py` no seu editor de código preferido (VS Code, PyCharm, etc.) e adicione o seguinte código:

Python

```
import requests
import pandas as pd

# --- CONFIGURAÇÕES INICIAIS ---
# Cole aqui o seu Token da API brapi
TOKEN_API = "SEU_TOKEN_AQUI"

# Lista de ativos que queremos monitorar
LISTA_DE_ATIVOS = "PETR4,VALE3,ITUB4,MGLU3"

# --- FUNÇÃO PARA BUSCAR DADOS NA API ---
def buscar_dados_ativos(token, ativos):
    """
    Busca os dados de cotação mais recentes para uma lista de ativos na API da brapi.
    """
    try:
        # Monta a URL da requisição
        url = f"https://brapi.dev/api/quote/{ativos}?token={token}"

        # Faz a requisição GET para a API
        response = requests.get(url)

        # Verifica se a requisição foi bem-sucedida (código 200)
        response.raise_for_status()

        # Extrai os dados em formato JSON
        dados = response.json()
```

```
        # Organiza os dados em um DataFrame do Pandas
        df = pd.DataFrame(dados['results'])

        # Seleciona e renomeia as colunas que nos interessam
        df_organizado = df[['symbol', 'regularMarketPrice', 'regularMarketChange',
'regularMarketChangePercent', 'logourl']]
        df_organizado.columns = ['Ativo', 'Preço Atual', 'Variação (R$)', 'Variação (%)', 'Logo']

        return df_organizado

    except requests.exceptions.HTTPError as errh:
        print(f"Erro de HTTP: {errh}")
        # Retorna um DataFrame vazio em caso de erro de HTTP (ex: token inválido)
        return pd.DataFrame()
    except requests.exceptions.ConnectionError as errc:
        print(f"Erro de Conexão: {errc}")
        # Retorna um DataFrame vazio em caso de erro de conexão
        return pd.DataFrame()
    except requests.exceptions.Timeout as errt:
        print(f"Erro de Timeout: {errt}")
        return pd.DataFrame()
    except requests.exceptions.RequestException as err:
        print(f"Erro na Requisição: {err}")
        return pd.DataFrame()

# --- TESTE DA FUNÇÃO ---
# (Esta parte será removida quando integrarmos com o Streamlit)
if __name__ == "__main__":
    dados_df = buscar_dados_ativos(TOKEN_API, LISTA_DE_ATIVOS)
    if not dados_df.empty:
        print("Dados dos Ativos:")
        print(dados_df)
```

Antes de continuar:

1. **Substitua** `"SEU_TOKEN_AQUI"` pelo seu token real da brapi.
2. Salve o arquivo `app.py`.

Execute um teste rápido no seu terminal para ver se a busca de dados está funcionando:
Bash
python app.py

3. Você deverá ver uma tabela com os dados dos ativos impressa no terminal.

Fase 3: Criando o Dashboard Web com Streamlit

Agora, vamos transformar nosso script em uma aplicação web interativa.

Passo 5: Integrando o Código com o Streamlit

Modifique o arquivo `app.py` para usar os comandos do Streamlit. Substitua todo o conteúdo do arquivo pelo código abaixo:

Python

```
import streamlit as st
import requests
import pandas as pd
from datetime import datetime

# --- CONFIGURAÇÕES INICIAIS DA PÁGINA E DA API ---
st.set_page_config(
    page_title="Dashboard de Ativos B3",
    page_icon="🇧🇷",
    layout="wide"
)

# Cole aqui o seu Token da API brapi
TOKEN_API = "SEU_TOKEN_AQUI" # <<<<< COLOQUE SEU TOKEN AQUI

# Lista de ativos que queremos monitorar um por vez
LISTA_DE_ATIVOS = "PETR4"

# --- FUNÇÃO PARA BUSCAR DADOS NA API ---
@st.cache_data(ttl=600) # Cache para não sobrecarregar a API (atualiza a cada 10 minutos)
def buscar_dados_ativos(token, ativos):
    """
    Busca os dados de cotação mais recentes para uma lista de ativos na API da brapi.
    """
    try:
        url = f"https://brapi.dev/api/quote/{ativos}?token={token}"
        response = requests.get(url)
        response.raise_for_status()
        dados = response.json()
        df = pd.DataFrame(dados['results'])
        df_organizado = df[['symbol', 'regularMarketPrice', 'regularMarketChangePercent', 'logourl']]
        df_organizado.columns = ['Ativo', 'Preço', 'Variação (%)', 'Logo']
        return df_organizado
    except Exception as e:
        st.error(f"Erro ao buscar dados da API: {e}")
        return pd.DataFrame()
```

```
# --- CONSTRUÇÃO DO DASHBOARD ---

# Título e cabeçalho
st.title("📈 Dashboard de Monitoramento de Ativos - B3")
st.markdown(f"Última atualização: {datetime.now().strftime('%d/%m/%Y %H:%M:%S')}")

# Busca os dados
dados_df = buscar_dados_ativos(TOKEN_API, LISTA_DE_ATIVOS)

if not dados_df.empty:
    # Exibição dos cards de métricas
    st.subheader("Resumo dos Ativos")

    # Criando colunas para os cards
    cols = st.columns(len(dados_df))

    for i, row in dados_df.iterrows():
        with cols[i]:
            # Define a cor da variação
            cor_delta = "normal"
            if row['Variação (%)'] < 0:
                cor_delta = "inverse"

            st.metric(
                label=row['Ativo'],
                value=f"R$ {row['Preço']:.2f}",
                delta=f"{row['Variação (%)']:.2f}%",
                delta_color=cor_delta
            )

    # Tabela com dados detalhados
    st.subheader("Detalhes dos Ativos")

    # Configuração da exibição da imagem na tabela
    st.dataframe(
        dados_df,
        column_config={
            "Logo": st.column_config.ImageColumn("Logo da Empresa"),
            "Preço": st.column_config.NumberColumn(
                "Preço (R$)",
                format="R$ %.2f"
            ),
            "Variação (%)": st.column_config.NumberColumn(
                "Variação Diária",
                format="%.2f%%"
            ),
        },
```

```
        hide_index=True,
        use_container_width=True
    )

    # Botão para forçar a atualização dos dados
    if st.button('Atualizar Dados'):
        st.cache_data.clear()
        st.rerun()

else:
    st.warning("Não foi possível buscar os dados dos ativos. Verifique seu token da API ou a conexão.")
```

Passo 6: Rodando a Sua Aplicação Web

1. **Lembre-se de substituir o token** no novo código.
2. Salve o arquivo `app.py`.

Volte ao seu terminal (com o ambiente virtual `(venv)` ainda ativo) e execute o seguinte comando:
Bash
streamlit run app.py

3.
4. O Streamlit iniciará um servidor local e abrirá automaticamente uma aba no seu navegador. Se não abrir, o terminal mostrará os endereços (Local URL e Network URL) para você acessar.

Pronto! Você acaba de criar seu primeiro dashboard de monitoramento de ativos!

Fase 4: Próximos Passos e Melhorias

Este é apenas o começo. A partir daqui, você pode evoluir seu projeto:

- **Entrada do Usuário:** Use `st.text_input` ou `st.multiselect` para permitir que o cliente digite quais ativos ele deseja monitorar, em vez de usar uma lista fixa.
- **Gráficos Históricos:** Use a mesma API (ou a biblioteca `yfinance`) para buscar dados históricos e plote gráficos de evolução de preços com `st.line_chart` ou bibliotecas mais avançadas como `plotly`.
- **Cálculo de Indicadores:** Com os dados históricos, calcule médias móveis, volatilidade ou outros indicadores fundamentalistas.

- **Alertas:** Crie uma lógica que verifique se um ativo atingiu um determinado preço e envie uma notificação (por exemplo, usando a API do Telegram ou enviando um e-mail com a biblioteca `smtplib`).
- **Deploy:** Use o "Streamlit Community Cloud" para hospedar seu dashboard gratuitamente na internet e compartilhá-lo com outras pessoas.