Mini Dashboard de Criptomoedas / Ações

Banco de Dados

- Usaremos **MongoDB** para armazenar dados históricos e cache.

1️⃣ Requisitos Funcionais

1. Buscar preços atuais de ativos (ações ou criptomoedas) via API externa (`yfinance` ou `CoinGecko`).
2. Fornecer variação do preço nas últimas 24 horas.
3. Gerar gráfico simples de histórico de preços (últimos 7 ou 30 dias).
4. Disponibilizar endpoints REST:
5. `/price?symbol=BTC` → retorna preço atual e variação.
6. `/history?symbol=BTC&days=30` → retorna histórico de preços.
7. `/prediction?symbol=BTC` → retorna previsão de tendência (opcional).
8. Filtrar ativos por tipo (ações ou criptomoedas).
9. Armazenar dados históricos no MongoDB para cache.

2️⃣ Requisitos Não Funcionais

1. Backend em Python (recomendado FastAPI ou Flask).
2. Sistema modular e de fácil manutenção.
3. Tempo de resposta da API aceitável (<1s para dados em cache).
4. Logs de requisições e erros salvos.
5. Respeito a limites de API externa (rate limit).
6. Código versionado no Git com README explicando setup.

3️⃣ Casos de Uso

Ator	Ação do Sistema	Resultado Esperado
Usuário/API	Solicita preço atual de um ativo	Retorna preço atual, variação em 24h
Usuário/API	Solicita histórico de preços de um ativo	Retorna lista de preços diários, pronto para gráfico
Usuário/API	Solicita previsão de preço	Retorna previsão de curto prazo (regressão simples)
Sistema	Atualiza dados históricos localmente	Base de dados interna atualizada automaticamente
Sistema/Admin	Loga erros ou problemas de API externa	Facilita manutenção e debug

---

4️⃣ Roteiro Prático de Implementação

Fase 1: Setup do Projeto

1. Criar ambiente virtual (`venv` ou `poetry`).
2. Instalar dependências:

```
pip install fastapi uvicorn yfinance pandas matplotlib requests scikit-learn pymongo
```

3. Estrutura básica de pastas:

```
crypto_dashboard/
├─ main.py
├─ api/
│  ├─ price.py
│  ├─ history.py
│  └─ prediction.py
├─ data/
│  └─ cache.db (MongoDB URI)
├─ models/
│  └─ regression_model.pkl
├─ utils/
│  └─ helpers.py
└─ requirements.txt
```

Fase 2: Integração com API Externa

1. Função para buscar dados do **yfinance** (ações) ou **CoinGecko** (cripto).
2. Normalizar dados em pandas DataFrame.
3. Salvar dados históricos no MongoDB para caching.

Fase 3: Criar Endpoints REST

1. `/price` → retorna preço atual + variação 24h.
2. `/history` → retorna histórico de preços (JSON ou gráfico).
3. `/prediction` → retorna previsão de preço futuro usando regressão linear simples.

Exemplo básico com FastAPI:

```
from fastapi import FastAPI
import yfinance as yf

app = FastAPI()
```

```
@app.get("/price")
def get_price(symbol: str):
    data = yf.Ticker(symbol).history(period="1d")
    price = data['Close'][-1]
    variation = (data['Close'][-1] - data['Open'][-1]) / data['Open'][-1] *
100
    return {"symbol": symbol, "price": price, "variation": variation}
```

Fase 4: Gráficos

1. Gerar gráficos de histórico com `matplotlib` ou `plotly`.
2. Salvar gráfico como PNG ou retornar base64 via API.

Fase 5: Modelo de Previsão Simples

1. Usar regressão linear (`scikit-learn`) com dados históricos.
2. Treinar modelo com preços passados e prever próximos 1-3 dias.
3. Endpoint `/prediction` retorna previsão.

Fase 6: Testes e Deploy

1. Testar endpoints com `Postman` ou `curl`.
2. Adicionar tratamento de erros e logs.
3. Rodar localmente:

   ```
   uvicorn main:app --reload
   ```

4. Opcional: deploy no Heroku, Railway ou Docker.