**Program Design Document (PDD) - script-2.py**

**1. Introdução**

Este documento descreve o design e a arquitetura do script Python teste-2.py, que tem como objetivo coletar dados de ações financeiras utilizando a API Alpha Vantage. Diferente de abordagens de web scraping, este script foca na interação com uma API oficial, garantindo maior estabilidade, confiabilidade e conformidade na obtenção de dados de mercado. Ele foi desenvolvido para extrair informações como preço, variação, volume, capitalização de mercado e relação P/E, e para persistir esses dados em um formato estruturado (CSV).

**1.1. Propósito**

O principal propósito do teste-2.py é fornecer uma solução robusta e programática para a coleta automatizada de dados de ações através da API Alpha Vantage. Isso permite que usuários e sistemas obtenham informações de mercado atualizadas de forma eficiente e confiável, facilitando análises, monitoramento e integração com outras aplicações, ao mesmo tempo em que respeita os limites de uso da API.

**1.2. Escopo**

O escopo deste script abrange:

•Coleta de Dados via API: Extração de dados de cotação de ações (GLOBAL\_QUOTE) e informações gerais da empresa (OVERVIEW) da API Alpha Vantage.

•Processamento de Dados: Parsing das respostas JSON da API e formatação dos dados em um dicionário padronizado.

•Combinação de Dados: Unificação de informações de diferentes endpoints da API para formar um conjunto de dados mais completo por ação.

•Persistência de Dados: Salvamento dos dados coletados em um arquivo CSV.

•Gerenciamento de Limites de Taxa: Implementação de atrasos para respeitar os limites de requisições da API Alpha Vantage.

•Registro de Eventos: Utilização de logging para monitorar o progresso e diagnosticar problemas.

O script não abrange:

•Coleta de dados históricos ou financeiros detalhados (séries temporais, balanços, demonstrações de resultados) além do escopo das funções GLOBAL\_QUOTE e OVERVIEW.

•Autenticação complexa ou gerenciamento de sessões para a API (a chave de API é suficiente).

•Análise de dados avançada ou visualização.

•Integração com bancos de dados ou outros sistemas de armazenamento além de CSV.

**1.3. Definições, Acrônimos e Abreviações**

•API: Application Programming Interface, um conjunto de definições e protocolos para construir e integrar software de aplicativos.

•Alpha Vantage: Provedor de APIs para dados financeiros e econômicos.

•JSON: JavaScript Object Notation, um formato de dados leve e legível por humanos para troca de dados.

•Pandas: Biblioteca Python para manipulação e análise de dados, fornecendo estruturas de dados como DataFrames.

•CSV: Comma Separated Values, um formato de arquivo de texto simples para armazenar dados tabulares.

•HTTP: Hypertext Transfer Protocol, o protocolo para transferência de informações na World Wide Web.

•Rate Limiting: Mecanismo usado por servidores de API para controlar o número de requisições que um cliente pode fazer em um determinado período.

•PDD: Program Design Document.

**2. Design de Alto Nível**

O teste-2.py segue um design modular e orientado a objetos, encapsulando a lógica de interação com a API Alpha Vantage em uma classe dedicada. Isso promove a reutilização de código, a manutenibilidade e a clareza da estrutura.

**2.1. Arquitetura Geral**

A arquitetura do script é linear e sequencial, com uma classe central (AlphaVantageCollector) que orquestra as operações de coleta via API. A função main() atua como o ponto de entrada, inicializando o coletor, definindo as ações a serem coletadas e chamando os métodos apropriados para executar a coleta e salvar os dados.

mermaid

Fonte

Falha ao renderizar o mermaid

**2.2. Componentes Principais**

•AlphaVantageCollector Classe: O coração do script, responsável por toda a lógica de interação com a API.

•Métodos Auxiliares Privados (\_ prefixo): Funções internas da classe que lidam com detalhes de implementação, como fazer requisições à API (\_make\_request) e parsing de diferentes tipos de dados da API (\_parse\_quote\_data, \_parse\_overview\_data).

•Métodos Públicos: Funções da classe que expõem a funcionalidade principal, como get\_stock\_quote, get\_company\_overview, get\_complete\_stock\_data, get\_multiple\_stocks\_data e save\_to\_csv.

•main() Função: O ponto de entrada do script, responsável pela configuração (incluindo a chave da API) e execução do fluxo principal.

•Módulo logging: Para fornecer feedback sobre o progresso e erros durante a execução.

**3. Design Detalhado**

**3.1. Classe AlphaVantageCollector**

**3.1.1. Atributos**

•api\_key (str): Chave de API da Alpha Vantage, necessária para autenticação nas requisições.

•output\_dir (str): Caminho para o diretório onde os arquivos CSV serão salvos. Inicializado no construtor.

•base\_url (str): URL base da API Alpha Vantage (https://www.alphavantage.co/query).

•session (requests.Session): Objeto de sessão requests para persistir parâmetros entre requisições.

**3.1.2. Métodos**

•\_\_init\_\_(self, api\_key: str, output\_dir: str = "data"):

•Propósito: Inicializar a instância do coletor, configurar a chave da API e o diretório de saída.

•Lógica: Cria o diretório output\_dir se não existir. Armazena a api\_key. Emite um aviso se a api\_key for inválida ou padrão.

•\_make\_request(self, function: str, symbol: str, \*\*kwargs) -> Optional[Dict]:

•Propósito: Construir e enviar requisições para a API Alpha Vantage.

•Lógica: Monta os parâmetros da requisição (função, símbolo, chave da API e quaisquer outros argumentos). Utiliza self.session.get() para fazer a requisição. Verifica o status\_code da resposta e o conteúdo JSON para mensagens de erro ou informação da API. Retorna o JSON da resposta ou None.

•Tratamento de Erros: Captura requests.exceptions.RequestException para lidar com problemas de rede ou timeout. Verifica chaves como "Error Message" e "Information" no JSON da resposta.

•\_parse\_quote\_data(self, data: Dict, symbol: str) -> Optional[Dict]:

•Propósito: Analisar e formatar os dados retornados pela função GLOBAL\_QUOTE da API.

•Lógica: Extrai os dados da chave "Global Quote". Mapeia os campos da API (ex: "05. price") para nomes mais amigáveis e padronizados. Adiciona a data/hora da coleta e a fonte. Campos não disponíveis nesta função (nome completo, capitalização de mercado, P/E) são marcados como "N/A".

•Tratamento de Erros: Verifica a estrutura esperada dos dados e captura exceções gerais durante o parsing.

•\_parse\_overview\_data(self, data: Dict, symbol: str) -> Optional[Dict]:

•Propósito: Analisar e formatar os dados retornados pela função OVERVIEW da API.

•Lógica: Extrai os dados diretamente do dicionário data. Mapeia campos como "Name", "MarketCapitalization" e "PERatio". Adiciona a data/hora da coleta e a fonte. Campos não disponíveis nesta função (preço, variação, volume) são marcados como "N/A".

•Tratamento de Erros: Verifica a estrutura esperada dos dados e captura exceções gerais durante o parsing.

•get\_stock\_quote(self, symbol: str) -> Optional[Dict]:

•Propósito: Obter apenas os dados de cotação de uma ação.

•Lógica: Chama \_make\_request com a função GLOBAL\_QUOTE e, em seguida, \_parse\_quote\_data.

•get\_company\_overview(self, symbol: str) -> Optional[Dict]:

•Propósito: Obter apenas os dados de visão geral de uma empresa.

•Lógica: Chama \_make\_request com a função OVERVIEW e, em seguida, \_parse\_overview\_data.

•get\_complete\_stock\_data(self, symbol: str) -> Optional[Dict]:

•Propósito: Obter um conjunto completo de dados para uma ação, combinando informações de cotação e visão geral.

•Lógica: Primeiro, chama get\_stock\_quote. Se bem-sucedido, chama get\_company\_overview. Em seguida, itera sobre os dados da visão geral para preencher quaisquer campos "N/A" nos dados de cotação, resultando em um dicionário de dados mais completo.

•get\_multiple\_stocks\_data(self, symbols: List[str]) -> pd.DataFrame:

•Propósito: Coletar dados completos para uma lista de ações.

•Lógica: Itera sobre a lista de symbols. Para cada símbolo, chama get\_complete\_stock\_data. Adiciona um atraso fixo de 12 segundos (time.sleep(12)) após cada requisição para respeitar o limite de 5 chamadas por minuto da API gratuita da Alpha Vantage. Constrói um pd.DataFrame a partir dos dados coletados.

•save\_to\_csv(self, df: pd.DataFrame, filename: str = None) -> str:

•Propósito: Salvar o DataFrame de dados em um arquivo CSV.

•Lógica: Se filename não for fornecido, gera um nome de arquivo único baseado na data e hora atuais. Constrói o caminho completo do arquivo usando os.path.join(). Utiliza df.to\_csv() para salvar o DataFrame, sem incluir o índice.

**3.2. Função main()**

•Propósito: Ponto de entrada e controle do fluxo de execução do script.

•Lógica: Define a api\_key (requer substituição pelo usuário). Define uma lista padrão de símbolos de ações. Instancia AlphaVantageCollector. Chama get\_multiple\_stocks\_data para iniciar a coleta. Se dados forem retornados, chama save\_to\_csv e imprime os dados no console. Caso contrário, imprime uma mensagem de falha.

**4. Requisitos Não Funcionais**

•Performance: A performance do script é diretamente limitada pelos limites de taxa da API Alpha Vantage (5 requisições por minuto para a chave gratuita). O atraso de 12 segundos por requisição é uma medida para garantir a conformidade com esses limites. Não é otimizado para coleta em massa de milhares de ações simultaneamente com a chave gratuita.

•Confiabilidade: A confiabilidade da coleta de dados é alta, pois depende de uma API oficial, que é mais estável do que web scraping. Erros de rede e mensagens de erro da API são tratados para evitar falhas completas.

•Segurança: A chave da API é exposta diretamente no código na função main(). Para ambientes de produção, a chave deve ser armazenada de forma mais segura (ex: variáveis de ambiente, gerenciadores de segredos).

•Manutenibilidade: O código é modular e bem comentado, facilitando a compreensão e futuras modificações. A separação de responsabilidades entre os métodos da classe contribui para a manutenibilidade.

•Escalabilidade: A escalabilidade é limitada pelos limites de taxa da API. Para maior escalabilidade, seria necessária uma assinatura premium da Alpha Vantage com limites de taxa mais altos.

**5. Plano de Testes (Visão Geral)**

Os testes para este script devem focar na validação da interação com a API, no tratamento de diferentes respostas da API e na conformidade com os limites de taxa.

•Testes Unitários: Embora não implementados no script atual, testes unitários seriam ideais para:

•\_make\_request(): Testar com chaves de API válidas/inválidas, símbolos existentes/inexistentes, simular erros de rede e diferentes respostas da API (sucesso, erro, informação).

•\_parse\_quote\_data() e \_parse\_overview\_data(): Testar com diferentes estruturas JSON (válidas e inválidas) para garantir que os dados sejam extraídos corretamente ou que o tratamento de erros funcione.

•get\_complete\_stock\_data(): Verificar se a combinação de dados funciona conforme o esperado, preenchendo "N/A" corretamente.

•save\_to\_csv(): Verificar se o arquivo CSV é criado corretamente e se o conteúdo corresponde ao DataFrame.

•Testes de Integração: Testar o fluxo completo:

•Executar o script com uma lista de símbolos de ações conhecidos e verificar se o arquivo CSV é gerado com os dados esperados.

•Testar com símbolos de ações inexistentes para verificar o tratamento de erros da API.

•Monitorar o comportamento do script ao longo do tempo para identificar se os limites de taxa estão sendo respeitados e se há bloqueios.

•Testes de Performance: Medir o tempo de execução para diferentes quantidades de ações e verificar se o atraso de 12 segundos é suficiente para evitar problemas com o limite de taxa.

**6. Considerações de Implantação**

O script é autônomo e pode ser executado em qualquer ambiente Python 3.x com as dependências instaladas. Requer uma chave de API da Alpha Vantage.

•Ambiente: Python 3.x.

•Dependências: requests, pandas.

•Chave de API: Necessária uma chave válida da Alpha Vantage.

•Execução: Via linha de comando (python teste-2.py).

•Saída: Arquivos CSV gerados no diretório data/ (ou configurado).

**7. Melhorias e Evolução Futura**

As melhorias sugeridas na documentação (documentation\_teste-2.md) também se aplicam ao design e podem ser consideradas para futuras iterações:

•Gerenciamento de Chave de API: Armazenar a chave de API em variáveis de ambiente ou um arquivo de configuração seguro.

•Tratamento de Erros Aprimorado: Adicionar retries com backoff exponencial para requisições da API.

•Paralelização: Para contas premium da Alpha Vantage, explorar o uso de threading ou asyncio para coletar dados de múltiplas ações de forma mais eficiente.

•Persistência em Banco de Dados: Integrar com um banco de dados (SQLite, PostgreSQL, MongoDB) para armazenamento mais robusto e consultas complexas.

•Dockerização: Criar um Dockerfile para empacotar o script e suas dependências, garantindo um ambiente de execução consistente e facilitando a implantação.

•Seleção de Funções da API: Permitir que o usuário especifique quais funções da API Alpha Vantage deseja usar para coletar dados, tornando o script mais flexível.

8. Autor

Lauro Bonometti.