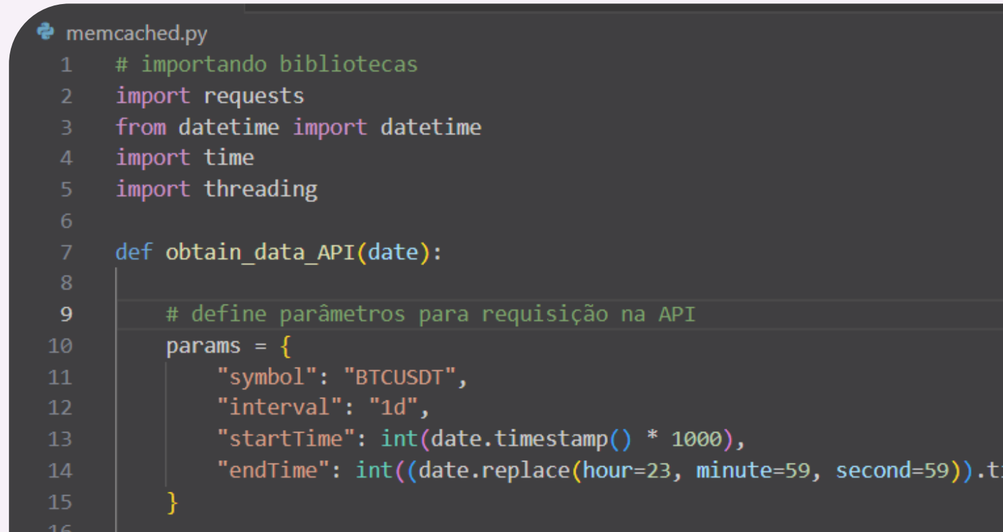
MEMCACHED

Simulação de Memcached em Python para consulta de dados da Binance

Bruno , Jorge, José e Rafael



Motivação/Objetivo

A ideia do trabalho é mostrar na prática como funciona um cache em memória com TTL, acelerando consultas repetidas em dados históricos de uma API real, simulando o comportamento do Memcached

Estrutura do Código

O código inicia importando bibliotecas para requisições web, manipulação de datas e controle de tempo e concorrência.

A função obtain_data_API recebe uma data, monta os parâmetros e pega, pela API pública da Binance, dados do Bitcoin para aquele dia

Funcionamento do Cache

Montamos um dicionário chamado MEMCACHED. Quando solicitamos uma data, primeiro verificamos se ela está no cache:

Se estiver, mostramos que veio do cache, acelerando o processamento.

Se não, buscamos na API, armazenamos no cache junto com o timestamp.

O cache tem TTL configurável pelo usuário — passado esse tempo, a entrada é removida automaticamente, simulando expiração real

Controle de TTL e ciclo do dado

No loop principal, além de executar as consultas, percorremos todos os itens do cache, mostramos quanto tempo cada um está ativo, e removemos os que passaram do TTL Time To Live (Tempo de Vida). O usuário pode escolher visualizar esse Ciclo de Vida do cache

Demonstração e Resultados

Rodando o programa, mostramos que a primeira consulta para uma data é lenta (pois faz a requisição), mas as próximas — para a mesma data — são instantâneas graças ao cache. Após o TTL, o dado é excluído e uma nova consulta volta a ser feita na API

Aprendizados

Aprendemos na prática o uso de TTL, gerenciamento do cache, controle concorrente de entrada com threads e integração real de API, tornando o trabalho completo para estudar conceitos de cache além da teoria.

Conclusão

Reforçamos a utilidade do Memcached para aplicações reais, principalmente para grandes volumes de requisições, e como TTL ajuda no controle automático do ciclo de vida dos dados