PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

Sistema de Cálculo de Cotas de Fundos de Criptomoedas

Rodrigo Biscaia Fernandes

Projeto Final de Graduação

CENTRO TÉCNICO CIENTÍFICO – CTC

DEPARTAMENTO DE INFORMÁTICA

Curso de Graduação em Engenharia da Computação

Rio de Janeiro, Dezembro de 2024

Rodrigo Biscaia Fernandes

Sistema de Cálculo de Cotas de Fundos de Criptomoedas

Relatório de Projeto Final, apresentado ao programa de Engenharia de Computação da PUC-Rio como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Augusto Cesar Espíndola Baffa

Rio de Janeiro
Dezembro de 2024

Resumo

Este trabalho descreve o desenvolvimento de um sistema automatizado para o cálculo da cota diária de fundos de criptomoedas, eliminando a necessidade de cálculos manuais por parte dos analistas. O sistema integra uma API REST escrita utilizando C#/.NET 8.0 com MongoDB para processar um arquivo JSON que contém as informações do portfólio de criptomoedas, incluindo suas respectivas cotações diárias. Após o processamento, o sistema gera um relatório detalhado, também em formato JSON, com as cotas diárias calculadas. O objetivo principal do projeto é aumentar a precisão, eficiência e escalabilidade do processo, minimizando a ocorrência de erros humanos e liberando os analistas para se concentrarem em atividades mais estratégicas e de maior valor agregado. Os resultados obtidos demonstram que o sistema foi capaz de reduzir significativamente o tempo de cálculo das cotas e eliminar erros decorrentes de operações manuais, garantindo maior confiabilidade nas informações disponibilizadas. Além disso, a integração com sistemas de gerenciamento de dados provou ser eficaz, facilitando a extração e a análise das cotas. A solução mostrou-se escalável e flexível, com potencial para suportar a inclusão de novos ativos no portfólio, o que é crucial em um ambiente de constante evolução, como o de criptomoedas.

Palavras-chave: Criptomoedas, Fundos

Abstract

This project describes the development of an automated system for calculating the daily net asset value (NAV) of cryptocurrency funds, eliminating the need for manual calculations by analysts. The system integrates a REST API built using C#/.NET 8.0 with MongoDB to process a JSON file containing the cryptocurrency portfolio information, including daily price quotes for each asset. After processing, the system generates detailed reports, also in JSON format, with the calculated daily NAVs. The main objective of the project is to enhance accuracy, efficiency, and scalability, minimizing the occurrence of human errors and allowing analysts to focus on more strategic, high-value tasks.

The results obtained demonstrate that the system significantly reduced the time required to calculate NAVs and eliminated errors caused by manual operations,

thereby ensuring greater reliability in the information provided. Additionally, the integration with data management systems proved effective, facilitating the extraction and analysis of NAVs. The solution was shown to be scalable and flexible, with the potential to support the inclusion of new assets in the portfolio, which is crucial in a rapidly evolving environment such as the cryptocurrency market.

Keywords: Cryptocurrencies, Portfolio

Sumário

1) Introdução

No atual cenário do mercado financeiro, a agilidade e a precisão no processamento de informações são cruciais para garantir a competitividade das empresas e a confiança dos investidores. O setor de criptomoedas, em particular, destaca-se por sua volatilidade e pelo grande volume de transações diárias, o que exige um monitoramento constante e detalhado dos portfólios de investimento (2). Diante dessa realidade, soluções automatizadas, como a desenvolvida neste trabalho, tornam-se essenciais para reduzir a complexidade das operações diárias, ao mesmo tempo em que aumentam a confiabilidade dos processos.

A automação de tarefas rotineiras, como o cálculo da cota diária de fundos de criptomoedas, é uma tendência crescente em diversas áreas do mercado financeiro (3). Com o avanço das tecnologias de programação e o uso de APIs REST, as empresas conseguem realizar integrações de sistemas de forma mais eficiente, garantindo que grandes volumes de dados sejam processados em tempo real e com alta precisão. Esse tipo de solução, além de economizar tempo e reduzir erros, contribui para uma maior transparência na gestão dos fundos, o que é altamente valorizado pelos cotistas e investidores.

Adicionalmente, a implementação de um banco de dados NoSQL, como o MongoDB, traz uma série de vantagens para o armazenamento e manipulação dos dados de criptomoedas, que geralmente apresentam grande variabilidade. Por ser um banco de dados orientado a documentos, o MongoDB permite uma estruturação mais flexível das informações, acomodando tanto dados de ativos tradicionais quanto os mais dinâmicos, como os relacionados às criptomoedas. Essa flexibilidade é especialmente relevante em um ambiente onde os preços variam constantemente e novas criptomoedas podem ser adicionadas ao portfólio com frequência(4).

Outro ponto importante a ser destacado é a capacidade do sistema de realizar os cálculos automaticamente em horários programados. Antes da automatização, os analistas precisavam dedicar grande parte do seu dia a esse processo, o que limitava o tempo disponível para a realização de análises mais profundas sobre o desempenho dos ativos. Com a automação, esses profissionais podem focar em decisões estratégicas que agreguem valor ao fundo, como ajustes na composição

do portfólio ou a identificação de oportunidades de mercado. Isso resulta em uma gestão mais eficiente e em ganhos de produtividade para a equipe como um todo (5).

A segurança (6) e (7) também é um fator primordial nesse tipo de implementação. Ao automatizar processos de cálculo e armazenar os dados em sistemas robustos, o risco de falhas humanas é significativamente reduzido (8). Além disso, as APIs permitem que os dados sejam acessados de maneira segura, com controle de permissões, o que garante que as informações dos fundos sejam mantidas confidenciais e protegidas contra acessos não autorizados (9). Portanto, além dos ganhos de eficiência e precisão, a solução também oferece um nível maior de segurança, um aspecto indispensável no mercado de ativos digitais (10).

2) Levantamento da Situação Atual

Atualmente, o sistema realiza o cálculo da cota do fundo de forma automatizada, representando um avanço significativo em relação ao método manual previamente utilizado. Esse aprimoramento elimina grande parte do trabalho repetitivo e reduz a margem de erro, proporcionando uma maior confiabilidade nos dados apresentados aos cotistas. A infraestrutura do sistema e a camada de business já estão completamente desenvolvidas, garantindo uma base sólida para o funcionamento da aplicação. O MongoDB foi escolhido como banco de dados por sua flexibilidade e escalabilidade, características essenciais para o armazenamento e recuperação de grandes volumes de dados, como os históricos de cotações e movimentações de ativos.

O sistema vai além de simplesmente calcular a cota do fundo de forma correta; ele também é capaz de gerar relatórios detalhados que oferecem uma visão mais completa do desempenho do fundo. Esses relatórios incluem informações críticas que facilitam a análise por parte dos gestores e analistas, permitindo uma tomada de decisão mais rápida e precisa. Na camada de *Controllers*, foram definidos três endpoints principais, que permitem uma interação eficiente com os dados. O primeiro endpoint é responsável por obter as informações do fundo em uma data específica, permitindo consultas pontuais e detalhadas. O segundo endpoint oferece a funcionalidade de recuperar os dados de todas as datas, útil para análises de desempenho histórico e comparações ao longo do tempo. Já o terceiro endpoint fornece métricas financeiras fundamentais, como GAV (Gross Asset

Value), NAV (Net Asset Value), GAVPS (Gross Asset Value per Share) e NAVPS (Net Asset Value per Share), métricas essenciais para a avaliação do valor total dos ativos e o desempenho por unidade do fundo.

Esses recursos não apenas automatizam o processo de cálculo, mas também aumentam a capacidade de análise do sistema, fornecendo informações cruciais de forma rápida e acessível. Essa integração entre a automação e a geração de relatórios possibilita uma gestão mais eficiente do fundo, oferecendo aos gestores ferramentas robustas para acompanhar o desempenho dos ativos e tomar decisões estratégicas com base em dados confiáveis. Além disso, o uso de uma arquitetura moderna com MongoDB e endpoints claros permite a fácil manutenção e expansão futura do sistema, garantindo que ele possa evoluir conforme as necessidades do fundo se tornem mais complexas e o volume de dados aumente.

A linguagem de programação utilizada no desenvolvimento deste sistema é o C#, uma escolha estratégica que se alinha com a robustez e a capacidade de integração que essa linguagem proporciona, especialmente em ambientes corporativos e na nuvem, como o Azure. O C# é amplamente reconhecido por sua versatilidade e eficiência na construção de aplicações de alto desempenho, sendo amplamente utilizado para soluções que exigem escalabilidade e integração com múltiplos serviços. Além disso, o C# oferece uma ampla gama de bibliotecas e frameworks que facilitam o desenvolvimento de APIs REST, tornando-o ideal para este projeto, que depende da comunicação eficiente entre o sistema e o banco de dados, além da geração de relatórios precisos.

Um dos principais benefícios de usar C# é a sua integração nativa com o ecossistema .NET, o que possibilita o desenvolvimento de soluções completas que podem ser facilmente hospedadas em plataformas de nuvem como o Azure. A robustez da linguagem e a vasta gama de ferramentas que ela oferece permitem ao sistema lidar com grandes volumes de dados e realizar cálculos complexos de maneira eficiente. Isso é especialmente importante no contexto deste projeto, onde a precisão é fundamental, dado que os cálculos de GAV, NAV, GAVPS e NAVPS precisam ser realizados com base em dados financeiros sensíveis, que incluem cotações de criptomoedas e saldos de portfólios.

Além da escolha da linguagem, o banco de dados MongoDB foi selecionado como uma solução ideal para armazenar e gerenciar os dados envolvidos no cálculo da

cota dos fundos de criptomoedas. MongoDB é um banco de dados NoSQL, conhecido por sua confiabilidade, flexibilidade e robustez, características que são essenciais ao lidar com dados financeiros que podem variar em volume e estrutura. Ao contrário de bancos de dados relacionais tradicionais, que podem ser mais rígidos em termos de esquema, o MongoDB permite uma estrutura mais flexível, o que é especialmente útil quando lidamos com informações de criptomoedas, cujas cotações e dados de mercado estão em constante mudança e crescimento.

A flexibilidade do MongoDB é um fator crucial neste projeto, já que permite que os dados sejam organizados em documentos JSON, o que facilita tanto o armazenamento quanto a recuperação rápida de informações para cálculos e relatórios. Isso é particularmente relevante para aplicações financeiras, onde os dados precisam ser acessados e manipulados com rapidez para garantir a atualidade das informações fornecidas aos cotistas. O MongoDB também é altamente escalável, permitindo que o sistema cresça conforme o número de fundos e transações aumenta, sem comprometer o desempenho.

A confiabilidade do MongoDB é outra vantagem importante, pois garante que os dados financeiros armazenados estejam seguros e acessíveis, o que é crucial ao lidar com informações sensíveis. A capacidade do MongoDB de lidar com transações financeiras de maneira eficiente e segura torna-o uma escolha natural para projetos como este, onde a precisão e a integridade dos dados são de suma importância. Além disso, o banco de dados oferece mecanismos de replicação e backups automáticos, o que garante a continuidade do serviço em caso de falhas, mantendo a integridade dos dados mesmo em cenários de grande demanda ou falhas de sistema.

Portanto, a combinação de C# e MongoDB proporciona uma solução tecnológica robusta, escalável e altamente eficiente para o cálculo da cota dos fundos de criptomoedas. O uso do C# garante que o sistema seja ágil, seguro e fácil de manter, enquanto o MongoDB oferece uma base de dados flexível e confiável, capaz de suportar o crescimento exponencial e a complexidade inerentes ao mercado de criptomoedas. Esses fatores tornam o sistema não apenas uma ferramenta poderosa para automatizar o cálculo da cota dos fundos, mas também uma solução preparada para futuras expansões e adaptações, conforme o mercado e as necessidades dos fundos evoluam.

Em resumo, o sistema automatizado desenvolvido não apenas otimiza o processo de cálculo das cotas dos fundos de criptomoedas, mas também traz uma série de melhorias em termos de precisão, eficiência e confiabilidade. Ao eliminar a necessidade de cálculos manuais, o sistema reduz significativamente a margem de erro, que é comum em processos conduzidos manualmente, especialmente em situações que envolvem dados financeiros complexos e flutuantes como as cotações de criptomoedas. Dessa forma, o sistema garante que os valores calculados, como GAV, NAV, GAVPS e NAVPS, sejam precisos e fornecidos de maneira consistente aos cotistas, contribuindo para a transparência e a confiança nas operações do fundo.

Além disso, a automatização deste processo permite que os analistas possam se concentrar em atividades mais estratégicas e de maior valor agregado, como a análise de desempenho do fundo, o desenvolvimento de novas estratégias de investimento e a otimização do portfólio. Essas tarefas exigem um nível mais elevado de atenção e especialização, que agora podem ser priorizadas, uma vez que o sistema assume a responsabilidade pelas rotinas operacionais diárias, como o cálculo das cotas e a geração de relatórios. Dessa forma, o tempo dos analistas é melhor aproveitado em atividades que realmente podem impactar positivamente os resultados do fundo, trazendo um diferencial competitivo para a gestão de ativos.

Outro benefício importante proporcionado pelo sistema automatizado é a escalabilidade. À medida que o fundo de criptomoedas cresce e o volume de transações aumenta, o sistema está preparado para lidar com esse crescimento sem comprometer o desempenho ou a precisão dos cálculos. A integração com o MongoDB, por exemplo, permite o armazenamento eficiente de grandes volumes de dados, garantindo que a aplicação continue funcionando de maneira fluida, mesmo com a expansão do portfólio de ativos. Isso é fundamental em um mercado dinâmico como o de criptomoedas, onde o ambiente muda rapidamente e novas oportunidades surgem constantemente.

Em resumo, a implementação deste sistema não só facilita o dia a dia operacional, como também melhora a qualidade dos dados financeiros gerados, traz maior flexibilidade para acompanhar o crescimento do fundo e libera os analistas para focarem em atividades mais críticas para o sucesso da operação. A combinação

desses fatores faz com que a adoção dessa solução tecnológica represente uma vantagem significativa para a gestão eficiente e eficaz dos fundos de criptomoedas.

3) Objetivos do trabalho

Nesta Seção são apresentados os objetivos e os recursos identificados durante o levantamento do projeto e que foram implementados no protótipo.

3.1 Objetivos Específicos

A solução proposta tem como objetivo principal a automatização do cálculo diário das cotas dos fundos de criptomoedas, com o intuito de eliminar a necessidade de cálculos manuais e reduzir significativamente o risco de erros humanos. A automatização deste processo não só traz mais agilidade, mas também uma grande eficiência no gerenciamento das cotas, garantindo que as análises sejam realizadas de forma mais precisa e rápida. Dessa forma, os objetivos específicos do projeto são:

Automatização do Processo: O sistema proposto será desenvolvido para calcular automaticamente a cota dos fundos de criptomoedas com base em dados estruturados fornecidos em arquivos JSON. Essa automatização permitirá que os analistas não precisem mais executar cálculos manuais, que, por serem repetitivos e propensos a erros, tomam um tempo considerável do trabalho diário. A solução trará uma abordagem mais eficiente e moderna para lidar com a precificação de criptomoedas, oferecendo resultados de forma quase instantânea. O uso de tecnologia de ponta, como APIs REST, será fundamental para garantir que o processo de cálculo seja integrado de maneira simples e direta aos sistemas já existentes, sem interrupções ou complicações.

Precisão e Confiabilidade: Outro objetivo essencial é garantir que os cálculos de cotas sejam não apenas rápidos, mas também extremamente precisos e confiáveis. A precisão nos cálculos é crucial, uma vez que qualquer erro poderia afetar substancialmente as decisões financeiras tomadas com base nas informações extraídas do sistema. Para alcançar esse nível de precisão, o sistema será projetado para trabalhar com dados verificáveis e consistentes, utilizando processos matemáticos avançados e a validação dos dados de entrada e saída. Além disso, a confiabilidade dos dados armazenados será assegurada através de boas práticas de backup, recuperação de dados e auditoria contínua, prevenindo qualquer tipo de falha ou corrupção de informações.

Eficiência: A eficiência do sistema será um dos pilares centrais do desenvolvimento, com o objetivo de reduzir significativamente o tempo e o esforço necessários para realizar o cálculo das cotas. Atualmente, o processo manual é demorado e exige uma grande quantidade de recursos humanos, que poderiam ser melhor alocados em atividades que agreguem mais valor à organização. Ao automatizar esses cálculos, o tempo gasto em cada operação será reduzido drasticamente, permitindo que os analistas se concentrem em outras tarefas de maior complexidade e relevância, como a análise de tendências de mercado, gestão de risco e tomada de decisões estratégicas. Essa mudança no fluxo de trabalho não só otimiza o tempo, mas também contribui para um ambiente mais produtivo e focado nas áreas mais críticas do negócio.

Escalabilidade e Manutenção: Por fim, a solução será desenvolvida com foco na escalabilidade e na facilidade de manutenção, para que possa ser adaptada conforme o crescimento do volume de dados e a evolução das necessidades do negócio. A infraestrutura proposta utilizará tecnologias como APIs REST e o banco de dados MongoDB, garantindo que o sistema possa lidar com grandes volumes de dados de maneira eficiente e sem perda de performance. Além disso, a arquitetura será planejada para permitir manutenções e atualizações de forma contínua, sem interrupções nos serviços. Isso proporcionará uma grande flexibilidade para a organização, que poderá ajustar o sistema conforme novas criptomoedas e novas exigências do mercado surjam, sem a necessidade de grandes revisões ou retrabalho.

3.2 Componentes do protótipo

O sistema será composto pelos seguintes componentes:

Entrada de Dados: O primeiro passo do processo é o recebimento de um arquivo JSON que contém o portfólio de criptomoedas, incluindo as cotações dos ativos e seus respectivos saldos. Este arquivo JSON servirá como a fonte primária de dados para o cálculo da cota do fundo, e será fundamental que os dados inseridos no arquivo estejam corretos e atualizados. O sistema será projetado para processar automaticamente este arquivo, garantindo que todos os dados necessários estejam acessíveis para os cálculos subsequentes. A entrada de dados precisa ser simples e eficiente, permitindo que os analistas ou outros responsáveis pela entrada de dados forneçam os arquivos de maneira fácil e rápida, sem complicações adicionais.

Processamento: Uma vez que o arquivo JSON é recebido, o próximo componente do sistema entra em ação. Utilizando C#/.NET 8.0 e uma API REST, o sistema processará os dados fornecidos no arquivo e calculará automaticamente a cota do fundo de criptomoedas em horários específicos. A escolha do C# e .NET 8.0 para o processamento é baseada na sua robustez e eficiência em lidar com grandes volumes de dados e a necessidade de realizar cálculos financeiros complexos de forma precisa e rápida. A integração com uma API REST permitirá que o sistema seja acessado de maneira simples e flexível, possibilitando a comunicação com outros sistemas que possam ser necessários para o processo de cálculo. Além disso, a execução em horários pré-definidos trará maior controle e previsibilidade, garantindo que os cálculos sejam realizados em momentos específicos, conforme exigido.

Armazenamento: O banco de dados MongoDB será utilizado para armazenar de maneira segura e acessível todos os resultados dos cálculos realizados. MongoDB foi escolhido devido à sua flexibilidade e escalabilidade, permitindo que grandes volumes de dados sejam armazenados e acessados de forma eficiente. Além disso, a utilização de um banco de dados NoSQL como o MongoDB facilita o armazenamento de dados estruturados de maneira dinâmica, sendo ideal para a natureza variável dos dados de criptomoedas. O banco de dados também garantirá que os resultados sejam facilmente acessíveis para os analistas, cotistas e administradores, permitindo que consultem os dados sempre que necessário, com segurança e rapidez.

Saída de Dados: O sistema gerará um novo arquivo JSON com as cotas calculadas, que será disponibilizado para os cotistas e administradores do fundo. Esse arquivo conterá todas as informações necessárias sobre o fundo, incluindo a cota calculada, para que os stakeholders possam acompanhar o desempenho do fundo de criptomoedas de maneira transparente e precisa. A saída de dados será organizada e formatada de forma que seja facilmente compreendida pelos usuários, proporcionando uma comunicação clara e eficaz entre os responsáveis pelo fundo e seus investidores.

Usuários/Programadores e Situações que se Deseja Apoiar:

Analistas de Investimentos: O sistema visa reduzir significativamente a carga de trabalho dos analistas de investimentos, automatizando o cálculo das cotas, que, no modelo atual, é feito manualmente. Ao eliminar esse processo repetitivo, o sistema permite que os analistas se concentrem em outras atividades de maior complexidade e valor agregado, como análise de mercado, projeções e

estratégias de investimento. Além disso, a automatização minimiza erros manuais, que podem ocorrer durante o processo de cálculo, melhorando a confiabilidade dos resultados.

Cotistas: A solução também beneficiará diretamente os cotistas, que terão acesso a dados precisos e atualizados sobre a cota do fundo, de forma automática e sem atrasos. A transparência fornecida por esse sistema aumentará a confiança dos investidores no fundo, visto que eles poderão acompanhar o desempenho dos ativos de forma mais eficiente e em tempo real. A disponibilização das cotas em tempo adequado ajudará os cotistas a tomarem decisões mais informadas sobre seus investimentos, permitindo que ajustem suas estratégias conforme necessário.

Desenvolvedores/Programadores: Para os desenvolvedores e programadores envolvidos na manutenção e evolução do sistema, a solução proporcionará uma base sólida e bem estruturada, com um código limpo e bem documentado. Isso garantirá que futuras melhorias ou ajustes possam ser implementados com facilidade, sem a necessidade de reescrever grandes partes do código. A utilização de tecnologias modernas e robustas, como C#/.NET 8.0, APIs REST e MongoDB, também facilitará o processo de manutenção e garantirá a alta performance e escalabilidade do sistema.

Diferenças ou Avanços em Relação às Soluções e Técnicas Existentes

A solução proposta se destaca das abordagens tradicionais de várias maneiras, trazendo vantagens claras em relação aos métodos manuais e outras soluções existentes no mercado:

Automatização Completa: Diferente das soluções tradicionais, que ainda dependem de cálculos manuais ou semi-automatizados, esta solução realiza todo o processo de cálculo automaticamente, utilizando uma API REST para garantir que os cálculos sejam executados em horários predefinidos. Isso elimina a necessidade de intervenção manual em todas as etapas, tornando o processo mais eficiente e livre de erros humanos. A automatização também contribui para a melhoria da produtividade da equipe, uma vez que os analistas não precisarão mais dedicar tempo a cálculos repetitivos e demorados.

Redução de Erros: A solução automatizada reduz substancialmente o risco de erros humanos, que são comuns em processos manuais. Ao garantir que os cálculos sejam realizados de forma precisa e sem a intervenção direta dos analistas, a precisão dos dados aumentará, o que é essencial em um ambiente financeiro onde pequenas falhas podem ter grandes impactos. A consistência nos

resultados gerados também garantirá maior confiança por parte dos cotistas e administradores do fundo.

Escalabilidade e Flexibilidade: Uma das principais vantagens desta solução é a sua escalabilidade e flexibilidade. O uso de tecnologias de nuvem e de um banco de dados robusto como MongoDB permite que o sistema seja facilmente expandido conforme a demanda aumente. Além disso, a flexibilidade do sistema permitirá que ele seja adaptado para incorporar novas criptomoedas ou ajustar os cálculos conforme o mercado evolui, sem necessidade de grandes revisões no código base.

Integração de Tecnologias Modernas: A utilização das tecnologias mais modernas e adequadas para o cenário atual, como C# e .NET 8.0, mostra um avanço significativo em relação às soluções tradicionais, que muitas vezes ainda utilizam sistemas legados ou tecnologias menos adequadas para lidar com o volume e a complexidade dos dados financeiros. C#/.NET 8.0 oferece uma plataforma robusta, segura e altamente performática para o desenvolvimento de sistemas financeiros complexos, tornando o sistema proposto mais eficiente, seguro e preparado para futuras inovações.

3.3 Recursos adicionais

REST API: Desenvolvimento e configuração das funções que realizarão o processamento e cálculo das cotas, além de fornecer os dados de ativos e preços na carteira.

Banco de Dados MongoDB: Configuração e implementação do banco de dados, incluindo a criação das tabelas necessárias e a inserção dos dados.

Processamento do Arquivo JSON: Implementação da lógica de leitura, processamento e cálculo das cotas a partir do arquivo JSON de entrada.

Geração do Arquivo JSON de Saída: Desenvolvimento da funcionalidade para gerar o arquivo JSON contendo as cotas calculadas.

Documentação Completa: Fornecimento de uma documentação detalhada do sistema, incluindo guias de uso e manutenção.

4) Escopo e Detalhamento da Solução

Esta seção descreve o escopo e os detalhes da solução que será apresentada à banca, destacando o que foi efetivamente implementado e os benefícios em relação às soluções existentes.

Nas partes a seguir, tratamos do diagrama do Minimundo, Casos de Uso e o diagrama de classes:

4.1 Diagrama de Mini Mundo

Aqui, podemos ver o Diagrama de Minimundo(Figura 1). Nele, podemos ver que o sistema Portfolio Manager API, possui 3 função primordiais, que são:

1) Calcular a cota do fundo com base em informações de balance(posição) e preços em uma determinada data.
2) Capturar informações da carteira do fundo na base de dados para qualquer data(incluindo posições de preço).
3) Capturar informações da carteira do fundo em determinada data(incluindo posições e preços).

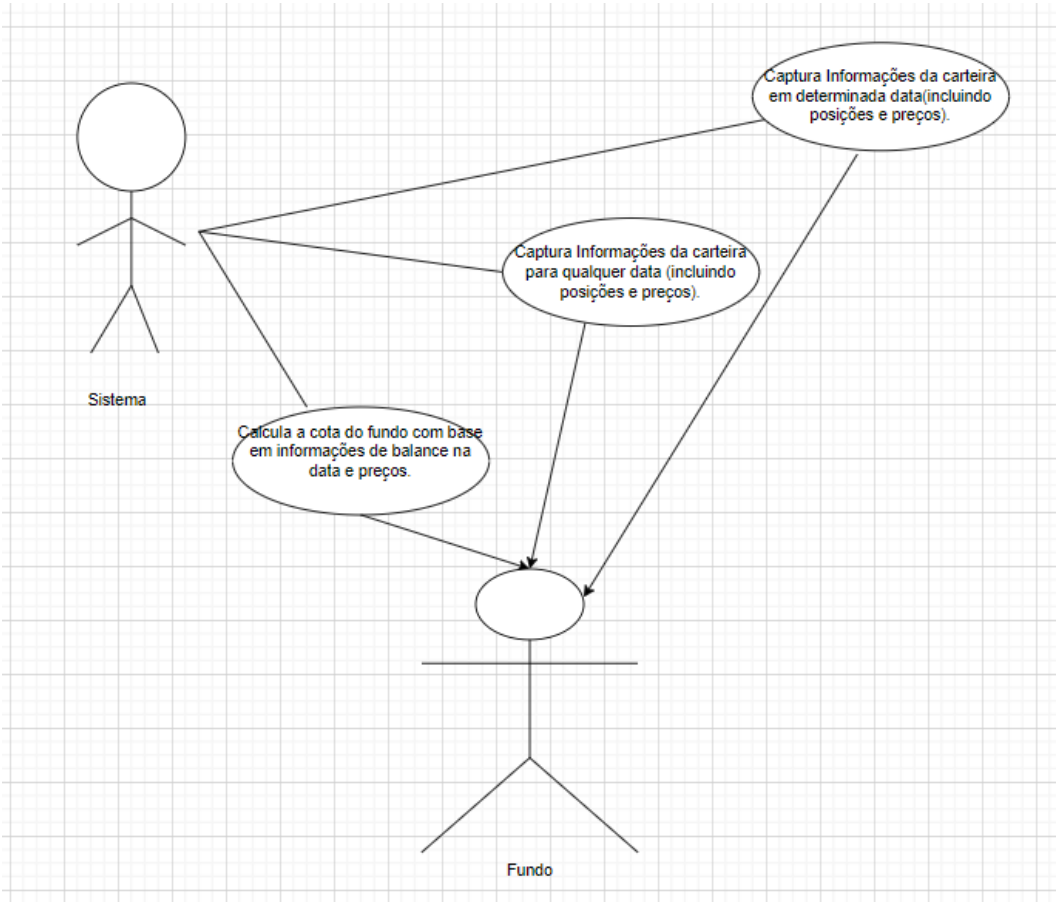

Figura 1: Diagrama Minimundo

4.2 Casos de Uso

Os casos de uso identificados para o desenvolvimento são listados abaixo.

Objetivo:	Capturar da carteira do fundo em determinada data(incluindo posições e preços).
Requisitos:	Está relacionado ao requisito de capturar os preços e as posições para o cálculo da cota.
Atores:	Sistema, Fundo
Prioridade:	-
Pré-condições:	Necessário que as informações sejam disponibilizadas corretamente pelo Banco de Dados.
Freqüência de uso:	Uso diário (Cota calculada de manhã).
Criticalidade:	-

Condição de Entrada:	A cota é calculada por uma função disparada em uma data.
Fluxo Principal:	Sistema é chamado via endpoint para uma data e retorna os valores dos preços e as posições dos ativos em carteira.
Fluxo Alternativo:	-
Pós-condições:	-
Regras de negócio:	-

Objetivo:	Capturar informações da carteira para todas as data (incluindo posições e preços).
Requisitos:	Está relacionado ao requisito de capturar os preços e as posições para o cálculo da cota.
Atores:	Sistema, Fundo
Prioridade:	-
Pré-condições:	Necessário que as informações sejam disponibilizadas corretamente pelo Banco de Dados.
Freqüência de uso:	Pode haver necessidade de exibição do histórico de posições.
Criticalidade:	-
Condição de Entrada:	A cota é calculada por uma função disparada.
Fluxo Principal:	Sistema é chamado via endpoint para e retorna o histórico dos valores dos preços e as posições dos ativos em carteira.
Fluxo Alternativo:	-
Pós-condições:	-
Regras de negócio:	-

Objetivo:	Calcular a cota do fundo com base em informações de balance na e preços.

Requisitos:	Está relacionado ao requisito de fazer o cálculo da cota em si
Atores:	Sistema, fundo
Prioridade:	-
Pré-condições:	Necessário que as informações sejam disponibilizadas corretamente pelo banco e que não existam erros no código.
Freqüência de uso:	Uso diário (Cota calculada de manhã).
Criticalidade:	-
Condição de Entrada:	A cota é calculada por uma função disparada em um horário.
Fluxo Principal:	Sistema utiliza preços e balanços capturados e retorna o valor da cota e sua variação diária em um JSON.
Fluxo Alternativo:	-
Pós-condições:	-
Regras de negócio:	-

4.3) Diagrama de Classes

A seguir, na figura 2, podemos ver o diagrama de classes da camada de Model. Esse diagrama inclui a entidade que é salva diariamente na base de dados e que contém as informações sobre o portfólio (PortfolioInfo, ProductInfo, Price, Balance) e que podem ser obtidas através dos endpoints “GetPortfolioByDate” e “GetPortfolioByDate” e as informações sobre o retorno da chamada “CalculateByPortfolioDate” (PortfolioReport).

Detalhamento das Classes de Data Transfer e DTOs

As classes de **Data Transfer** ou **DTOs** (Data Transfer Objects) são usadas para transferir dados entre camadas do sistema, especialmente para facilitar a comunicação entre o controlador, os repositórios e as classes de processamento. No seu caso, temos a classe PortfolioInfo e suas sub-classes internas, como ProductInfo, Price, e Balance, que desempenham um papel essencial na representação dos dados de portfólio.

1. Classe PortfolioInfo

A PortfolioInfo é uma classe central no sistema, responsável por encapsular as informações do portfólio. Essa classe contém todos os dados necessários para gerar um **PortfolioReport**, que é o relatório final de portfólio gerado pelo processo.

Estrutura e Responsabilidades

- **PortfolioInfo**: Contém todos os dados relacionados ao portfólio de um usuário ou fundo de investimentos em um determinado momento no tempo.
    - Dentro dessa classe, temos a **ProductInfo**, que encapsula detalhes dos produtos financeiros no portfólio.
    - A **ProductInfo**, por sua vez, possui as sub-classes **Price** e **Balance**, que contêm informações sobre os preços dos produtos e os saldos no portfólio.

Campos da Classe PortfolioInfo

- **ProductInfo productInfo**: Essa propriedade encapsula os detalhes dos produtos no portfólio. A classe ProductInfo contém informações cruciais para o cálculo, como preços e saldos.

2. Classe ProductInfo

A ProductInfo é uma classe que representa um produto financeiro no portfólio. Essa classe fornece detalhes sobre o produto que são necessários para calcular a performance do portfólio e gerar o relatório.

Estrutura e Responsabilidades

- **ProductInfo**: A principal função dessa classe é fornecer as informações sobre os produtos financeiros que compõem o portfólio. Dentro dela, existem detalhes como **Price** e **Balance** que descrevem, respectivamente, os preços e os saldos dos produtos financeiros.

Campos da Classe ProductInfo

- **Price price**: Esta classe contém os preços associados aos produtos financeiros. Os preços podem variar ao longo do tempo e são fundamentais para calcular a valorização do portfólio e o seu retorno.
- **Balance balance**: A classe Balance contém informações sobre o saldo de cada produto no portfólio, ou seja, a quantidade de cada ativo ou produto presente no portfólio.

3. Classe Price

A classe Price armazena os preços dos produtos financeiros no portfólio. Os preços são um dos principais determinantes na avaliação do portfólio, pois influenciam diretamente no cálculo de rentabilidade, avaliação de ativos e em outros cálculos financeiros.

Estrutura e Responsabilidades

- **Price**: Armazena o preço de um produto financeiro em um momento específico. Este preço será usado para calcular o valor total do produto no portfólio.

Campos da Classe Price

- **Valor do preço**: Este campo representa o valor monetário do produto no momento da consulta. O preço pode ser utilizado para calcular o ganho ou perda do produto financeiro ao longo do tempo.

4. Classe Balance

A classe Balance contém informações sobre os saldos de cada produto no portfólio, ou seja, a quantidade de ativos ou valores de cada tipo de produto no portfólio. Juntamente com o preço, o saldo é essencial para calcular o valor total do produto no portfólio.

Estrutura e Responsabilidades

- **Balance**: Esta classe encapsula a quantidade ou valor de cada ativo ou produto no portfólio, permitindo o cálculo da exposição total a um determinado ativo ou classe de ativos.

Campos da Classe Balance

- **Quantidade de ativos**: Esse campo armazena o número de unidades do produto no portfólio. Ele é utilizado para calcular o valor total de cada produto, multiplicando o preço pela quantidade.

Como as Classes se Relacionam

Agora, vamos analisar como essas classes se interagem e como os dados são transferidos ao longo do fluxo.

1. **PortfolioInfo**: Representa o portfólio de um fundo ou usuário em uma data específica. Ela contém a classe **ProductInfo**, que possui as sub-classes **Price** e **Balance**. O PortfolioInfo é a classe que será utilizada pelos métodos do controlador (PortfolioController) e pelos repositórios para gerar relatórios.
2. **ProductInfo**: Cada instância de ProductInfo contém detalhes sobre um produto financeiro no portfólio, incluindo o preço atual do produto (em

Price) e a quantidade (em Balance). Essas informações são essenciais para calcular o valor total dos produtos e o portfólio como um todo.

3. **Price e Balance**: Essas classes são fundamentais para calcular o valor total de um ativo no portfólio. O preço de um produto é multiplicado pela quantidade disponível no portfólio para obter o valor total do ativo, que contribui para a avaliação do portfólio. Juntas, essas classes fornecem a base para cálculos de valorização, alocação de ativos e rentabilidade.

Fluxo de Dados no Sistema

- O controlador PortfolioController recebe a solicitação para calcular o portfólio em uma data específica.
- O controlador consulta o PortfolioRepository para obter as instâncias de PortfolioInfo, que contêm as sub-classes ProductInfo, Price e Balance.
- Com os dados recebidos, o controlador chama a classe ProcessProduct para calcular o valor total do portfólio e gerar o relatório. O valor total é calculado usando os preços e saldos dos produtos.
- A classe ProcessProduct também invoca a classe ProcessAdmFee para calcular as taxas administrativas associadas ao portfólio, aplicando-as conforme necessário.
- O relatório gerado (PortfolioReport) é então retornado ao usuário ou ao sistema solicitante.

Conclusão sobre as Classes de Data Transfer

As classes PortfolioInfo, ProductInfo, Price e Balance desempenham um papel vital na representação e no processamento dos dados de portfólio. Elas são projetadas para serem simples, mas potentes, fornecendo uma estrutura clara para armazenar e processar as informações financeiras do portfólio. A interdependência entre essas classes garante que todas as informações relevantes sejam consideradas durante os cálculos e na geração do relatório final. Essas classes são essenciais para o funcionamento eficiente do sistema, garantindo que os cálculos sejam realizados com precisão e que as informações do portfólio sejam apresentadas de forma clara e compreensível.

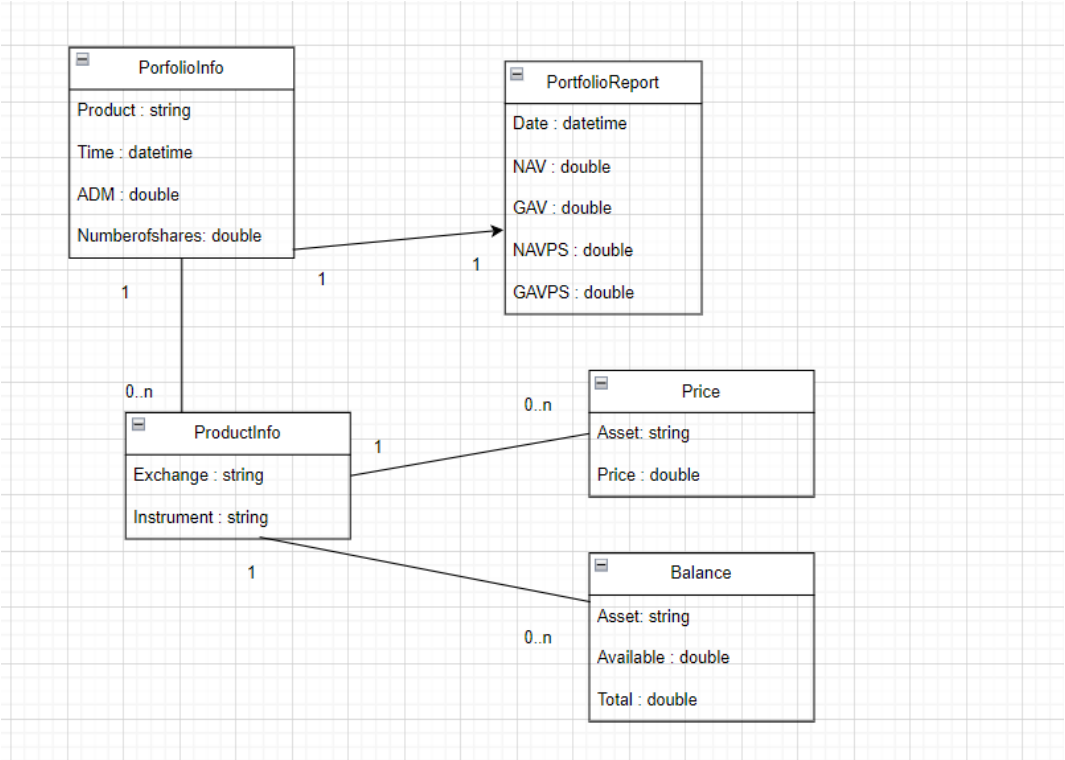

Figura 2: Diagrama de Cases da camada de model

Na figura 3 podemos ver os 3 fluxos possíveis gerados pelos 3 endpoints disponíveis de forma detalhada do ponto de vista das classes envolvidas e dos métodos.

Detalhamento da figura 3

Este sistema foi projetado para calcular e gerar relatórios detalhados sobre o portfólio de investimentos, considerando informações como alocação de ativos, taxas administrativas, rentabilidade e outros aspectos financeiros. O fluxo principal envolve a interação entre diferentes classes, cada uma com responsabilidades específicas, desde o controle do processo até o cálculo das taxas e a geração do relatório final. Abaixo, descreverei o papel de cada classe e a interação entre elas.

1. Classe: PortfolioController

A classe PortfolioController atua como o ponto de entrada do sistema, orquestrando a execução do processo de cálculo do portfólio e a geração do relatório. É responsável por invocar os métodos necessários em outras classes

para realizar os cálculos e processar os dados do portfólio. A seguir estão os métodos dessa classe:

Métodos da classe PortfolioController

1. **CalculatePortfolioByDate(date: datetime) : PortfolioReport**:
    - **Descrição**: Este é o principal método do controlador, que é chamado quando o sistema precisa calcular o portfólio para uma data específica. O método recebe a data de execução (date) e inicia o processo de cálculo, chamando outras classes para obter os dados necessários e calcular os valores.
    - **Fluxo**:
        - O método chama a classe ProcessProduct, que contém a lógica de processamento do portfólio, passando a data como parâmetro.
        - O controlador recebe o relatório gerado (um objeto PortfolioReport) e o retorna como resposta.
2. **GetPortfolioByDate(date: datetime) : PortfolioInfo**:
    - **Descrição**: Esse método é utilizado para obter as informações detalhadas sobre o portfólio em uma data específica. Ele acessa dados no repositório e retorna um objeto PortfolioInfo, que contém os dados relevantes para o cálculo do portfólio.
3. **GetPortfolioAllDates() : List<PortfolioInfo>**:
    - **Descrição**: Este método retorna uma lista de objetos PortfolioInfo com as informações sobre o portfólio em todas as datas disponíveis. Ele pode ser utilizado para consultas históricas ou análises de desempenho ao longo do tempo.

2. Classe: PortfolioRepository

A classe PortfolioRepository é responsável pela interação com a base de dados, realizando consultas no MongoDB para obter as informações necessárias sobre o portfólio. Ela fornece métodos para acessar dados de portfólio para uma data específica ou para todas as datas.

Métodos da classe PortfolioRepository

1. **GetByDate(date: datetime) : PortfolioInfo**:
    - **Descrição**: Este método é responsável por retornar as informações detalhadas de um portfólio para uma data específica. Ele consulta o banco de dados MongoDB e retorna os dados como um objeto PortfolioInfo.

2. **GetAll(date: datetime) : List<PortfolioInfo>**:
    - **Descrição**: Este método retorna uma lista de objetos PortfolioInfo com as informações sobre o portfólio para todas as datas disponíveis. Ele realiza a consulta no banco de dados MongoDB e retorna todos os registros de portfólio.

3. Classe: ProcessProduct

A classe ProcessProduct é responsável pelo processamento de dados financeiros e pela geração do relatório do portfólio. Ela realiza os cálculos necessários para determinar o valor e as características do portfólio em um determinado momento.

Métodos da classe ProcessProduct

1. **Process(executionTime: datetime) : PortfolioReport**:
    - **Descrição**: Esse método é o principal da classe ProcessProduct e é chamado pela classe PortfolioController. Ele realiza o processamento dos dados dos produtos financeiros (ativos, fundos, ações, etc.) para gerar um relatório do portfólio em um momento específico.
    - **Fluxo**:
        - O método recebe a data de execução (executionTime) e usa essa data para consultar os dados do portfólio e calcular seu valor, rentabilidade e alocação.
        - Após processar os dados, o método retorna um PortfolioReport, que contém todos os detalhes do portfólio calculado.
    - **Interação com a classe ProcessAdmFee**: Dentro do método Process, a classe ProcessProduct chama o método CalculateAndReturnAdmDocument da classe ProcessAdmFee para calcular as taxas administrativas que devem ser aplicadas ao portfólio.

4. Classe: ProcessAdmFee

A classe ProcessAdmFee é responsável por calcular a taxa administrativa aplicada ao portfólio. As taxas administrativas são comuns em fundos de investimento e podem ser aplicadas de diferentes formas, dependendo do valor ou rentabilidade do portfólio.

Métodos da classe ProcessAdmFee

1. **CalculateAndReturnAdmDocument(portfolioInfo: PortfolioInfo, firstPortfolioInfo: PortfolioInfo) : PortfolioReport**:
    - **Descrição**: Este método é responsável pelo cálculo da taxa administrativa com base no portfólio atual (portfolioInfo) e no portfólio inicial ou de referência (firstPortfolioInfo). Ele calcula a taxa administrativa e gera um relatório do portfólio com essa taxa aplicada.
    - **Fluxo**:
        - O método compara os dois objetos PortfolioInfo para determinar a rentabilidade ou variação do valor do portfólio.
        - Com base nessa comparação, a taxa administrativa é calculada e aplicada.
        - O método retorna um PortfolioReport que inclui o valor da taxa administrativa aplicada ao portfólio.

Interação entre as Classes

1. **PortfolioController e PortfolioRepository**:
    - Quando o método CalculatePortfolioByDate da PortfolioController é chamado, ele pode precisar acessar as informações sobre o portfólio para a data fornecida. Isso é feito chamando os métodos GetByDate ou GetAll da PortfolioRepository, que fazem as consultas ao banco de dados MongoDB.
2. **PortfolioController e ProcessProduct**:
    - Após obter os dados do portfólio, o controlador chama o método Process da ProcessProduct para processar esses dados e gerar o relatório do portfólio.
3. **ProcessProduct e ProcessAdmFee**:
    - Dentro do método Process, a classe ProcessProduct chama o método CalculateAndReturnAdmDocument da ProcessAdmFee para calcular a taxa administrativa. Essa taxa é aplicada ao portfólio antes de gerar o relatório final.
4. **ProcessProduct e o Geração do PortfolioReport**:
    - O PortfolioReport gerado pelo método Process da ProcessProduct inclui todas as informações sobre o portfólio, como valor, rentabilidade, alocação e a taxa administrativa aplicada. Esse

relatório é então retornado ao controlador ou utilizado para outras análises.

Fluxo Completo

1. O usuário ou sistema solicita o cálculo do portfólio em uma data específica, chamando o método CalculatePortfolioByDate da PortfolioController.
2. A PortfolioController consulta o repositório para obter as informações sobre o portfólio para a data fornecida, utilizando os métodos GetByDate ou GetAll da PortfolioRepository.
3. A PortfolioController chama o método Process da ProcessProduct, passando a data de execução (executionTime).
4. A ProcessProduct processa os dados do portfólio e chama o método CalculateAndReturnAdmDocument da ProcessAdmFee para calcular e aplicar a taxa administrativa.
5. O PortfolioReport final, que inclui todos os dados processados e as taxas aplicadas, é retornado pela ProcessProduct e enviado de volta ao controlador.
6. O PortfolioController retorna o PortfolioReport final para o usuário ou sistema.

Conclusão

O sistema descrito é um conjunto bem estruturado de classes, com responsabilidades bem definidas. A interação entre as classes PortfolioController, PortfolioRepository, ProcessProduct e ProcessAdmFee permite calcular de forma eficiente e detalhada o valor do portfólio, incluindo o impacto das taxas administrativas. O design modular e a separação de responsabilidades entre as classes tornam o sistema fácil de manter e escalar, permitindo a inclusão de novos cálculos e funcionalidades conforme necessário

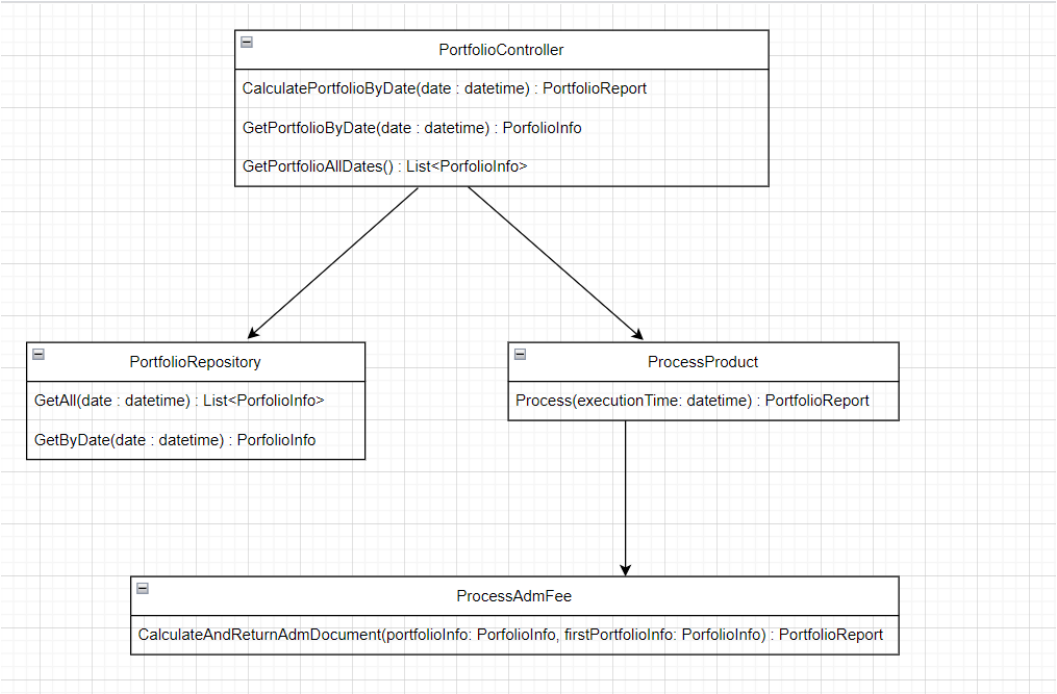

Figura 3: Diagrama de Cases das camadas Controller, Business e Repository

Relatório Detalhado das Classes e Fluxo de Cálculos do Portfólio

Este relatório detalha as classes envolvidas no cálculo e geração de relatórios de um portfólio de criptomoedas. O sistema segue uma arquitetura orientada a objetos, em que cada classe tem uma responsabilidade específica, colaborando com as demais para fornecer uma visão clara e precisa do portfólio e de seus ativos digitais. A dinâmica de criptos pode ser mais volátil e exige uma atualização constante de preços e saldos para garantir a precisão dos cálculos.

1. PortfolioController: Controlador Central do Sistema

A classe PortfolioController é o ponto de entrada principal para a lógica de controle do sistema. Ela gerencia a execução das operações relacionadas ao cálculo do portfólio e à obtenção de relatórios. Seu papel é orquestrar as interações entre os repositórios de dados e as classes de processamento para fornecer os resultados esperados.

Métodos Importantes da Classe PortfolioController

- **CalculatePortfolioByDate(date: datetime) : PortfolioReport**: Este método calcula o **valor total do portfólio de criptomoedas** para uma data específica, levando em consideração os preços de mercado das criptos e seus saldos no portfólio.

- **GetPortfolioByDate(date: datetime) : PortfolioInfo**: Este método retorna as informações detalhadas do portfólio para a data solicitada, como o saldo de cada cripto ativo e seus respectivos preços.
- **GetPortfolioAllDates() : List<PortfolioInfo>**: Este método retorna todas as informações históricas do portfólio, permitindo analisar seu desempenho ao longo do tempo.

2. PortfolioRepository: Acesso ao Banco de Dados

A classe PortfolioRepository é responsável por fazer a interação com o banco de dados (neste caso, provavelmente MongoDB) para recuperar os dados relacionados ao portfólio. Ela contém métodos que buscam os dados do portfólio para uma data específica ou para todas as datas disponíveis.

Métodos Importantes da Classe PortfolioRepository

- **GetAll(date: datetime) : List<PortfolioInfo>**: Esse método retorna todas as informações de portfólio para uma data específica, incluindo todos os saldos de criptomoedas e seus respectivos preços de mercado na data solicitada.
- **GetByDate(date: datetime) : PortfolioInfo**: Retorna informações detalhadas do portfólio para uma data específica, como o saldo de cada cripto ativo e o preço atual de cada um.

Esses métodos são chamados pelos métodos do PortfolioController para obter os dados necessários para os cálculos.

3. ProcessProduct: Processamento dos Ativos (Criptomoedas)

A classe ProcessProduct tem como objetivo realizar os cálculos necessários para determinar o valor de mercado de cada cripto ativo no portfólio. Ela é chamada pelo método CalculatePortfolioByDate da classe PortfolioController e faz o cálculo do valor do portfólio para uma data específica.

Método Importante da Classe ProcessProduct

- **Process(executionTime: datetime) : PortfolioReport**: Esse método realiza os cálculos necessários para calcular o valor do portfólio de criptomoedas na data solicitada. Ele usa as informações de preço das criptos e o saldo de cada cripto ativo para gerar o **relatório final**.

4. ProcessAdmFee: Cálculo de Taxas Administrativas

A classe ProcessAdmFee está diretamente envolvida no cálculo das taxas administrativas aplicáveis ao portfólio de criptomoedas. Isso é especialmente relevante em portfólios de criptos, onde as taxas de transação ou de custódia podem ter um impacto significativo nos resultados.

Método Importante da Classe ProcessAdmFee

- **CalculateAndReturnAdmDocument(portfolioInfo: PortfolioInfo, firstPortfolioInfo: PortfolioInfo) : PortfolioReport**: Este método calcula as taxas administrativas associadas ao portfólio de criptomoedas, considerando as taxas de transação e outros custos envolvidos na manutenção do portfólio.

5. PortfolioInfo: Estrutura de Dados do Portfólio

A classe PortfolioInfo contém todas as informações relacionadas ao portfólio de criptomoedas, incluindo os saldos das criptos e os preços de mercado desses ativos. Ela é uma das classes principais para geração do relatório de portfólio.

Campos Importantes da Classe PortfolioInfo

- **ProductInfo productInfo**: Este campo armazena as informações sobre as criptomoedas no portfólio, como o nome, o preço e o saldo.
- **Price price**: Dentro da ProductInfo, o campo Price contém o preço de mercado da cripto na data específica.
- **Balance balance**: Dentro da ProductInfo, o campo Balance armazena o saldo de cada cripto ativo no portfólio.

6. ProductInfo: Informações sobre as Criptomoedas

A classe ProductInfo contém detalhes sobre cada cripto ativo que faz parte do portfólio. Ela armazena as informações de preço e saldo de cada cripto, que são usadas para calcular o valor total de cada ativo dentro do portfólio.

Campos Importantes da Classe ProductInfo

- **Price price**: Armazena o preço de mercado da cripto. Esse preço pode ser atualizado periodicamente, refletindo as flutuações do mercado.
- **Balance balance**: Armazena o saldo de criptomoeda no portfólio. Esse valor representa a quantidade de unidades de cada cripto ativo no portfólio.

7. Price: O Preço de Mercado das Criptomoedas

A classe Price contém o preço de mercado de cada cripto ativo no portfólio. Esse preço é crucial para calcular o valor total de cada cripto dentro do portfólio, uma vez que o valor do ativo é o preço multiplicado pelo saldo.

Campos Importantes da Classe Price

- **Preço**: O valor do ativo em uma data específica, que pode ser acessado e atualizado conforme o mercado se altera. Esse preço é utilizado para calcular o valor total de cada cripto ativo.

8. Balance: O Saldo das Criptos no Portfólio

A classe Balance contém o saldo de unidades de cada cripto ativo dentro do portfólio. O saldo é usado junto com o preço de mercado para calcular o valor total de cada cripto ativo no portfólio.

Campos Importantes da Classe Balance

- **Quantidade de ativos**: A quantidade de unidades da cripto no portfólio. Esse valor é essencial para calcular o valor total de cada ativo no portfólio. Exemplo: Se o preço de uma cripto é 10.000 reais e o saldo é 2, o valor total da cripto será 20.000 reais.

Fluxo de Dados no Sistema para Criptos

1. **Solicitação do Relatório**: O usuário ou sistema solicita o valor total do portfólio de criptos para uma data específica.
2. **Obtenção dos Dados do Portfólio**: O método CalculatePortfolioByDate(date: datetime) do PortfolioController é chamado. Este, por sua vez, chama o PortfolioRepository.GetByDate(date) para obter os dados de PortfolioInfo para a data solicitada.
3. **Processamento do Portfólio**: A classe ProcessProduct é chamada para calcular o valor de cada cripto ativo no portfólio. Ela usa os preços atuais das criptos (armazenados na classe Price) e o saldo de cada cripto (armazenado na classe Balance) para calcular o valor total.
4. **Cálculo de Taxas**: A classe ProcessAdmFee calcula as taxas administrativas associadas ao portfólio de criptomoedas.
5. **Geração do Relatório**: Após calcular o valor total do portfólio e as taxas, a classe PortfolioReport é gerada, consolidando todas as informações para o relatório final.

Conclusão Final

O sistema foi projetado para calcular e gerar relatórios precisos de portfólios de criptomoedas, utilizando uma arquitetura orientada a objetos para organizar as informações sobre os ativos digitais. As classes envolvidas trabalham juntas para acessar, processar e gerar relatórios que refletem com precisão o valor de mercado dos ativos e as taxas administrativas associadas ao portfólio.

Ao garantir que cada classe tenha responsabilidades bem definidas, o sistema proporciona um fluxo de dados eficiente e escalável, permitindo que os cálculos e relatórios sejam realizados de forma precisa e rápida.

5) Projeto e Especificação do Sistema

O sistema desenvolvido é uma solução automatizada para o cálculo da cota diária de fundos de criptomoedas, que visa melhorar a eficiência e a precisão do processo de gestão de portfólios. A arquitetura do sistema é composta por diversos módulos interconectados, cada um com responsabilidades específicas, que se combinam para fornecer uma solução robusta e escalável.

5.1 Estrutura do Sistema

A arquitetura do sistema é baseada em uma estrutura de microserviços, utilizando a linguagem de programação C# e o framework .NET. O banco de dados MongoDB foi escolhido para armazenar as informações do portfólio devido à sua flexibilidade e capacidade de lidar com dados financeiros complexos. O sistema é dividido em várias camadas:

Camada de Infraestrutura: Responsável pela comunicação com o banco de dados MongoDB, implementada através da classe PortfolioRepository. Esta classe contém métodos para acessar e manipular os dados do portfólio, como GetByDate, que recupera informações de um portfólio com base em uma data específica.

Camada de Business: Inclui as lógicas de processamento de dados, como a classe ProcessAdmFee, que calcula a taxa de administração e gera relatórios a partir dos dados do portfólio. O método CalculateAndReturnAdmDocument realiza os cálculos necessários, como o valor total do portfólio, GAV (Valor de Ativos Geridos), NAV (Valor Líquido do Ativo), GAVPS (Valor de Ativos Geridos por Ação) e NAVPS (Valor Líquido do Ativo por Ação). Os cálculos são realizados da seguinte maneira:

1. O valor total do portfólio é calculado iterando sobre os saldos das criptomoedas, multiplicando a quantidade de cada ativo pelo seu preço

2. O valor inicial do portfólio é calculado de forma semelhante, permitindo a determinação da taxa de administração com base na variação do valor do portfólio ao longo do tempo.
3. O GAV e o NAV são então calculados:

O GAV é o total de ativos geridos, enquanto o NAV considera a taxa de administração, proporcionando uma visão precisa do valor líquido disponível para os cotistas.

Abaixo, vemos no algoritmo 1, a lógica de cálculo do NAV, GAV, NAVPS E GAVPS:

```
Função CalculateAndReturnAdmDocument(portfolioInfo, firstPortfolioInfo)

    // Inicializar variáveis para armazenar o valor total do portfólio atual e inicial
    valorTotalPortfolio <- 0.0

    // Calcular o valor total do portfólio atual
    Para cada item em portfolioInfo.ProdutoInfo[0].Saldos
        total <- item.Total
        preco <- Buscar preço correspondente de item.Ativo em portfolioInfo.ProdutoInfo[0].Precos
        valorTotalPortfolio <- valorTotalPortfolio + (total * preco)
    Fim Para

    valorTotalPortfolioInicial <- 0.0

    // Calcular o valor total do portfólio inicial
    Para cada item em firstPortfolioInfo.ProdutoInfo[0].Saldos
        total <- item.Total
        preco <- Buscar preço correspondente de item.Ativo em firstPortfolioInfo.ProdutoInfo[0].Precos
        valorTotalPortfolioInicial <- valorTotalPortfolioInicial + (total * preco)
    Fim Para

    // Calcular a taxa de administração (Admfee)
    taxaAdm <- portfolioInfo.ADM * (valorTotalPortfolio - valorTotalPortfolioInicial)

    // Criar o relatório do portfólio
    relatorio <- Novo RelatorioDePortfolio()

    // Preencher os campos do relatório
    relatorio.Tempo <- portfolioInfo.Tempo
    relatorio.GAV <- valorTotalPortfolio
    relatorio.NAV <- valorTotalPortfolio - taxaAdm
    relatorio.GAVPS <- relatorio.GAV / portfolioInfo.NumeroDeCotas
    relatorio.NAVPS <- relatorio.NAV / portfolioInfo.NumeroDeCotas

    // Retornar o relatório gerado
    Retornar relatorio
Fim Função
```

Algoritmo 1

Camada de Controllers: Esta camada expõe a funcionalidade do sistema através de uma API REST. Três endpoints principais foram definidos:

a) CalculatePortfolioByDate: calcula a cota do portfólio para uma data específica.
b) GetPortfolioByDate: recupera os dados do portfólio para uma data específica.
c) GetPortfolioAllDates: retorna os dados de todas as datas disponíveis.

5.2 Características Marcantes da Solução

Uma das principais características do sistema é sua capacidade de automatizar o cálculo da cota diária de fundos de criptomoedas, reduzindo significativamente a carga de trabalho manual dos analistas e minimizando o risco de erros humanos. Além disso, a solução é escalável e flexível, permitindo a integração com sistemas de gerenciamento de dados existentes e a adaptação a futuras necessidades do mercado.

As contribuições deste projeto incluem não apenas a implementação de um sistema eficiente e confiável, mas também a criação de uma base sólida para futuras expansões e melhorias. A documentação da API e do sistema está disponível em apêndices, permitindo uma fácil compreensão e utilização da solução desenvolvida.

5.3 Uso do Sistema

O sistema é utilizado através de uma interface de API REST, onde os analistas podem interagir com os módulos de forma eficiente. As operações são realizadas através de chamadas HTTP, permitindo que os analistas solicitem cálculos ou recuperem dados de portfólios de maneira rápida e intuitiva.

Em suma, o projeto representa um avanço significativo na gestão de fundos de criptomoedas, proporcionando uma solução robusta e automatizada que não apenas melhora a eficiência operacional, mas também aumenta a precisão e a confiabilidade dos dados fornecidos aos cotistas.

6) Implementação e Avaliação

6.1 Planejamento e Execução de Testes Funcionais

A implementação do sistema passou por rigorosos testes funcionais para garantir que todas as funcionalidades estivessem operando conforme o esperado. Para isso, foi elaborado um plano de testes abrangente, que incluiu as seguintes técnicas:

- **Testes Unitários**: Cada módulo do sistema, incluindo a lógica de negócios e os métodos de acesso ao banco de dados, foi testado de forma isolada para validar sua funcionalidade. Ferramentas como o xUnit foram utilizadas para escrever e executar os testes.

- **Testes de Integração**: Após a validação das funcionalidades individuais, os testes de integração foram realizados para garantir que os módulos interagissem corretamente entre si. Os endpoints da API foram testados para verificar se as chamadas HTTP retornavam os resultados esperados.
- **Cenários de Teste**: Os principais cenários de teste incluíram:

    - Cálculo da cota do portfólio para uma data específica.
    - Recuperação de dados do portfólio para datas específicas e todas as datas disponíveis.
    - Verificação da precisão dos cálculos de GAV e NAV em diferentes situações de mercado.
    -

Os resultados dos testes foram positivos, com todas as funcionalidades testadas apresentando resultados corretos e sem erros. A cobertura dos testes foi monitorada, garantindo que a maioria do código estivesse coberta por testes automatizados.

6.2 Planejamento e Execução de Outros Testes

Além dos testes funcionais, foram realizados testes de desempenho para avaliar a escalabilidade e a eficiência do sistema sob carga. Esses testes incluíram:

- **Testes de Carga**: Simulando um grande número de chamadas simultâneas aos endpoints da API, medindo o tempo de resposta e a capacidade do sistema de lidar com a carga.
-
- **Testes de Estresse**: Excedendo os limites esperados de uso para identificar pontos de falha e avaliar como o sistema se comporta em situações extremas.

Os resultados dos testes de desempenho mostraram que o sistema se comportou de maneira robusta, suportando um alto volume de requisições sem degradação significativa no tempo de resposta.

6.3 Testes realizados

Foram realizados testes nos 3 endpoints para datas de 2024 no mês de outubro.

Teste 1:	Capturar as informações do Portfolio via o endpoint “GetPortfolioByDate” e retornar as informações do portfolio cada uma das 3 datas testadas de forma individual em um objeto (PorfolioInfo) -> retornou com sucesso os documentos para 12, 13 e 14 de outubro de 2024.
Teste 2:	Capturar as informações do Portfolio via o endpoint “GetPortfolioAllDates” e retornar as informações do portfolio para todas as datas que tiverem documentos na base via uma lista do objeto PorfolioInfo -> retornou com sucesso os documentos para 12, 13 e 14 de outubro de 2024 juntos em uma lista, conforme esperado.
Teste 3:	Calcular o NAV, GAV, NAVPS e GAVPS do Portfolio para uma data específica via o endpoint “CalculatePortfolioByDate” e retornar as informações em um objeto (PortfolioReport) -> retornou com sucesso os documentos para 12, 13 e 14 de outubro de 2024.

6.4 Comentários sobre a Implementação

Durante a implementação do sistema, alguns desafios foram encontrados:

- **Integração com o MongoDB**: No início do desenvolvimento, houve dificuldades na configuração correta da conexão com o banco de dados MongoDB, que atrasou o progresso. Essa questão foi resolvida após

revisar a documentação do MongoDB e ajustar as configurações de conexão.

- **Cálculos de GAV e NAV**: A lógica inicial para o cálculo da taxa de administração apresentou erros, especialmente em cenários onde os preços dos ativos flutuavam significativamente. Para contornar isso, foi necessário refinar a lógica de cálculo e adicionar validações para garantir que os dados dos ativos fossem corretamente recuperados antes de realizar os cálculos.
- **Manutenção da Documentação**: A documentação inicial do sistema foi desatualizada durante o processo de desenvolvimento. Para resolver esse problema, uma rotina de revisão da documentação foi estabelecida, garantindo que todas as alterações no código fossem refletidas nos documentos.

A implementação geral foi bem-sucedida, resultando em um sistema robusto, capaz de atender às necessidades de cálculo e gerenciamento de cotas de fundos de criptomoedas.

7) Considerações finais

7.1 Contribuições do Trabalho

Este trabalho trouxe várias contribuições significativas tanto para a comunidade técnica quanto para os usuários finais:

- **Automação de Processos**: A implementação do sistema de cálculo automático das cotas de fundos de criptomoedas representa um avanço significativo em relação aos métodos manuais anteriores, oferecendo maior precisão e eficiência. Isso é especialmente relevante em um campo onde a agilidade e a exatidão são cruciais.

- **Documentação e Boas Práticas**: A documentação detalhada do sistema, juntamente com a adoção de boas práticas de desenvolvimento, serve como um guia para futuros desenvolvedores que desejem aprimorar ou expandir a solução. Essa abordagem facilita a manutenção e a escalabilidade do sistema.

- **Integração de Tecnologias**: O uso do MongoDB e do C# em um ambiente de nuvem, como o Azure, demonstra a viabilidade de integrar tecnologias modernas para resolver problemas financeiros complexos, contribuindo para o desenvolvimento de soluções mais robustas e confiáveis.

7.2 Aprendizados

Este projeto proporcionou uma valiosa experiência de aprendizado em diversas áreas:

- **Desenvolvimento Ágil**: A importância de um ciclo de desenvolvimento ágil e iterativo foi claramente evidenciada, permitindo ajustes rápidos e correções de erros à medida que o sistema evoluía.

- **Trabalho em Equipe**: A colaboração eficaz com colegas e a comunicação constante foram cruciais para superar desafios e garantir a entrega de um produto de qualidade.

- **Gestão de Dados**: A manipulação e análise de dados financeiros complexos, incluindo a importância de garantir a precisão e a integridade dos dados, foram aspectos críticos que enriqueceram minha compreensão da área.

7.3 Reflexões sobre o Processo

Se eu pudesse recomeçar este projeto, algumas abordagens diferentes poderiam ser consideradas:

- **Melhor Planejamento de Requisitos**: Um mapeamento mais detalhado dos requisitos iniciais teria ajudado a evitar alterações significativas durante o desenvolvimento. Isso teria reduzido o tempo de retrabalho e melhorado a eficiência.

- **Escolha de Tecnologias**: Embora a escolha de C# e MongoDB tenha se mostrado eficaz, explorar alternativas, como bancos de dados relacionais ou outras linguagens, poderia ter gerado insights diferentes e potencialmente melhorias na performance.

- **Testes de Usabilidade**: A inclusão de testes de usabilidade desde o início poderia ter fornecido uma perspectiva valiosa sobre a interação do usuário com o sistema, permitindo melhorias mais alinhadas com as necessidades dos usuários finais.

7.4 Oportunidades para Trabalhos Futuros

Identifiquei várias oportunidades para trabalhos futuros que podem ser recomendadas para próximos alunos:

- **Expansão da Funcionalidade**: A adição de recursos, como uma interface de usuário gráfica e relatórios personalizados, poderia melhorar a acessibilidade do sistema e atender a um público mais amplo.
- **Análise Preditiva**: Integrar algoritmos de machine learning para prever o desempenho de ativos e otimizar estratégias de investimento poderia ser um tema interessante para futuras pesquisas.

- **Integração com APIs Externas**: A conexão do sistema com APIs de mercado financeiro poderia fornecer dados em tempo real, aumentando a relevância e a utilidade do sistema.

- **Estudos de Caso e Comparações**: A realização de estudos de caso para comparar a performance do sistema desenvolvido com soluções existentes poderia oferecer uma visão crítica e validação da eficácia da abordagem adotada.

Essas recomendações podem ajudar a ampliar o escopo do projeto e fomentar a inovação na área de gestão de portfólios de ativos financeiros.

8) Referências Bibliográficas

[1] C. Martin., Robert; **Clean Code: A Handbook of Agile Software Craftsmanship**, Pearson, Estados Unidos, 1 de agosto de 2008

[2] Garcia, Henrique; **Como funcionam as cotas dos fundos de investimentos?**, Smart Brain, Brasil, 11 de março de 2021

[3] Renan Salinas; **O crescimento exponencial da automação no setor financeiro**, Brasil, 15 de outubro de 2024

[4] Redação da XP Educação; **Bancos de dados NoSQL: entenda o conceito e confira as categorias e exemplos**, Brasil, 1 de julho de 2022

[5] Stefani Group; **Automação no mercado financeiro impulsiona eficiência e promove inovação**, Brasil, 10 de dezembro de 2024

[6] Hugo Oliveira; **Segurança da Informação: o que é, os pilares e importância!**, Brasil, 20 de setembro de 2024

[7] Douglas da Silva; **Qual a importância da segurança da informação e como aplicar?**, Brasil, 22 de julho de 2022

[8] Pierre Veyrat; **Por que apostar em automatização de processos administrativos?**, Brasil, 22 de junho de 2020

[9] Bruna Gomes; **Segurança de API: qual a sua importância?**, Brasil, 20 de junho de 2023

[10] Pedro freitas; **O Futuro dos Criptoativos no Brasil: Regulamentação e Segurança nas Operações**, Brasil, 23 de outubro de 2024

[11] Ian Sommervile; **Engenharia de Software,** Pearson, Estados Unidos, 1 de dezembro de 2011