# Universidade Federal do Rio Grande do Sul - Instituto de Informática

## Departamento de Informática Aplicada - INF01120 - Técnicas de Construção de Programas

## Trabalho Prático - Etapa 1: Requisitos, projeto e interface

## 1. Especificação

### 1.2 Mudanças em relação à Etapa anterior

A única mudança em relação ao que foi especificado no documento da Etapa 0 é a substituição da linguagem de programação: utilizaremos Python ao invés de Java. Fora isso, o objetivo do sistema, suas funcionalidades principais e o escopo permanecem os mesmos, conforme planejado inicialmente.

### 1.3 Requisitos

#### Requisitos Funcionais

| Código | Descrição |
| --- | --- |
| RF-1 | O sistema deve permitir ao usuário cadastrar novas carteiras de ações, especificando nome e ativos integrantes. |
| RF-2 | O sistema deve exibir o desempenho atual das carteiras, incluindo valor total e rendimento percentual. |
| RF-3 | O sistema deve atualizar cotações das ações automaticamente a partir de uma API externa (por exemplo, Yahoo Finance). |
| RF-4 | O sistema deve permitir visualizar o histórico de desempenho das carteiras em gráficos. |
| RF-5 | O sistema deve possibilitar ao usuário exportar relatórios das carteiras em CSV. |

#### Requisitos Não Funcionais

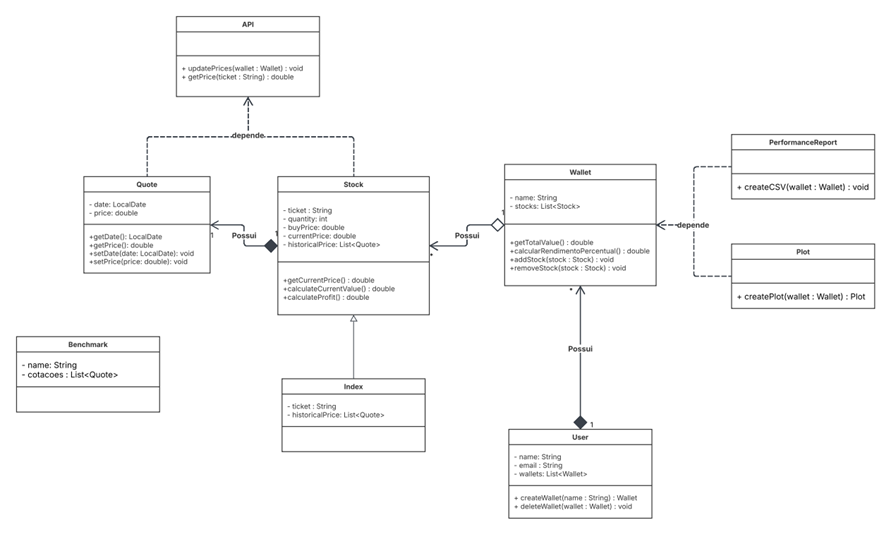
| Código | Descrição |
| --- | --- |
| RNF-1 | O sistema deve apresentar uma interface gráfica intuitiva e responsiva. |
| RNF-2 | As atualizações de cotações devem ocorrer em tempo inferior a 5 segundos por requisição de carteira. |
| RNF-3 | O sistema deve avisar o usuário se não conseguir atualizar as cotações. |
| RNF-4 | O sistema deve salvar automaticamente os dados das carteiras do usuário em arquivos CSV ao sair, e carregá-los ao iniciar, garantindo a persistência dos dados. |
| RNF-5 | O sistema deverá carregar os dados (até 10k ações) em menos de 5 segundos. |

Nos requisitos funcionais, priorizamos as funcionalidades essenciais para a operação mínima do sistema, como cadastro e consulta de desempenho (RF-1 e RF-2), seguidas por atualizações automáticas e visualizações (RF-3 e RF-4), e por último funcionalidades complementares como exportação de relatórios (RF-5).

Nos requisitos não funcionais, priorizamos a usabilidade e performance percebida pelo usuário (RNF-1, RNF-2), seguida pelo tratamento de erros que a API pode apresentar (RNF-3) e por fim persistência de dados e sua performance (RNF-4 e RNF-5).

### 1.4 Projeto

#### Diagrama de Classes



Classes:  
  
- Quote

- Atributos:

- date: LocalDate

- price: double

- Métodos:

+ getDate(): LocalDate

+ getPrice(): double

+ setDate(date: LocalDate): void

+ setPrice(price: double): void

- Wallet

- Atributos:  
 - name: String  
 - stocks: List<Stock>  
 - Métodos:  
 + getTotalValue() : double  
 + calcularRendimentoPercentual() : double  
 + addStock(stock : Stock) : void  
 + removeStock(stock : Stock) : void  
  
- Stock

- Atributos:

- ticket : String

- quantity: int

- buyPrice: double

- currentPrice: double

- historicalPrice: List<Quote>

- Métodos:

+ getCurrentPrice() : double

+ calculateCurrentValue() : double

+ calculateProfit() : double  
  
- User  
 - Atributos:  
 - name: String  
 - email : String  
 - wallets: List<Wallet>  
 - Métodos:  
 + createWallet(name : String) : Wallet  
 + deleteWallet(wallet : Wallet) : void

- API  
 - Métodos:  
 + updatePrices(wallet : Wallet) : void  
 + getPrice(ticket : String) : double  
  
- PerformanceReport   
 - Métodos:  
 + createCSV(wallet : Wallet) : void  
  
- Plot  
 - Métodos:  
 + createPlot(wallet : Wallet) : Plot  
- Index:

- Atributos:

- ticket : String

- historicalPrice: List<Quote>

- Benchmark  
 - Atributos:  
 - name: String  
 - cotacoes : List<Quote>  
 - Métodos:  
 + …

**Relacionamentos:**

Associação/Agregação entre Wallet e Stock (1 para muitos).

Associação/Composição entre Quote e Stock (1 para 1).

Associação/Composição entre User e Wallet.

Dependência entre PerformanceReport e Wallet.

Dependência entre Plot e Wallet.

Dependência entre API e Quote/Stock.

Especialização de Stock em relação a Index

Associação:

User e Wallet: Um usuário pode ter várias carteiras (Wallet), e cada carteira pertence a um único usuário.

Relação: User - Wallet (Um usuário tem várias carteiras).

Agregação:

Wallet e Stock: Uma carteira pode ter várias ações (Stock), mas as ações podem existir independentemente da carteira (elas podem ser parte de outras carteiras).

Relação: Wallet - Stock(Uma carteira contém várias ações).

Composição:

Quotation e Stock: A cotação (Quote) é uma entidade dependente de uma ação específica (Stock). Se a cotação for excluída, a relação com a ação também é perdida.

Relação: Quote- Stock (Uma cotação pertence a uma única ação).

Generalização/Especialização:

Index e Stock: A classe Stock pode ser especializada a partir da classe Index, onde Index contém apenas os preços históricos.

Relação: Stock - Holding (Stock é uma especialização de Index, Stock é um Index com mais informações).

Dependência:

Plot e Wallet: A classe Plot depende da classe Wallet para criar o gráfico com base no desempenho das ações da carteira

Relação: Plot - Wallet ( Plot depende das ações contidas em Wallet).

**Discussão:**

O projeto foi criado com o objetivo de acompanhar o desempenho de uma carteira de ações, utilizando dados atualizados obtidos por meio de uma API. A estrutura proposta faz uso dos principais conceitos da Programação Orientada a Objetos (POO), permitindo uma organização modular, coesa e de fácil manutenção, além de facilitar a distribuição das tarefas entre os membros do grupo.

Adotando o paradigma do “dividir para conquistar”, fizemos a decomposição do programa em diferentes categorias de classes, conforme suas responsabilidades no sistema:

* Funcionais/operacionais: User, Wallet, Stock, Benchmark... são as classes principais, que atendem diretamente os RF’s.
* Auxiliares: API, Quote, Plot.... São responsáveis por operações de suporte à lógica principal, como a classe API, que realiza update e get dos valores das cotações.

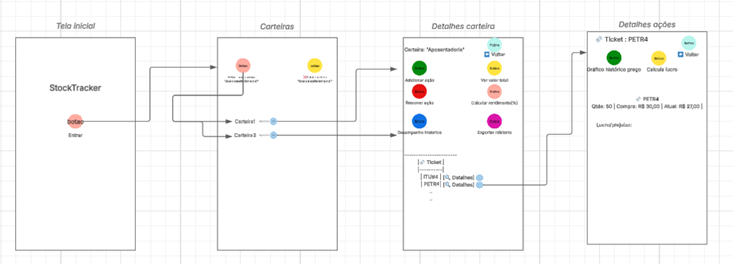
Essa divisão clara de responsabilidades facilita o trabalho colaborativo, pois permite que diferentes integrantes do grupo atuem em partes distintas do sistema de forma independente. Por exemplo, um membro que precisa utilizar o método updatePrices da classe API não precisa compreender sua lógica interna ou como os dados são recuperados da fonte, basta conhecer sua interface pública e o que ela oferece.

**Explicação do diagrama:**

O diagrama de classes do sistema foi estruturado para representar de forma clara e coesa os principais elementos envolvidos no acompanhamento de carteiras de ações. Ele inclui classes fundamentais que modelam conceitos centrais como User, Wallet e Stock, refletindo diretamente a lógica do domínio financeiro.

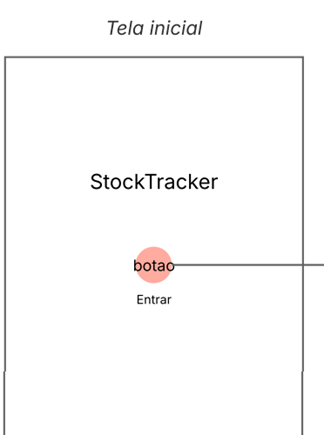
Além disso, há classes auxiliares importantes como API, responsável por fornecer dados de mercado simulados, e PerformanceReport, que permite gerar análises e comparações de desempenho. Essa organização modular favorece a separação de responsabilidades, facilita o trabalho em equipe, seguindo os princípios da POO.

### 1.5 Interface com o Usuário

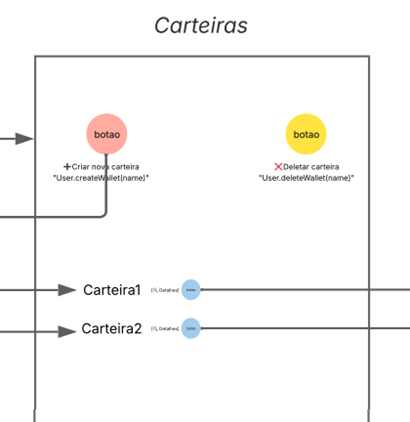


O layout adotado na interface segue uma abordagem hierárquica [Tela Inicial→Carteiras→DetalhesCarteira→Detalhes Ações], facilitando a utilização do programa para os usuários.

Começando pela tela inicial: ela contém o nome do aplicativo seguido de um botão para iniciar.



A seguir, temos a interface para o usuário acompanhar suas carteiras, permitindo-o de forma clara e intuitiva criar ou deletar carteiras, assim como consultar os detalhes de cada uma.



Em seguida, a seção de detalhes da carteira apresenta ao usuário as ações que a compõem, oferecendo funcionalidades como adicionar ou remover ativos, visualizar o valor total da carteira, seu desempenho e rendimento.



Por fim, a seção de detalhes de cada ação permite ao usuário gerar um gráfico com o histórico de preços e calcular o lucro ou prejuízo associado ao papel.



Vale ressaltar que a interface apresentada é uma versão simplificada, com o objetivo principal de demonstrar as funcionalidades do projeto. A seguir, será exibida uma interface mais próxima da versão final do StockTracker.

