Documento de arquitetura sistema Gestão de Investimentos

Introdução

Este documento descreve a arquitetura de um sistema de gestão de investimentos desenvolvido em .NET com C# e utilizando MongoDB como banco de dados. A aplicação tem como objetivo fornecer uma plataforma eficiente e segura para que usuários gerenciem seus portfólios de investimentos, incluindo ações, títulos e criptomoedas, com um backend robusto e escalável.

O sistema foi projetado seguindo os princípios da **Clean Architecture**, garantindo uma separação clara de responsabilidades e promovendo a manutenção, escalabilidade e testabilidade do código. A arquitetura é dividida em várias camadas, incluindo **API**, **APPLICATION**, **DOMAIN**, **INFRA.DATA**, **INFRA.IOC**, e **TESTS**, cada uma responsável
por um aspecto específico da funcionalidade e fluxo de dados da aplicação.

Objetivos do Sistema

O sistema de gestão de investimentos foi concebido para atender às seguintes necessidades:

- **Gestão de Portfólios:** Permitir que usuários criem, editem e visualizem seus portfólios de investimento, organizando e monitorando diferentes tipos de ativos, como ações, títulos e criptomoedas.
- **Registro de Transações:** Facilitar o registro de transações de compra e venda de ativos, mantendo um histórico detalhado de todas as operações financeiras.
- **Autenticação Segura:** Garantir a segurança dos dados dos usuários através de um mecanismo robusto de autenticação baseado em tokens JWT (JSON Web Tokens).
- **Escalabilidade e Performance:** Suportar um grande volume de dados e requisições simultâneas, utilizando MongoDB como banco de dados para garantir alta performance e escalabilidade.
- **Facilidade de Manutenção e Evolução:** Utilizar uma arquitetura modular e desacoplada para permitir a fácil adição de novas funcionalidades e manutenção do sistema.

Justificativa para a Arquitetura e Tecnologias Escolhidas

A escolha do **MongoDB** como banco de dados foi motivada por sua flexibilidade no armazenamento de documentos JSON, alta performance em operações de escrita e capacidade de escalabilidade horizontal. Essas características o tornam ideal para

uma aplicação que lida com dados financeiros complexos e variáveis, como o gerenciamento de ativos e transações.

O uso do **C#** e do **.NET** no backend fornece uma base sólida para o desenvolvimento, graças à robustez da linguagem, suporte para práticas de desenvolvimento orientadas a objetos, alta performance, e uma ampla gama de bibliotecas e ferramentas para segurança e comunicação com bancos de dados. Além disso, o framework .NET Core é multiplataforma, permitindo fácil implantação em diferentes ambientes, incluindo containers Docker.

Estrutura do Documento

Este documento de arquitetura está organizado nas seguintes seções:

1. **Descrição Geral da Arquitetura:** Uma visão geral da arquitetura do sistema, detalhando cada camada e suas responsabilidades.
2. **Modelo de Dados:** Definição das entidades principais, incluindo suas propriedades e relacionamentos.
3. **Fluxo de Dados e Casos de Uso:** Descrição dos principais fluxos de dados e casos de uso da aplicação, incluindo autenticação, criação de portfólios, e registro de transações.
4. **Definição dos Endpoints da API:** Detalhamento dos endpoints disponíveis na API RESTful, incluindo métodos, parâmetros e exemplos de requisições e respostas.
5. **Camadas da Aplicação:** Explicação detalhada das responsabilidades de cada camada da arquitetura (API, Application, Domain, Infra.Data, Infra.IOC, Tests).
6. **Casos de Teste:** Definição dos casos de teste para garantir a qualidade e a robustez da aplicação.
7. **Justificativas Técnicas:** Argumentos para a escolha de tecnologias e práticas adotadas no desenvolvimento.

Este documento é destinado a desenvolvedores, arquitetos de software, engenheiros de DevOps e outros profissionais interessados em entender a estrutura e os fundamentos técnicos do sistema de gestão de investimentos. Ele serve como guia para o desenvolvimento, manutenção e evolução contínua da aplicação.

Estrutura da Aplicação

O projeto será dividido nas seguintes camadas:

API: Responsável por expor os endpoints da aplicação via HTTP usando controllers.

APPLICATION: Contém os casos de uso, serviços e DTOs (Data Transfer Objects). Esta camada orquestra o fluxo de dados entre a API e a camada DOMAIN.

DOMAIN: Contém as entidades, interfaces de repositório e regras de negócio.

INFRA.DATA: Implementação dos repositórios e contextos de banco de dados, bem como qualquer lógica de persistência específica.

INFRA.IOC: Configuração da Injeção de Dependência (Dependency Injection).
TESTS: Contém os testes unitários e de integração.

Entidades e Regras de Negócio

1. Entidades

Usuário
- `Id`: Identificador único do usuário.
- `Nome`: Nome completo do usuário.
- `Email`: Email do usuário.
- `Senha`: Hash da senha para autenticação.

Portfólio
- `Id`: Identificador único do portfólio.
- `UsuarioId`: Referência ao usuário proprietário do portfólio.
- `Nome`: Nome descritivo do portfólio.
- `Descricao`: Descrição detalhada do portfólio.

Ativo
- `Id`: Identificador único do ativo.
- `TipoAtivo`: Tipo do ativo (e.g, Ações, Títulos, Criptomoedas).
- `Nome`: Nome do ativo.
- `Codigo`: Código de negociação do ativo (e.g., AAPL, BTC).

Transação
- `Id`: Identificador único da transação.
- `PortfolioId`: Referência ao portfólio em que a transação foi realizada.
- `AtivoId`: Referência ao ativo negociado.
- `TipoTransacao`: Tipo da transação (compra ou venda).
- `Quantidade`: Quantidade de ativos negociados.
- `Preco`: Preço por unidade do ativo no momento da transação.
- `DataTransacao`: Data e hora em que a transação foi efetuada.

Endpoints

Autenticação

- **POST /Usuario/autenticar**: Endpoint para autenticação do usuário.
  - Requisição:

```
{
      "email": "usuario@exemplo.com",
      "senha": "senha123"
}
```

  - Resposta: `"JWT_TOKEN"`

Usuários

- **POST / Usuario/cadastrar:** Criar um novo usuário.
  - Requisição:

```
{
      "nome": "João da Silva",
      "email": "joao@exemplo.com",
      "senha": "senha123"
}
```

  - Resposta: `"ID_USER"`

- **GET/ Usuario/listar:** Lista dos usuários do sistema.
  - Requisição: Requisição: Header com JWT Token.
  - Resposta:

```
[{
      "id": "1",
      "nome": "João da Silva",
      "email":"joao@exemplo.com",
      "role": 0
}]
```

- **DELETE /Usuario/{Id} :** Lista os usuários do sistema.
  - Requisição: Requisição: Header com JWT Token e Id do usuário
  - Resposta: `Status code 200 ok`

Portfólios

- **GET /Portfolio/listar**: Listar todos os portfólios do usuário autenticado.
  - Requisição: Header com JWT Token.
  - Resposta:

```
[{
      "usuarioId": "1",
      "id": "1",
      "nome": "Portfólio A",
      "descricao": "Meu primeiro portfólio"
}]
```

- **POST /Portfolio/cadastrar**: Criar um novo portfólio.
  - Requisição:

```
{
     "usuarioId": "1",
      "nome": "Portfólio B",
      "descricao": "Portfólio de
      teste"
}
```

  - Resposta: `Status code 200 ok`

- **GET /Portfolio/{id}**: Obter detalhes de um portfólio específico.
  - Requisição: Header com JWT Token.
  - Resposta:

```
{
     "usuarioId": "1",
     "id": "1",
     "nome": "Portfólio A",
     "descricao": "Meu primeiro portfólio"
}
```

- **DELETE /Portfolio/{id}**: Remover um portfólio.
  - Requisição: Header com JWT Token.
  - Resposta: `Status code 200 ok`

Ativos

- **GET /Ativo/listar** Listar todos os ativos disponíveis.
  - Requisição: Header com JWT Token.
  - Resposta:

```
[{
     "id": "1",
     "tipoAtivo": 1,
     "nome": "Apple",
     "codigo": "AAPL"
}]
```

- **POST /Ativo/cadastrar**: Criar um novo ativo.
  - Requisição:

```
{
     "tipoAtivo": "Ações",
     "nome": "Apple",
     "codigo": "AAPL"
 }
```

  - Resposta: `Status code 200 ok`

- **GET /Ativo/{id}**: Obter detalhes de um ativo específico.
  - Requisição: Header com JWT Token.
  - Resposta:

```
{
     "id": "1",
     "tipoAtivo": 1,
     "nome": "Apple",
     "codigo": "AAPL"
}
```

- **DELETE /Ativo/{id}**: Remover um ativo.
  - Requisição: Header com JWT Token.
  - Resposta: `Status code 200 ok`

Transações

- **GET /Transacao/listar**: Listar todas as transações de um portfólio.
  - Requisição: Header com JWT Token e `PortfolioId`.
  - Resposta:

```
[{
     "portfolioId": "1",
     "ativoId": "1",
     "id": "1",
     "tipoTransacao": 0,
     "quantidade": 10,
     "preco": 150,
     "dataTransacao": "2024-09-03T12:00:00Z"
}]
```

- **POST /Transacao/cadastrar**: Registrar uma nova transação.
  - Requisição:

```
{
     "portfolioId": "1",
     "ativoId": "1",
     "tipoTransacao": 0,
     "quantidade": 10,
     "preco": 150
}
```

  - Resposta: `Status code 200 ok`

- **GET /Transacao/{Id}**: Obter detalhes de uma transação específica.
  - Requisição: Header com JWT Token e Id
  - Resposta:

```
{
     "portfolioId": "1",
     "ativoId": "1",
     "id": "1",
     "tipoTransacao": 0,
     "quantidade": 10,
     "preco": 150,
     "dataTransacao": "2024-09-03T12:00:00Z"
}
```

- **DELETE / Transacao /{id}**: Remover uma transação.
  - Requisição: Header com JWT Token.
  - Resposta: `Status code 200 ok`

Casos de Teste

Autenticação

- Verificar se o login com credenciais corretas retorna um token JWT válido.
- Verificar se o login com credenciais incorretas retorna um erro de autenticação.

Usuário

- Criar um novo usuário e verificar se é salvo corretamente no banco de dados.
- Tentar criar um usuário com um e-mail já existente e verificar se retorna erro.

Portfólio

- Criar um novo portfólio e verificar se é salvo corretamente no banco de dados.
- Listar portfólios e verificar se todos os portifólios do usuário autenticado são retornados.
- Obter detalhes de um portfólio existente e verificar os dados retornados.
- Remover um portfólio e verificar se foi removido corretamente.

Ativo

- Listar ativos e verificar se todos os ativos cadastrados estão sendo retornados.

Transação

- Registrar uma nova transação e verificar se é salva corretamente.
- Listar todas as transações de um portfólio e verificar se os dados retornados estão corretos.

Implementação das Camadas

1. API

- Controladores responsáveis pelos endpoints HTTP.
- Uso de Swagger para documentar e testar a API.

2. APPLICATION

- Casos de uso (ex.: `CreateUser`, `LoginUser`, `CreatePortfolio`).
- DTOs para transporte de dados entre camadas.

3. DOMAIN

- Entidades (`User`, `Portfolio`, `Asset`, `Transaction`).
- Interfaces de repositórios (`IUserRepository`, `IPortfolioRepository`, etc.).

4. INFRA.DATA

- Implementação das interfaces de repositório para MongoDB.
- Configuração de contexto de banco de dados.

5. INFRA.IOC

- Configuração de injeção de dependência usando `Microsoft.Extensions.DependencyInjection`.

6. TESTS

- Testes unitários para validar lógica de negócio.
- Testes de integração para validar a interação entre camadas e com o banco de dados.