DOCUMENTAÇÃO TÉCNICA DO SISTEMA DE CONTROLE DE AÇÕES FINANCEIRAS

Autor: Matheus Maia
Local: Online
Data: 24 de junho de 2025

SUMÁRIO

---

1 INTRODUÇÃO

Este documento apresenta a documentação técnica detalhada do sistema de controle de ações financeiras desenvolvido na plataforma Django. O sistema oferece aos usuários funcionalidades essenciais para a gestão de seus investimentos, incluindo a capacidade de realizar transferências bancárias, comprar e vender ações, marcar ações como favoritas para acompanhamento e uma interface administrativa para gerenciamento do sistema.

O objetivo desta documentação é fornecer uma visão clara e abrangente da arquitetura do software, detalhando as classes, seus relacionamentos e os padrões

de projeto empregados. A análise aqui apresentada serve como um guia para desenvolvedores, arquitetos de software e outras partes interessadas no entendimento, manutenção e evolução do sistema.

2 METODOLOGIA

Para a elaboração desta documentação, foi utilizada a abordagem de modelagem visual através da Linguagem de Modelagem Unificada (UML - Unified Modeling Language). O diagrama de classes UML, fornecido como base (diagrama_classes_django_abnt.png), foi o ponto de partida para a análise da estrutura estática do sistema.

A metodologia consistiu nos seguintes passos:
1. **Análise de Código Fonte:** Exame dos arquivos de código do projeto Django para extrair as definições de classes, atributos, métodos e relacionamentos.
2. **Mapeamento UML:** Correlação da análise do código com o diagrama de classes UML para validar e complementar a representação visual da arquitetura.
3. **Estruturação ABNT:** Organização das informações coletadas em um formato de documentação técnica, seguindo a norma NBR 14724 da ABNT para trabalhos acadêmicos e técnicos, garantindo clareza, padronização e rastreabilidade.

A documentação foca em uma descrição hierárquica das camadas do padrão Model-View-Controller (MVC), inerente ao framework Django, para proporcionar uma compreensão lógica e organizada da arquitetura.

3 DESCRIÇÃO DAS CLASSES

O sistema é estruturado em três camadas principais, conforme o padrão MVC do Django: Models, Forms e Views.

3.1 Camada de Modelos (Models)

A camada de Modelos é responsável pela representação dos dados e pela lógica de negócio fundamental.

3.1.1 Account

Representa a conta financeira de um usuário no sistema.

- **Atributos:**
  - `user`: Relacionamento um-para-um (OneToOneField) com o modelo `User` do Django.
  - `account_number`: Número da conta (CharField).
  - `balance`: Saldo disponível na conta (DecimalField).
  - `theme`: Preferência de tema do usuário (CharField).
- **Métodos:**
  - `__str__()`: Retorna a representação textual da conta.

3.1.2 Transfer

Modela uma transferência de fundos entre dois usuários.

- **Atributos:**
  - `sender`: Chave estrangeira (ForeignKey) para o usuário remetente.
  - `recipient`: Chave estrangeira (ForeignKey) para o usuário destinatário.
  - `amount`: Valor da transferência (DecimalField).
  - `created_at`: Data e hora da criação da transferência (DateTimeField).
- **Métodos:**
  - `execute()`: Lógica de negócio para efetivar a transferência, debitando do remetente e creditando no destinatário.
  - `__str__()`: Representação textual da transferência.

3.1.3 Stock

Representa uma ação disponível para negociação no mercado.

- **Atributos:**
  - `name`: Nome da empresa/ação (CharField).
  - `code`: Código (ticker) da ação (CharField).
  - `current_price`: Preço atual da ação (DecimalField).
- **Métodos:**
  - `__str__()`: Retorna o código da ação.

3.1.4 Trade

Registra uma operação de compra ou venda de ações por um usuário.

- **Atributos:**
  - `BUY`, `SELL`, `TYPE_CHOICES`: Constantes para definir o tipo de operação.
  - `user`: Chave estrangeira (ForeignKey) para o usuário que realiza a operação.
  - `stock`: Chave estrangeira (ForeignKey) para a ação negociada.
  - `quantity`: Quantidade de ações negociadas (PositiveIntegerField).
  - `trade_type`: Tipo de operação (CharField, 'buy' ou 'sell').
  - `price`: Preço da ação no momento da operação (DecimalField).
  - `created_at`: Data e hora da operação (DateTimeField).
- **Métodos:**
  - `execute()`: Processa a operação de compra ou venda, ajustando o saldo do usuário e, potencialmente, seu portfólio de ações.
  - `__str__()`: Representação textual da operação.

3.1.5 FavoriteStock

Permite que um usuário marque uma ação como favorita.

- **Atributos:**
  - `user`: Chave estrangeira (ForeignKey) para o usuário.
  - `stock`: Chave estrangeira (ForeignKey) para a ação.
- **Métodos:**
  - `__str__()`: Representação textual do favorito.
- **Constraints:**
  - `unique_together`: Garante que um usuário só pode favoritar a mesma ação uma única vez.

3.2 Camada de Formulários (Forms)

Esta camada é responsável pela validação e manipulação de dados de entrada do usuário.

3.2.1 TransferForm

Formulário para a realização de transferências. Herda de `forms.ModelForm`.

- **Métodos de Validação:**
  - `clean_account()`: Valida se a conta do destinatário existe.
  - `clean_amount()`: Valida se o saldo do remetente é suficiente para a transferência.
- **Métodos de Negócio:**
  - `save()`: Cria a instância do modelo `Transfer` após a validação.

3.2.2 TradeForm

Formulário para a execução de operações de compra e venda de ações. Herda de `forms.Form`.

- **Métodos de Validação:**
  - `clean()`: Validação complexa que verifica o saldo do usuário para compras e a quantidade de ações em carteira para vendas.

3.2.3 BootstrapUserCreationForm

Formulário customizado para o registro de novos usuários. Herda de `UserCreationForm` do Django.

- **Funcionalidade:** Adiciona classes CSS do Bootstrap aos campos do formulário para estilização e implementa a validação de confirmação de senha.

3.2.4 AdminSetUserPasswordForm

Formulário utilizado na interface administrativa para que um administrador possa redefinir a senha de um usuário. Herda de `forms.Form`.

- **Métodos:**
  - `clean_username()`: Valida se o usuário informado existe.
  - `save()`: Salva a nova senha para o usuário.

3.3 Camada de Visões (Views)

A camada de Visões processa as requisições HTTP, interage com os Modelos e Formulários, e renderiza as respostas.

3.3.1 AdminPasswordResetView

Uma Class-Based View para a página de redefinição de senha pelo administrador. Herda de `LoginRequiredMixin`, `UserPassesTestMixin`, `FormView`.

- **Funcionalidade:** Controla o acesso à funcionalidade (apenas para superusuários) e processa o `AdminSetUserPasswordForm`.

3.3.2 Visões Baseadas em Funções (Function-Based Views)

O sistema utiliza majoritariamente visões baseadas em funções para implementar as funcionalidades principais.

- **Principais Funções:**
  - `register`: Controla o registro de novos usuários.
  - `dashboard`: Exibe o painel principal do usuário.
  - `make_transfer`: Gerencia o processo de transferência de fundos.
  - `stock_list`: Lista as ações disponíveis.
  - `operate_stock`: Processa a compra ou venda de uma ação.
  - `toggle_favorite_stock`: Adiciona ou remove uma ação da lista de favoritos do usuário.
  - `trade_history`: Exibe o histórico de transações do usuário.
  - `delete_user`: Permite que um administrador exclua um usuário.

4 RELACIONAMENTOS E CARDINALIDADE

A seguir, são detalhados os relacionamentos entre as principais entidades do sistema, conforme modelado no banco de dados.

- **User ↔ Account (1:1):** Cada `User` possui exatamente uma `Account`. Este relacionamento é implementado com um `OneToOneField` no modelo `Account`, estabelecendo um vínculo direto e único.
- **User ↔ Transfer (1:N):** Um `User` pode ser o remetente (`sender`) ou o destinatário (`recipient`) de múltiplas (`N`) `Transfer`s. Cada `Transfer`, no entanto, tem apenas um remetente e um destinatário. Isso é modelado com dois campos `ForeignKey` no modelo `Transfer`.
- **User ↔ Trade (1:N):** Um `User` pode realizar múltiplas (`N`) `Trade`s (operações de compra/venda). Cada `Trade` está associada a um único `User`.
- **Stock ↔ Trade (1:N):** Uma `Stock` (ação) pode estar presente em múltiplas (`N`) `Trade`s. Cada `Trade` refere-se a uma única `Stock`.

- **User ↔ FavoriteStock (1:N):** Um `User` pode ter várias (`N`) `FavoriteStock`s.
- **Stock ↔ FavoriteStock (1:N):** Uma `Stock` pode ser favoritada por múltiplos (`N`) `User`s. O relacionamento entre `User` e `Stock` através de `FavoriteStock` é, portanto, de Muitos-para-Muitos (N:M), com a tabela `FavoriteStock` atuando como a tabela de junção.

5 PADRÕES DE PROJETO

A arquitetura do sistema emprega diversos padrões de projeto, sendo o mais proeminente o **Model-View-Controller (MVC)**, que é a base do framework Django (embora o Django se refira a ele como **Model-View-Template (MVT)**).

- **Model-View-Controller (MVC/MVT):**
  - **Model (Modelos):** As classes `Account`, `Transfer`, `Stock`, `Trade` e `FavoriteStock` representam a camada de dados e lógica de negócio.
  - **View (Visões):** As funções e classes de view (`AdminPasswordResetView`, `dashboard`, etc.) atuam como o "Controller", processando as requisições do usuário, interagindo com os modelos e selecionando qual template renderizar.
  - **Template (Templates HTML):** Embora não detalhados neste documento, os templates HTML correspondem à "View" no padrão MVC tradicional, sendo responsáveis pela apresentação dos dados.
- **Mixin:** O padrão Mixin é utilizado em `AdminPasswordResetView` com `LoginRequiredMixin` e `UserPassesTestMixin` para adicionar funcionalidades de controle de acesso de forma reutilizável e desacoplada.
- **Form Object:** As classes de `Forms` (`TransferForm`, `TradeForm`) atuam como objetos de formulário, encapsulando a complexidade da validação de dados de entrada antes que eles afetem os modelos de domínio.

6 CONCLUSÃO

A arquitetura do sistema de controle de ações financeiras, construída sobre o framework Django, demonstra uma estrutura robusta e bem organizada, alinhada com as melhores práticas de desenvolvimento web. A clara separação de responsabilidades, imposta pelo padrão MVT, facilita a manutenção, o teste e a evolução do sistema.

Os modelos de dados representam de forma eficaz as entidades do domínio financeiro, e os relacionamentos entre eles são consistentes e bem definidos. A camada de formulários adiciona um nível crucial de segurança e integridade através de validações de negócio detalhadas, como a verificação de saldo e a posse de ativos antes de permitir transações. As visões orquestram o fluxo de interação do usuário de maneira lógica e eficiente.

A utilização de padrões de projeto consagrados confere flexibilidade e escalabilidade à aplicação. Conclui-se que a arquitetura apresentada é sólida e adequada para os requisitos de uma plataforma de controle financeiro, fornecendo uma base confiável para futuras expansões e aprimoramentos.

7 REFERÊNCIAS

1. ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724:** Informação e documentação — Trabalhos acadêmicos — Apresentação. Rio de Janeiro, 2011.
2. DJANGO SOFTWARE FOUNDATION. **Django Documentation**. Disponível em: https://docs.djangoproject.com/en/stable/. Acesso em: 24 jun. 2025.
3. BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **The Unified Modeling Language User Guide**. 2. ed. Addison-Wesley, 2005.