Documentação

(Entrega 02)

Projeto de Software

Professor: Fábio Morais

Grupo: 07

Lázaro Queiroz do Nascimento - 123110628
Matheus Galdino de Souza - 123111147
João Lucas Gomes Brandão - 123110397
Igor Raffael Menezes de Melo - 123110549
Vinícius de Oliveira Porto - 123110505

Descrição USs Implementadas

US06 - Notificação da Disponibilidade de Ativos para Interessados
Responsáveis: Lázaro (50%), Matheus (50%)

Descrição

Os métodos de criação, deleção e alteração de ativos foram realocados para um controller que possuísse contato com os services tanto de Usuario quanto de Ativo, dado que era necessário informações de um Usuario - o Administrador - para realizar dadas ações nos Ativos.

Foi adicionado o método adicionarInteressado, o qual é responsável por marcar interesse de um cliente por um dado ativo, através do armazenamento do ID do cliente em uma coleção específica (de disponibilidade ou variação de cotação) dentro do Ativo (Ação ou Criptomoeda). Ele é adicionado conforme a disponibilidade do ativo ou plano do cliente.

No método atualizarCotacao, responsável por alterar a cotação de um dado ativo (Ação ou Criptomoeda), foi implementada a lógica de notificação do interessado, quando realiza-se uma variação de mais de 10% do valor atual do ativo, o método notificarInteressado é invocado, que, por sua vez aciona o Logger e imprime na saída padrão avisos direcionados a cada um dos clientes premium interessados naquele ativo.

US07 - Notificação da Disponibilidade de Ativos para Interessados
Responsáveis: João Lucas (50%), Igor Raffael (50%)

Descrição

No método *ativarOuDesativar*, responsável por alternar a disponibilidade de um ativo, foi implementada a lógica que verifica seu status para atribuir o cliente à lista correspondente — interessados na variação da cotação ou na mudança de indisponível para disponível — visando notificá-los adequadamente.

Para os clientes interessados na segunda situação, a lista é esvaziada após a notificação, garantindo que ela seja enviada apenas uma vez.

Além disso, foram criados testes de integração no Controller intermediário de Ativo e Cliente para validar essa funcionalidade.

US08 - Visualização, por parte de um cliente, de informações de um ativo específico antes de realizar a compra
Responsáveis: Vinícius Porto (100%)

Descrição

Foi implementado o método visualizarAtivo, que permite ao cliente visualizar, antes da compra, informações detalhadas de um ativo, incluindo tipo, cotação atual, descrição e status de disponibilidade, conforme definido na US.

Também foram desenvolvidos testes de integração para validar o correto funcionamento do método para clientes dos planos normal e premium, cobrindo diferentes tipos de ativos (Tesouro Direto, criptomoeda e ação), garantindo que as informações retornadas atendam aos critérios estabelecidos.

Decisões de Design

(Futuramente) Padrão State no estado da compra e do resgate - Quando as USs responsáveis forem requisitadas.

(Futuramente) Padrão Observer na notificação dos ativos - Quando o assunto for ministrado pelo professor.

1. Estratégias de Modelagem de Ativos

- **Padrão Strategy:** utilizado para a definição dinâmica do comportamento de ativos com base em seu tipo (Ação, CriptoMoeda, TesouroDireto).
- **Conversão das interfaces para classes abstratas:** necessária para possibilitar a persistência com *JPA*, mantendo a extensibilidade e coesão da arquitetura.
- **Enum TipoAtivo exclusivo no DTO:** utilizado para reduzir o acoplamento entre os DTOs e a estrutura interna da aplicação.
- **Enum StatusDisponibilidade em vez de boolean:** melhora a legibilidade código e reduz a possibilidade de erros lógicos.

2. Integração entre Serviços

- **Facade entre AtivoService e ClienteService:** Criação da classe *AtivoClienteService* para intermediar a comunicação entre os serviços de Cliente e Ativo, especialmente para a US05, promovendo uma separação das responsabilidades e organização da lógica de negócio.
- **Repositórios separados para Cliente e Administrador:** apesar de existir apenas um administrador, foi decidida a separação dos repositórios para manter a clareza e a responsabilidade única de cada camada.

- **Autenticação simplificada:** embora tenha sido testada a criação de *um AutenticacaoService*, optou-se por verificar diretamente o *codigoAcesso* nos repositórios, de acordo com a autoridade exigida por cada funcionalidade.

3. Desacoplamento entre DTOs e Domínio

- O enum *TipoAtivo* é utilizado exclusivamente no DTO de atualização (Patch), evitando exposição da estrutura interna do domínio.
- Essa abordagem facilita a validação automática e garante segurança tipada com valores restritos.
- A lógica de ativação e desativação de ativos foi encapsulada dentro da própria entidade, centralizando as regras de negócio e promovendo a coesão.

4. Lógica Centralizada no Service

- Toda a lógica de negócio relativa à atualização de ativos foi centralizada na classe *AtivoServiceImpl*.
- O controller atua como uma camada de delegação, encaminhando as requisições ao serviço.
- Para evitar o uso de instanceof, a estrutura dos ativos foi modificada para incluir um enum *TipoAtivo*, permitindo o uso de métodos como *getTipo()* e *getNomeTipo()* diretamente, aumentando legibilidade e coesão.

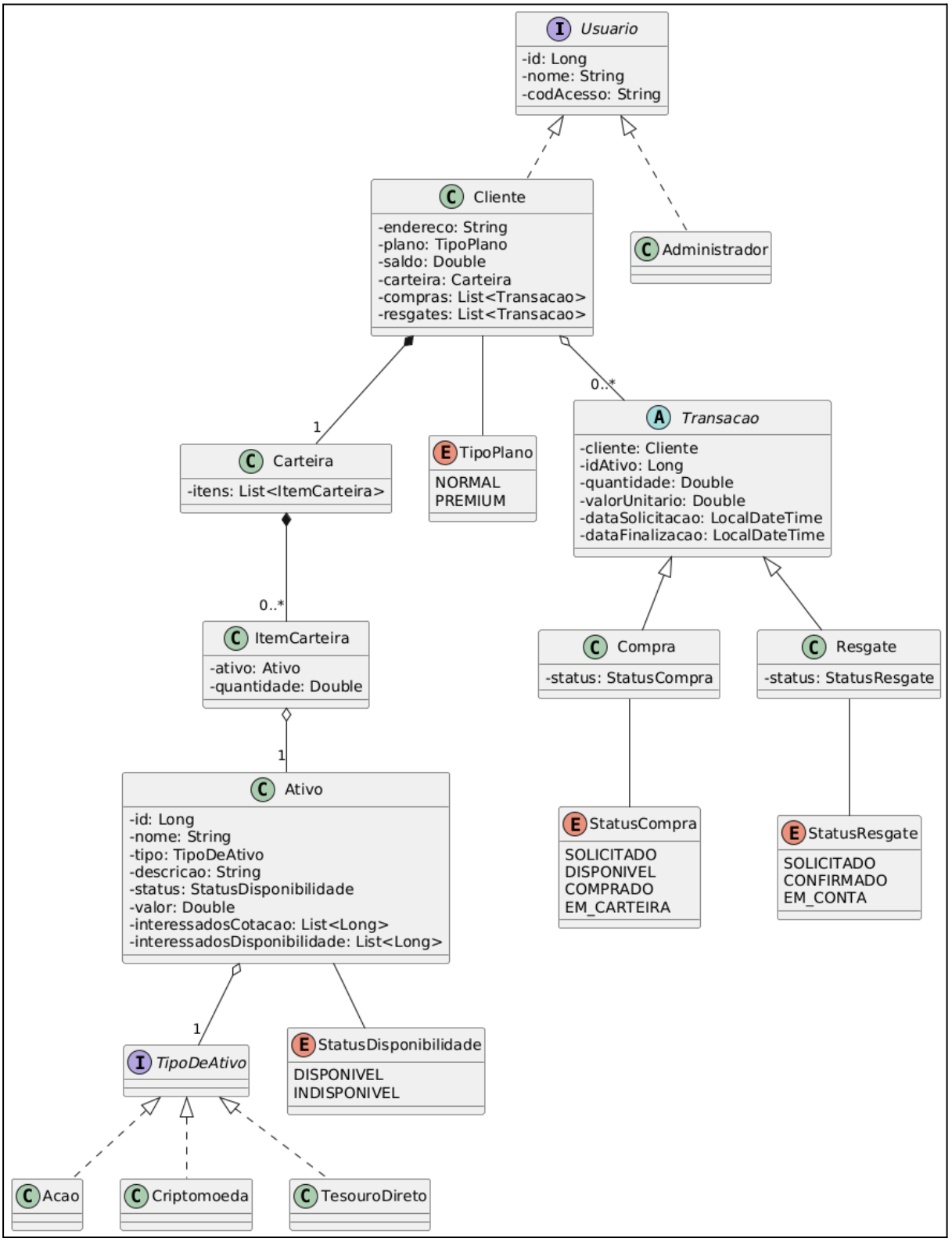
5. Herança e Estrutura de Usuários

- A classe abstrata *Usuario* agrupa atributos comuns a *Cliente* e *Administrador* (como id, nome e codigoAcesso).
- A herança é persistida com a estratégia *SINGLE_TABLE* do JPA, utilizando @DiscriminatorColumn para diferenciar os tipos de usuários.
  A classe *Cliente* representa a entidade persistente e está localizada no pacote model.
  Utiliza-se *ClientePostPutRequestDTO* para entrada de dados e *ClienteResponseDTO* para saída, evitando a exposição de informações sensíveis.
- O mapeamento entre DTOs e entidade é realizado com o ModelMapper, configurado na classe *ModelMapperConfig*, com typeMap específico que ignora o id durante atualizações.
- O enum *TipoPlano* define os diferentes planos de forma tipada e segura, garantindo consistência nos dados de domínio.

6. Composição de Funcionalidade entre Serviços

- Foi necessário utilizar serviços e controllers tanto de *Cliente* quanto de *Ativo*.
- Para evitar acoplamento excessivo e garantir organização, foi criado um novo controller que integra os dois serviços, funcionando como um ponto de orquestração das regras de negócio relacionadas à listagem de ativos conforme o plano do cliente.

Diagrama


Usuario
-id: Long
-nome: String
-codAcesso: String
Cliente
-endereco: String
-plano: TipoPlano
-saldo: Double
-carteira: Carteira
-compras: List<Transacao>
-resgates: List<Transacao>
Administrador
1
0..*
Carteira
-itens: List<ItemCarteira>
TipoPlano
NORMAL
PREMIUM
Transacao
-cliente: Cliente
-idAtivo: Long
-quantidade: Double
-valorUnitario: Double
-dataSolicitacao: LocalDateTime
-dataFinalizacao: LocalDateTime
0..*
ItemCarteira
-ativo: Ativo
-quantidade: Double
Compra
-status: StatusCompra
Resgate
-status: StatusResgate
1
Ativo
-id: Long
-nome: String
-tipo: TipoDeAtivo
-descricao: String
-status: StatusDisponibilidade
-valor: Double
-interessadosCotacao: List<Long>
-interessadosDisponibilidade: List<Long>
StatusCompra
SOLICITADO
DISPONIVEL
COMPRADO
EM_CARTEIRA
StatusResgate
SOLICITADO
CONFIRMADO
EM_CONTA
1
TipoDeAtivo
StatusDisponibilidade
DISPONIVEL
INDISPONIVEL
Acao
Criptomoeda
TesouroDireto