Projeto de Bloco: Engenharia de Softwares Escaláveis [25E3_5]

TP02

Aluno: André Augusto Ferrarez

Prof°: Leonardo Silva da Gloria

26/09/2025

SUMÁRIO

Link
https://github.com/AndreFerrarez/transacoes

Segunda Entrega: Desenvolver uma Camada de Persistência Real

1. **Modelagem de Dados:**

A solução proposta é manter uma tabela separada **“transacao_historico”** (modelo simples, explícito) e gravar nela sempre que o service executar operações que alterem o estado (salvar/excluir). Essa abordagem é simples, transparente e funciona sem dependências extras (ex.: Envers).

Explicação: A classe **“TransacaoHistorico”** armazena uma "foto" da transação no momento da operação, com um campo **“operacao”** para identificar o tipo de evento (CREATE/UPDATE/DELETE). Mantive a **“transacaoId”** para ligar ao **“Transacao”** original.

Foi feito também uma melhoria na classe **“Transacao”** para incluir construtores e garantir que a **“data”** seja persistida corretamente.

Explicação: adicionei anotações de validação (@NotNull, @Positive) para reforçar integridade em nível de aplicação (o Spring Validation irá usá-las no controller se ativarmos). Mantive a **“data”** com valor padrão de now().

2. **Integração de JPA com Spring Data:**

Foi adicionado um repositório Spring Data para “**Transacao”** (com métodos de consulta úteis) e criei a interface **“TransacaoHistorico”** com métodos úteis para consultar histórico por “**transacaoId”** e por “**moeda”**.

Explicação: adicionei ao “TransacaoRepository” os metodos findByMoedaIgnoreCase e findByTipoIgnoreCase para consultas mais eficientes no serviço (evitar findAll().stream().filter(...)).
E a “TransacaoHistoricoRepository” foi adicionado **findByTransacaoIdOrderByDataOperacaoDesc**

e **findByMoedaIgnoreCaseOrderByDataOperacaoDesc** para as consultas de transacaoId e moeda.
Quando delegado ao banco, ganhamos performance e possibilidade futura de paginação.

3. **Gerenciamento de Dados:**

Garantir que sempre que criamos/atualizamos/excluímos uma transação, gravamos uma entrada no histórico.

Explicação detalhada:

- A lógica de gravação do histórico foi colocada no Service (não em EntityListener) porque o service é gerenciado pelo Spring e tem acesso aos repositórios — isso simplifica a injeção e transações.

- O método salvar detecta se é CREATE (id == null) ou UPDATE e grava um TransacaoHistorico correspondente.

- O método excluir primeiro busca a entidade (snapshot), grava histórico com operacao = "DELETE", e só então exclui.

- Usei @Transactional nos métodos que gravam histórico + alteram dados para garantir atomicidade: ou salva transação + histórico ambos, ou falha tudo.

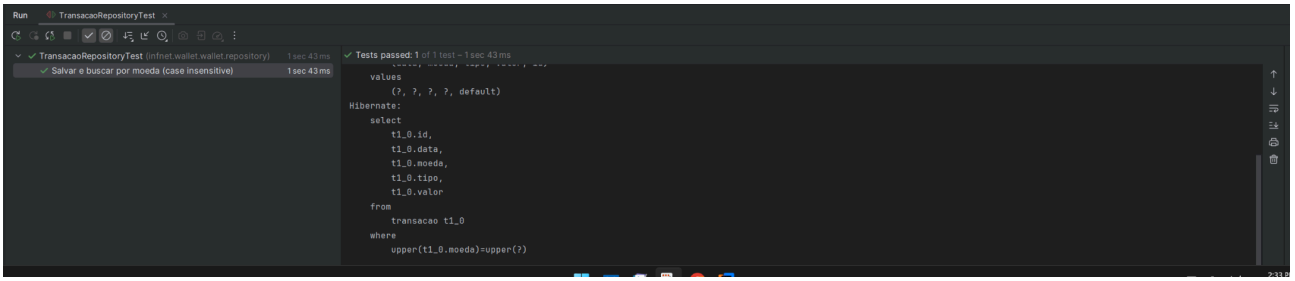
4. **Integração de Funcionalidades de Histórico de Dados:**

Foi atualizado a controller “TransacaoController” para expor endpoints que consultem histórico “**transacaoId”** e por **“moeda”**.

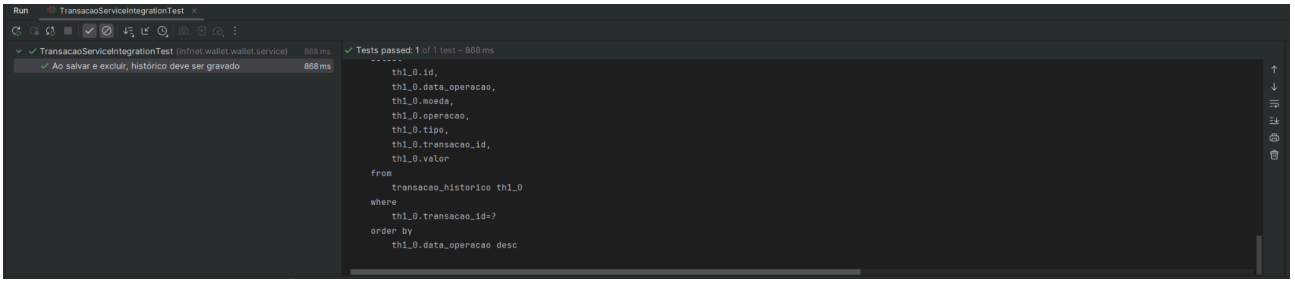
5. **Implementação de Testes:**

Foi implementado testes que validem a camada de persistência (repositórios e o fluxo que grava histórico).
Dois testes principais:

- TransacaoRepositoryTest com @DataJpaTest para operações básicas e método findByMoedaIgnoreCase

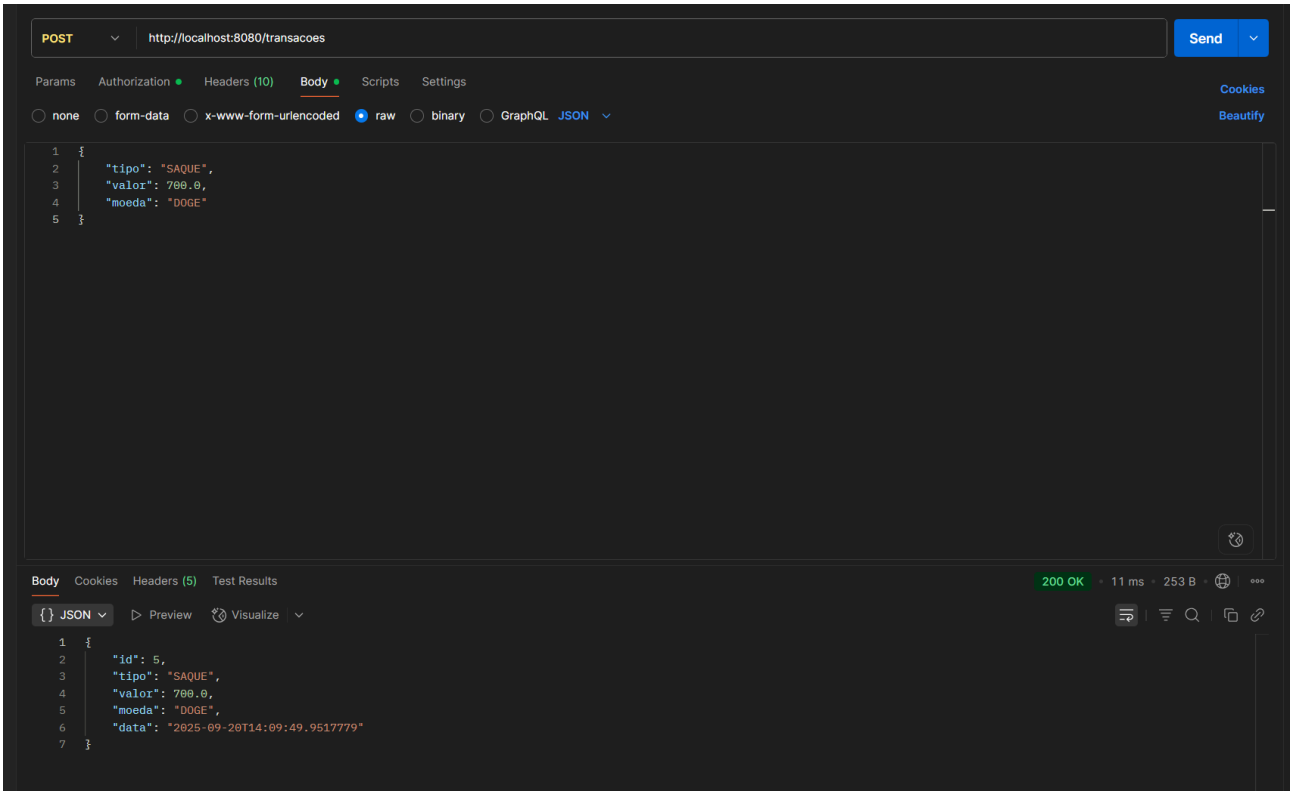


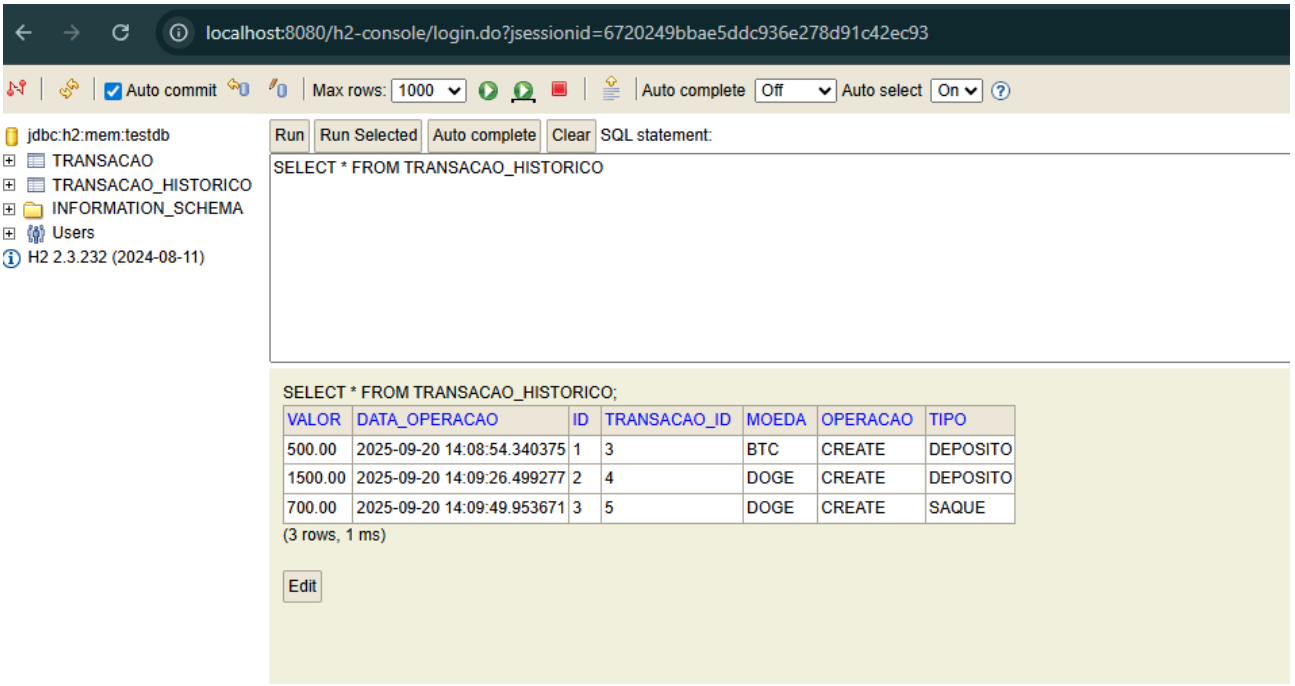
- TransacaoServiceIntegrationTest com @SpringBootTest para validar que salvar/excluir grava o histórico.



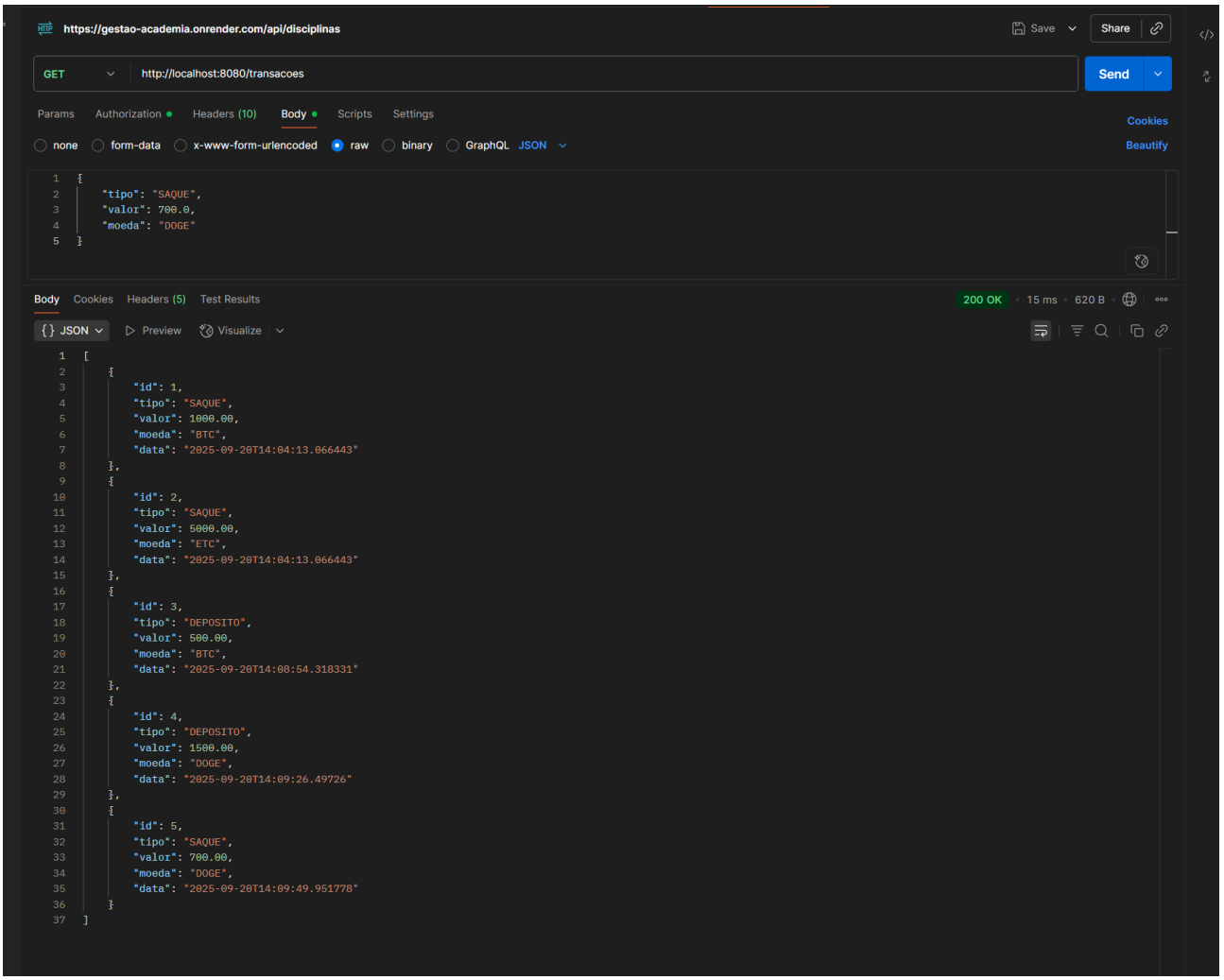
Prints que representam a aplicação rodando em H2 para fins de teste local, confirmando a implementação do histórico.
Mantendo assim a aplicação com rastreabilidade de cada operação feita.

POST api/transacoes + JSON

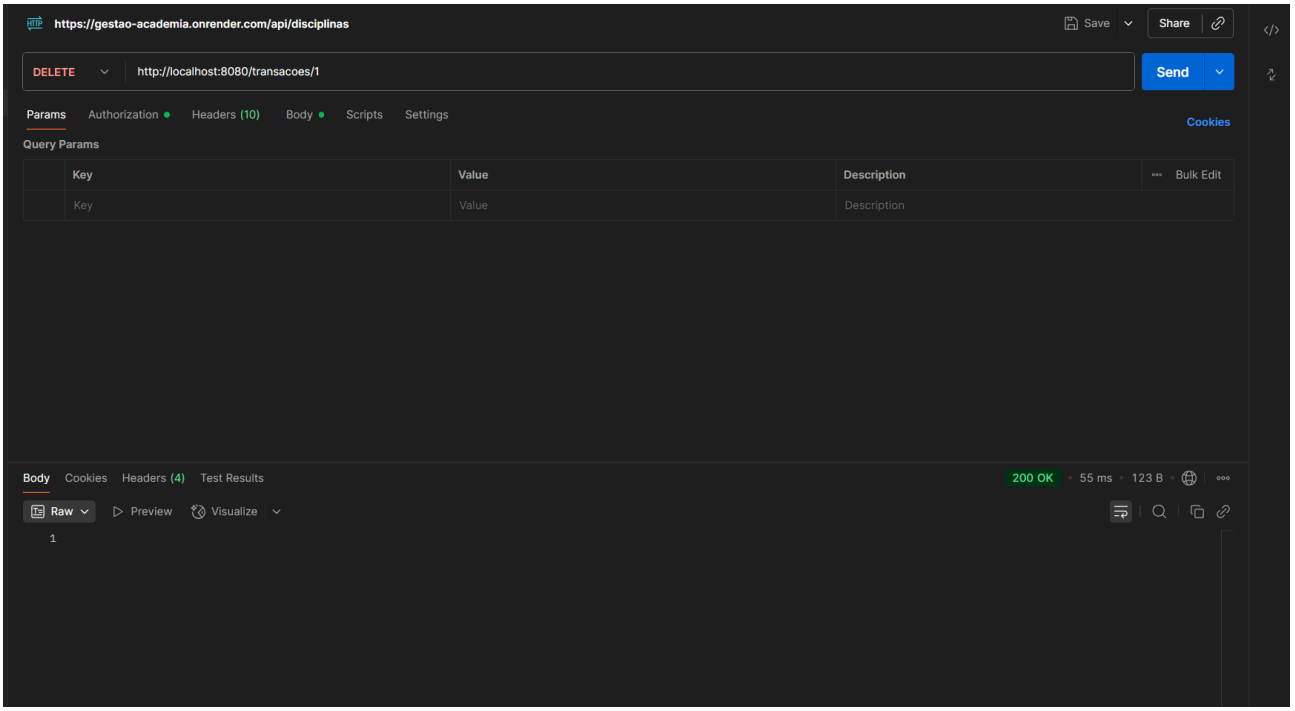


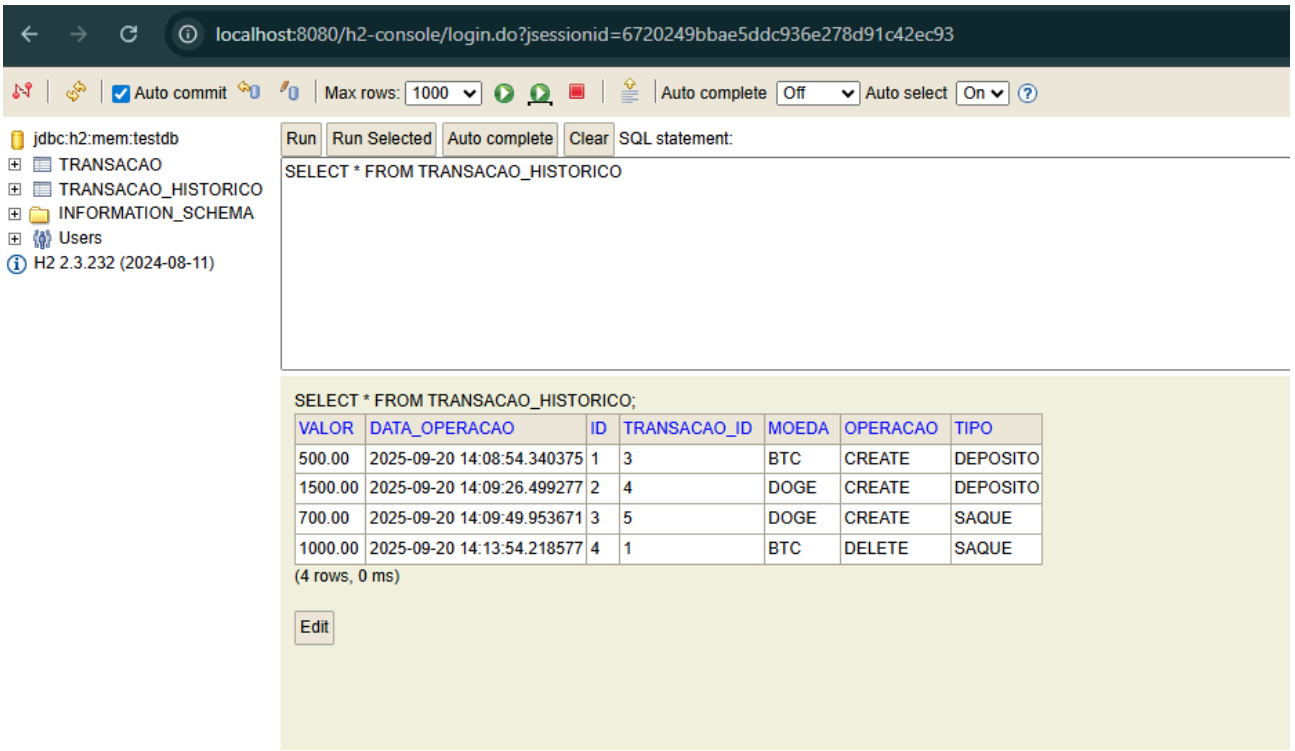


GET api/transacos



DELETE api/transacoes/{id}





GET(histórico) api/transacoes/historico/moeda/BTC