Documentação do Sistema de Transações

Descrição do Projeto
Esta aplicação é uma API de gerenciamento de transações, que permite receber, listar, obter estatísticas e remover transações. A API é desenvolvida em Java utilizando o Spring Boot.

Pré-requisitos

-Linguagem**: Java 17+
-Ferramentas Necessárias:
 - Docker (opcional, para execução via container)

Instruções de Instalação

Clonando o Repositório

1. Clone o repositório:
   bash
   git clone -b feat/gustavo https://github.com/rafaellarosa07/desafio-tecnico.git
  cd desafioAPITransacao

Configuração Inicial

1. Usando o Maven Wrapper:
   - O projeto inclui o Maven Wrapper, então você não precisa ter o Maven instalado no seu sistema. Você pode usar os comandos abaixo para construir e executar a aplicação:

   - Para Linux/macOS:
     bash
     ./mvnw clean install
     ./mvnw spring-boot:run

   - Para Windows:
     bash
     mvnw.cmd clean install
     mvnw.cmd spring-boot:run

Instruções de Construção

1. Construindo o Projeto com Maven Wrapper:
   - Para Linux/macOS:
     bash
     ./mvnw clean install

   - Para Windows:
     bash
     mvnw.cmd clean install

Instruções de Execução

Executando Localmente

1. Executando a aplicação com Maven Wrapper:
   - Para Linux/macOS:
     bash
     ./mvnw spring-boot:run

   - Para Windows:
     bash
     mvnw.cmd spring-boot:run

   A aplicação estará disponível em `http://localhost:8080`.

Executando com Docker

1. Construindo a imagem Docker:
   ```bash
   docker build -t desafio-api-transacao .
   ```

2. Executando o container Docker
   bash
   docker run -p 8080:8080 desafio-api-transacao

Instruções de Teste

1. Executando os testes com Maven Wrapper:
   - Para Linux/macOS
     bash
     ./mvnw test

   - Para Windows
     bash
     mvnw.cmd test

Endpoints da API

 POST /transacao`
- Descrição**: Recebe uma nova transação.
- Parâmetros de Entrada
  - `valor` (double): Valor da transação (não pode ser negativo).
  - `dataHora` (OffsetDateTime): Data e hora da transação (não pode estar no futuro).
- Exemplo de Requisição
   bash (ou postman utilizando a url e passando os parametros no body)
   curl -X POST http://localhost:8080/transacao -H "Content-Type: application/json" -d '{"valor": 100.0, "dataHora": "2023-06-09T12:34:56Z"}'
   ```
- Respostas
  - `201 Created`: Transação criada com sucesso.
  - `422 Unprocessable Entity`: Se o valor da transação for negativo ou a data estiver no futuro.
  - `400 Bad Request`: Para outros erros de validação.

`GET /transacao`
- **Descrição**: Lista todas as transações.
- Exemplo de Requisição
   bash/postman
   curl -X GET http://localhost:8080/transacao
   ```
- Respostas
  - `200 OK`: Lista de transações retornada com sucesso.
  - `404 Not Found`: Se não houver transações cadastradas.

`GET /transacao/estatistica`
- Descrição**: Obtém estatísticas das transações nos últimos N segundos.
- Parâmetros de Query
  - `tempo` (int, default: 60): Número de segundos para considerar nas estatísticas, utilizar o parâmetro query especificamente quando o valor for diferente de ‘60’.
- Exemplo de Requisição com tempo customizado
    bash
    curl -X GET "http://localhost:8080/transacao/estatistica?tempo=30"
    ```
- Exemplo de Requisição sem parâmetro `tempo` (valor padrão = 60
    bash
    curl -X GET http://localhost:8080/transacao/estatistica
    ```
- Respostas
  - `200 OK`: Estatísticas retornadas com sucesso.
  - `404 Not Found`: Se não houver transações no intervalo de tempo especificado.

`DELETE /transacao/delete`
- **Descrição**: Remove todas as transações.
- **Exemplo de Requisição**:
    bash
    curl -X DELETE http://localhost:8080/transacao/delete
    ```
- **Respostas**:
  - `200 OK`: Todas as transações foram removidas com sucesso.
  - `400 Bad Request`: Em caso de erro ao tentar remover as transações.
    ```