desafio-itau-backend

Itaú Unibanco - Desafio de Programação

Este é um desafio bacana tanto de desenvolvimento de software quanto de engenharia de software. Queremos testar sua capacidade de construir um software com várias partes diferentes funcionando em conjunto!

Vídeo no youtube https://youtu.be/uke3i4uOejs

1. Introdução

Sua missão, caso você aceite, é criar uma API REST que recebe Transações e retorna Estatísticas sob essas transações. Para este desafio, a API deve ser criada utilizando-se de Java ou Kotlin e Spring Boot.

Um bom lugar para se começar é o Spring Starter.

Dica: Não existe uma forma certa ou errada de resolver o desafio! Vamos avaliar coisas como a qualidade do seu código, o quão fácil é de compreender o código, organização do projeto, quantidade e qualidade dos testes, preocupação com segurança e vários outros fatores :)

2. Definição do desafio

Neste desafio você deve **criar uma API REST** no GitHub ou GitLab. **Leia com atenção todas as instruções a seguir!**

2.1. Restrições Técnicas

Seu projeto:

- **DEVE** estar no GitHub ou GitLab
- **NÃO DEVE** fazer *fork* de nenhum outro projeto
- **DEVE** ter pelo menos 1 commit por cada endpoint (mínimo de 3 commits)
    - Queremos ver a evolução do seu projeto com o tempo ;)
- Todos os commits **DEVEM** ser feitos pelo mesmo usuário que criou o projeto
    - Entendemos que algumas pessoas tem usuários pessoais e profissionais, ou um usuário diferente usado para estudar. Atenção com isso se você for uma dessas pessoas!
- **DEVE** seguir exatamente os *endpoints* descritos a seguir
    - Por exemplo, /transacao não é a mesma coisa que /transacoes
- **DEVE** aceitar e responder com objetos exatamente como descritos a seguir
    - Por exemplo, dataHora não é a mesma coisa que data-hora ou dtTransacao
- **NÃO DEVE** utilizar quaisquer sistemas de banco de dados (como H2, MySQL, PostgreSQL, ...) ou cache (como Redis, Memcached, Infinispan, ...)

- **DEVE** armazenar todos os dados **em memória**
- **DEVE** aceitar e responder apenas com JSON

Atenção! Por motivos de segurança, não podemos aceitar projetos enviados como arquivos. Você **DEVE** disponibilizar seu projeto publicamente para que possamos acessá-lo e corrigi-lo! Após receber uma resposta de nós, sinta-se livre para tornar seu projeto **privado** :)

2.2. Endpoints da API

A seguir serão especificados os endpoints que devem estar presentes na sua API e a funcionalidade esperada de cada um deles.

2.2.1. Receber Transações: POST /transacao

Este é o endpoint que irá receber as Transações. Cada transação consiste de um valor e uma dataHora de quando ela aconteceu:

```
{
  "valor": 123.45,
  "dataHora": "2020-08-07T12:34:56.789-03:00"
}
```

Os campos no JSON acima significam o seguinte:

Campo	Significado	Obrigatório?
valor	**Valor em decimal com ponto flutuante** da transação	Sim
dataHora	**Data/Hora no padrão ISO 8601** em que a transação aconteceu	Sim

Dica: O Spring Boot, por padrão, consegue compreender datas no padrão ISO 8601 sem problemas. Experimente utilizar um atributo do tipo OffsetDateTime!

A API só aceitará transações que:

1. Tenham os campos valor e dataHora preenchidos
2. A transação **NÃO DEVE** acontecer no futuro
3. A transação **DEVE** ter acontecido a qualquer momento no passado
4. A transação **NÃO DEVE** ter valor negativo
5. A transação **DEVE** ter valor igual ou maior que 0 (zero)

Como resposta, espera-se que este endpoint responda com:

- 201 Created sem nenhum corpo
    - A transação foi aceita (ou seja foi validada, está válida e foi registrada)

- 422 Unprocessable Entity sem nenhum corpo
    - A transação **não** foi aceita por qualquer motivo (1 ou mais dos critérios de aceite não foram atendidos - por exemplo: uma transação com valor menor que 0)
- 400 Bad Request sem nenhum corpo
    - A API não compreendeu a requisição do cliente (por exemplo: um JSON inválido)

2.2.2. Limpar Transações: DELETE /transacao

Este endpoint simplesmente **apaga todos os dados de transações** que estejam armazenados.

Como resposta, espera-se que este endpoint responda com:

- 200 OK sem nenhum corpo
    - Todas as informações foram apagadas com sucesso

2.2.3. Calcular Estatísticas: GET /estatistica

Este endpoint deve retornar estatísticas das transações que **aconteceram nos últimos 60 segundos (1 minuto)**. As estatísticas que devem ser calculadas são:

```
{
  "count": 10,
  "sum": 1234.56,
  "avg": 123.456,
  "min": 12.34,
  "max": 123.56
}
```

Os campos no JSON acima significam o seguinte:

Campo	Significado	Obrigatório?
count	**Quantidade de transações** nos últimos 60 segundos	Sim
sum	**Soma total do valor** transacionado nos últimos 60 segundos	Sim
avg	**Média do valor** transacionado nos últimos 60 segundos	Sim
min	**Menor valor** transacionado nos últimos 60 segundos	Sim

Campo	Significado	Obrigatório?
max	**Maior valor** transacionado nos últimos 60 segundos	Sim

Dica: Há um objeto no Java 8+ chamado DoubleSummaryStatistics que pode lhe ajudar ou servir de inspiração.

Como resposta, espera-se que este endpoint responda com:

- 200 OK com os dados das estatísticas
    - Um JSON com os campos count, sum, avg, min e max todos preenchidos com seus respectivos valores
    - **Atenção!** Quando não houverem transações nos últimos 60 segundos considere todos os valores como 0 (zero)

4. Extras

Vamos propôr a seguir alguns desafios extras caso você queira testar seus conhecimentos ao máximo! Nenhum desses requisitos é obrigatório, mas são desejados e podem ser um diferencial!

1. **Testes automatizados:** Sejam unitários e/ou funcionais, testes automatizados são importantes e ajudam a evitar problemas no futuro. Se você fizer testes automatizados, atente-se na efetividade dos seus testes! Por exemplo, testar apenas os "caminhos felizes" não é muito efetivo.
2. **Containerização:** Você consegue criar meios para disponibilizar sua aplicação como um container? *OBS: Não é necessário publicar o container da sua aplicação!*
3. **Logs:** Sua aplicação informa o que está acontecendo enquanto ela trabalha? Isso é útil para ajudar as pessoas desenvolvedoras a solucionar eventuais problemas que possam ocorrer.
4. **Observabilidade:** Sua API tem algum endpoint para verificação da saúde da aplicação (healthcheck)?
5. **Performance:** Você consegue estimar quanto tempo sua aplicação gasta para calcular as estatísticas?
6. **Tratamento de Erros:** O Spring Boot dá às pessoas desenvolvedoras ferramentas para se melhorar o tratamento de erros padrão. Você consegue alterar os erros padrão para retornar *quais* erros ocorreram?
7. **Documentação da API:** Você consegue documentar sua API?
   Existem ferramentas e padrões que podem te ajudar com isso!
8. **Documentação do Sistema:** Sua aplicação provavelmente precisa ser construída antes de ser executada. Você consegue documentar como outra pessoa que pegou sua aplicação pela primeira vez pode construir e executar sua aplicação?

9. **Configurações:** Você consegue deixar sua aplicação configurável em relação a quantidade de segundos para calcular as estatísticas? Por exemplo: o padrão é 60 segundos, mas e se o usuário quiser 120 segundos?