Code Challenge: Autorizador

Você deverá implementar uma aplicação que autoriza transações para uma conta específica, seguindo uma série de regras predefinidas.

Por favor leia as instruções abaixo e sinta-se à vontade para fazer perguntas caso ache necessário.

⚠️ **IMPORTANTE:** Por favor remova toda informação que possa lhe identificar nos arquivos da solução para o desafio que você enviar. Atenção especial para os seguinte:

- Arquivos da solução como código, testes, namespaces, binários, comentários e nomes dos arquivos;
- Configuração do versionador de código como a pasta de configuração do git por exemplo;
- Comentários automáticos que seu editor de código pode ter adicionado aos arquivos;
- Documentação do código como annotations, metadata, e README.MD.

Preparando seu desafio para envio

Sua solução deve conter um arquivo de README com a descrição das suas decisões de design de código que forem relevantes, junto das instruções de como compilar (build) e executar a sua aplicação.

O processo de build e execução da sua aplicação deve ser possível num sistema operacional Unix ou Mac. Builds conteinerizadas são bem vindas.

Você pode utilizar bibliotecas de código aberto (open source) que acredite serem adequadas para ajudar na solução do problema, mas por favor tente limitar o uso de frameworks e boilerplate code desnecessários.

Exemplo de uso do Autorizador

Como o programa deve funcionar?

Seu programa receberá como entrada linhas em formato `json` na entrada padrão (`stdin`) e deve fornecer uma saída em formato `json` para cada uma das entradas - imagine isso como um fluxo de eventos chegando ao autorizador.

Como o programa deve ser executado?

Dado um arquivo chamado `operations` que contém diversas linhas descrevendo operações no formato `json`:

```
$ cat operations
{"account": {"active-card": true, "available-limit": 100}}
{"transaction": {"merchant": "Burger King", "amount": 20, "time": "2019-02-13T10:00:00.000Z"}}
{"transaction": {"merchant": "Habbib's", "amount": 90, "time": "2019-02-13T11:00:00.000Z"}}
```

A aplicação deve ser capaz de receber o conteúdo do arquivo via `stdin`, e para cada operação processada fornecer um output adequado de acordo com a lógica de negócio:

```
$ authorize < operations

{"account": {"active-card": true, "available-limit": 100}, "violations": []}
{"account": {"active-card": true, "available-limit": 80}, "violations": []}
{"account": {"active-card": true, "available-limit": 80}, "violations": ["insufficient-limit"]}
```

Estado da aplicação

O programa **não deve depender** de nenhum banco de dados externo. O estado interno da aplicação deve ser gerenciado em memória explicitamente por alguma estrutura que achar adequada. O estado da aplicação deve estar vazio sempre que a aplicação for inicializada.

Operações do Autorizador

O programa deve lidar com dois tipos de operações, decidindo qual delas executar de acordo com a linha que estiver sendo processada:

1. Criação da conta
2. Autorização de uma transação na conta

Para simplificar o programa, você pode assumir que:

- Todos os valores monetários são inteiros positivos, portanto é uma moeda sem centavos;
- As transações na conta chegarão no autorizador em ordem cronológica.

---

1. Criação da conta

Entrada

Cria a conta com os atributos `available-limit` (limite disponível) e `active-card` (cartão ativo). Para simplificar o programa, vamos assumir que o Autorizador lidará com apenas uma conta.

Saída

O estado atual da conta criada junto de quaisquer violações da lógica de negócios. Se não houverem violações no processamento da operação, o campo `violations` deve retornar um vetor vazio `[]`.

Violações da lógica de negócios

Uma vez criada, a conta não deve ser atualizada ou recriada. Se o aplicativo receber uma segunda instrução de criação de conta, ele deve retornar a seguinte violação: `account-already-initialized` (Conta já foi inicializada).

Exemplos

```
# Input
    {"account": {"active-card": true, "available-limit": 100}}
    {"account": {"active-card": true, "available-limit": 350}}

# Output
    {"account": {"active-card": true, "available-limit": 100}, "violations":
[]}
    {"account": {"active-card": true, "available-limit": 100}, "violations":
["account-already-initialized" ]}
```

---

2. Autorização de transação

Entrada

Tenta autorizar uma transação para um determinado `merchant` (comerciante), `amount` (valor da transação) e `time` (horário da transação) de acordo com o estado da conta criada e **as últimas transações que foram autorizadas**.

Saída

O estado atual da conta junto de quaisquer violações da lógica de negócios.

Violações da lógica de negócios

Você deve implementar as seguintes regras, tendo em mente que **novas regras aparecerão no futuro**:

- Nenhuma transação deve ser aceita sem que a conta tenha sido inicializada: `account-not-initialized`
- Nenhuma transação deve ser aceita quando o cartão não estiver ativo: `card-not-active`
- O valor da transação não deve exceder o limite disponível: `insufficient-limit`
- Não deve haver mais que 3 transações de qualquer comerciante em um intervalo de 2 minutos: `high-frequency-small-interval`
- Não deve haver mais que 1 transação similar (mesmo valor e comerciante) no intervalo de 2

minutos: `double-transaction`

Exemplos

Processando uma transação com sucesso

Dado que há uma conta com cartão ativo (`active-card: true`) e limite disponível de 100 (`available-limit: 100`):

```
# Input
    {"transaction": {"merchant": "Burger King", "amount": 20, "time": "2019-02-13T10:00:00.000Z"}}

# Output
    {"account": {"active-card": true, "available-limit": 80}, "violations": []}
```

Processando uma transação que viola a lógica do Autorizador

Dado que há uma conta com cartão ativo (`active-card: true`), limite disponível de 80 (`available-limit: 80`), e 3 transações ocorreram nos úlltimos 2 minutos:

```
# Input
    {"transaction": {"merchant": "Habbib's", "amount": 90, "time": "2019-02-13T10:01:00.000Z"}}

# Output
    {"account": {"active-card": true, "available-limit": 80}, "violations": ["insufficient-limit", "high-frequency-small-interval"]}
```

Lidando com erros

- Por favor assuma que não ocorrerão erros na conversão do `json` de entrada. Na avaliação da sua solução nós não vamos utilizar entradas que contenham erros, estejam mal formatadas, ou que quebrem o contrato.
- As violações das regras de negócios **não são consideradas erros** porque se espera que elas ocorram e devem ser listadas no campo `violations` (violações) das saídas conforme descrito nos esquemas de `output` (saída) dos exemplos. Isso significa que a execução do programa deve continuar normalmente após qualquer tipo de violação.

Nossas Expectativas

Nós do Nubank valorizamos **código simples, elegante e que funcione**. Esse exercício deve refletir sua compreensão a respeito. É esperado que:

- Sua solução seja de **qualidade**;
- A aplicação seja **sustentável**, sendo de fácil manutenção e bem estruturada;
- A aplicação seja **extensível**, sendo fácil a implementação de novas lógicas de negócio.

Desta forma, procuraremos avaliar:

- O bom uso de imutabilidade no código;
- Testes de unidade e integração de qualidade;
- Documentação onde for necessário;
- Instruções sobre como executar o código.

Notas gerais

- Esse desafio poderá ser estendido por você e por outra pessoa engenheira do Nubank durante uma outra etapa do processo;
- Você deve entregar o código fonte de sua solução para nós em um arquivo comprimido (zip) contendo o código e toda documentação possível. Favor não incluir arquivos desnecessários como binários compilados, bibliotecas, etc;
- Não faça o upload da sua solução em nenhum repositório público como GitHub, BitBucket, etc;
- O projeto deve ser implementado como um processador de streams via entrada padrão (stdin) ao invés de uma API REST.