Desafio de código: Autorizador de Transações Financeiras

Você está encarregado de implementar uma aplicação que autoriza uma transação para uma “conta corrente” específica, seguindo um conjunto de regras predefinidas.

Nossas expectativas:

Valorizamos um código simples, elegante e funcional. Este exercício deve refletir sua compreensão disso.

Espera-se que sua solução seja de qualidade de produção, sustentável e extensível. Por isso, vamos procurar:

- Imutabilidade;
- Testes de unidade e integração;
- Documentação onde necessário;
- Instruções para executar o código;

Notas gerais:

- **Não se preocupe em finalizar por completo o exercício, apenas use o tempo para entregar o máximo que conseguir, na melhor qualidade possível. Não haverá deméritos apenas pelo fato do exercício não estar completo.**

- O desafio deve ser escrito utilizando Javascript ou Typescript com NodeJS;
- O uso de frameworks está autorizado como achar necessário;
- Este desafio pode ser estendido por você na companhia de um desenvolvedor interno em outra etapa do processo;
- Você deve enviar o código-fonte da solução para nós como um arquivo compactado contendo o código e a possível documentação. Por favor, certifique-se de não incluir arquivos desnecessários, como binários compilados, bibliotecas, etc;
- Não envie sua solução para repositórios públicos no GitHub, BitBucket, etc;
- O projeto deve ser implementado como um aplicativo de fluxo contínuo em vez de um Rest API.

Packing

Seu arquivo README deve conter uma descrição das opções de design de código relevantes, junto com instruções sobre como fazer build e executar seu aplicativo.

Executar o aplicativo deve ser possível em Windows, ou Linux ou Mac.
Aplicações "dockerizadas" são bem-vindas, mas não obrigatórias.

Você pode usar bibliotecas de código aberto que achar adequadas, mas evite o máximo possível adicionar estruturas e códigos clichês desnecessários.

Uso de amostra

Seu programa receberá linhas `json` como entrada no stdin, e deve fornecer uma saída em linha `json` para cada linha recebida -> imagine isso como um fluxo de eventos chegando ao autorizador.

Entrada:

```
$ 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"}}
```

Saída:

```
$ authorizer < 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 precisa depender de nenhum banco de dados externo. O estado da aplicação pode ser tratado diretamente em memória. O estado deve ser redefinido/zerado no início da aplicação.

Operações

O programa lida com dois tipos de operações, decidindo qual de acordo com a linha que está sendo processada:

1. Criação de conta
2. Autorização de transação

Para simplificar, você pode assumir:

- Todos os valores monetários são inteiros positivos usando uma moeda sem centavos
- As transações chegarão em ordem cronológica
- Todas as linhas estarão corretas, sem quebra de contrato do json.

---

1. Criação de conta:

Entrada:
Cria a conta com limite disponível e indicativo de cartão ativo. Para simplificar, nós assumimos que o aplicativo lidará com apenas uma única conta.

Saída:
O estado atual da conta criada + todas as violações da lógica de negócios.

Regras do negócio:
Depois de criada, a conta não deve ser atualizada ou recriada: `account-already-initialized`

Exemplos:

```
Entrada:
{"account": {"active-card": true, "available-limit": 100}}
...
{"account": {"active-card": true, "available-limit": 350}}

Saída:
{"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 comerciante, com o valor e data/hora da transação.

Saída:
O estado atual da conta + quaisquer violações da lógica de negócios.

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

- Nenhuma transação deve ser aceita sem uma conta devidamente inicializada: `account-not-initialized`
- Nenhuma transação deve ser aceita quando o cartão não está ativo: `card-not-active`
- O valor da transação não deve exceder o limite disponível: `insufficient-limit`
- Não deve haver mais de 1 transação semelhante (mesmo valor e comerciante) em um intervalo de 2 minutos: `doubled-transaction`

Exemplos:

Dado que existe uma conta com `active-card`=true e `available-limit`=100:

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

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

Dado que existe uma conta com `active-card`=true, `available-limit`=80 e 2 transações com o mesmo valor e comerciante ocorreram nos últimos 2 minutos:

Endrada:

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

Saída:

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

Tratamento de erros:

- Assuma que não ocorrerão erros de schema do json de entrada. Não iremos avaliar o seu código contra entrada que quebra o contrato json.

- Violações das regras de negócios não são consideradas erros. Como é esperado que aconteça e deve ser listado no campo de violações das saídas `(violations)` como descrito no esquema de saída nos exemplos. Isso significa que o programa deve continuar executando normalmente após qualquer violação.

IMPORTANTE: anonimize / remova todas as informações pessoais presentes em seu desafio, atenção especial a:

● Arquivos fonte: código, testes, namespaces, empacotamento, comentários e nomes de arquivo;
● Informações sobre o autor do controle de versão;
● Comentários automáticos que seu ambiente de desenvolvimento pode adicionar;
● Documentação fornecida, como arquivos README.MD