**Case Cortex**

Os exercícios são baseados em parametrizações que serão solicitadas na hora da apresentação. **Dica: Quanto menos hardcoded, melhor :)**

**Observações:**

* Você pode utilizar um banco de dados relacional ou não relacional como PostgreSQL, Oracle, MYSQL, H2, HSQLDB, MongoDB, dentro outros, arquivos em disco, estrutura em memória ou qualquer outro serviço relacionado desejado
* Ao enviar o código final, deve estar claro como rodar os projetos feitos e quais serviços precisam ser configurados/acessados
* Preferencialmente, recomendamos uso de NodeJS, Python ou Java mas você pode utilizar qualquer linguagem e/ou tecnologia que queira
* Se você disponibilizar os serviços em algum lugar para rodarmos diretamente, como um Heroku, você ganha pontos extras
* Caso não consiga terminar algum exercício, sinalize no readme a dificuldade encontrada e/ou o que você faria diferente se tivesse mais tempo

**Exercício 1:**

Construa um serviço de conversão de moedas. Este serviço deve obter os dados de cotação do site do **Banco Central do Brasil.** Deve ser possível realizar uma chamada a um método da API do serviço com parâmetros de **data da cotação** (ex: 10/09/2019), **moeda de origem** (ex: EUR), **moeda final** (ex: USD) e **valor desejado** (ex: 150). O resultado desta chamada é o **valor convertido** para a moeda final

**Exercício 2:**

Faça um esquema de uma arquitetura utilizando o serviço feito no exercício 1.

A arquitetura deve atendar os requisitos abaixo:

* Deve existir um sistema de cache. O cache deve ter validade de 30min e não deve ter sua validade renovada com novas requisições. Ex: se eu pedir ao serviço uma conversão do valor 150 de EUR para USD no dia 10/092019, o resultado deve ficar cacheado por 30min, considerando essa a minha primeira requisição. Requisições em até 30min devem ser respondidas pelo sistema de cache, para os mesmos inputs. Depois desse tempo, uma nova requisição deve ser feita para o serviço. A resposta recebida deve indicar se a mesma veio do sistema de cache ou se foi um resultado live, feito na hora pelo serviço.
* Deve existir um sistema de fila para o serviço. Deve ser possível estabelecer uma prioridade para a fila, isto é, deve ser possível dizer que um determinado item pode “furar” a fila e ser atendido primeiro.

O resultado desse exercício deve ser um diagrama, um esquema. Você pode complementar com um texto, explicando como os itens da arquitetura se comunicam e como pensou para fazer o desenho da mesma.

Você pode também usar vários caches em diferentes locais assim como as filas. Outras coisas não descritas na arquitetura também podem ser incluídas, como banco de dados (diversos), sistemas de logs, métricas e etc. É obrigatório ter ao menos um sistema de cache e uma fila.

**Exercício 3:**

Implemente a arquitetura desenhada no exercício 2. Considere, caso a arquitetura fique muito complexa, implementar apenas uma parte dela. Desejável ao menos a parte do cache e da fila.

**Requisitos**

* Disponibilizar o projeto em um repositório GIT
* Disponibilizar um README documentando os processos construídos e como executar os projetos

**Critérios de avaliação**

* **Organização do código**: Separação de módulos, view, model, back-end, etc.
* **Flexibilidade do código:** Quanto mais variáveis parametrizáveis, melhor será a avaliação.
* **Assertividade**: A aplicação está fazendo o que é esperado? Se tem algo faltando, o README explica o porquê?
* **Clareza**: O README explica de forma resumida qual é o problema e como pode rodar a aplicação?
* **Legibilidade do código** (incluindo comentários)
* **Segurança**: Existe alguma vulnerabilidade clara?
* **Cobertura de testes** (Não esperamos cobertura completa)
* **Histórico de commits** (estrutura e qualidade)
* **Escolhas técnicas**: A escolha das bibliotecas, banco de dados, arquitetura, etc. É a melhor escolha para a aplicação?