Uma loja de livros com múltiplas camadas

Neste trabalho vamos desenvolver uma loja de livros *online* que usa uma arquitetura separada em duas camadas: *frontend* e *backend*. A camada de *frontend* aceita as requisições dos clientes e faz o processamento inicial dessas requisições. A camada de *backend* é dividida em dois componentes: um servidor de catálogo e um servidor de encomenda. O servidor de catálogo contém o catálogo da loja, isto é, para cada entrada ele armazena o identificador do livro, nome, preço e a quantidade em estoque. O servidor de encomenda mantém uma lista de todas as ordens de compras.

A arquitetura da loja

O servidor de *frontend* implementa três operações:

1. `procura(nome)` – permite que um cliente procure pelo nome de um livro e retorna todas as entradas que casam com a procura efetuada, isto é, retorna o identificador e o nome de cada livro que satisfaz a consulta.
2. `detalhes(id)` – permite que um cliente consulte os detalhes de um livro, isto é, retorna o preço e o número de itens em estoque do identificador consultado.
3. `compra(id)` – permite que um cliente compre um determinado livro a partir do seu identificador.

Conforme exemplificado na Figura 1, as duas primeiras operações disparam consultas ao servidor de catálogo e a última operação dispara uma requisição ao servidor de encomenda.

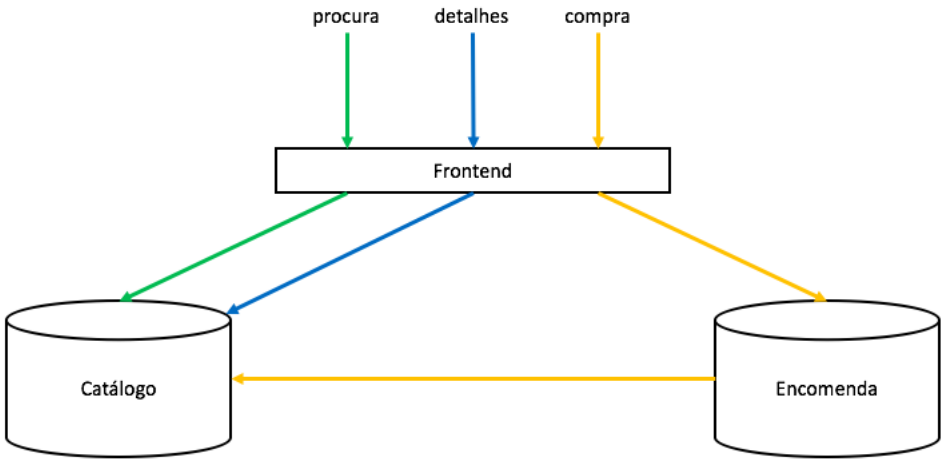

Figura 1: Uma representação da arquitetura da loja

O servidor de catálogo implementa quatro operações:

1. `consultaNome(nome)` – o servidor retorna todas as entradas que casam com o nome que foi consultado.
2. `consultaId(id)` – o servidor retorna os detalhes do identificador que foi consultado.
3. `atualizaPreco(id, preco)` – o servidor atualiza o preço de um determinado livro.
4. `atualizaEstoque(id, qtd)` – o servidor atualiza a nova quantidade em estoque de um determinado livro.

O servidor de encomenda implementa uma operação:

1. `compra(id)` – ao receber uma requisição de compra, o servidor de encomenda consulta o servidor de catálogo para verificar se o livro está disponível no estoque e decrementa em um o número de itens em estoque. A requisição de compra pode falhar se não houver itens em estoque. Assuma que o estoque é atualizado periodicamente e que o catálogo também é atualizado, mesmo que seja para adicionar novos livros.

O primeiro problema a ser resolvido

O dono da loja quer fazer uma promoção para celebrar um ano de funcionamento da loja. Durante a promoção, cada décimo cliente ganha 10% de desconto na sua compra. Como o dono da loja espera que a promoção seja um sucesso, ele deseja que a camada de *frontend* seja replicada em três servidores. Cada cliente pode enviar suas requisições para qualquer uma das réplicas de *frontend*. O *backend* do sistema não será replicado e os servidores de *frontend* devem continuar redirecionando as requisições para os servidores de *backend*.

Para resolver o problema, assuma que os relógios podem não estar sincronizados e mesmo assim queremos determinar uma ordem em que os eventos acontecem (note que os eventos são as requisições efetuadas pelos clientes).

Para implementar a solução, os servidores de *frotend* devem manter relógios lógicos individuais. Você pode usar tanto relógios lógicos de Lamport quanto vetores de relógios lógicos para determinar uma ordem de todas as requisições efetuadas pelos clientes aos servidores de *frontend*. Escolha entre o *multicast* totalmente ordenado e o *multicast* causalmente ordenado para implementar a sua solução. Note que você deve efetuar o *multicast* de uma requisição que chega para um servidor de *frontend* para todos os outros servidores de *frontend*. Note que você deve efetuar o *multicast* de todas as requisições para todos os processos, além de enviar a requisição para o servidor de *frontend* que irá processar a requisição. Use os valores dos relógios lógicos para determinar a ordem das requisições de compra, lembrando que toda décima requisição recebe 10% de desconto no livro que está sendo comprado.

Entrega

- **Todos** os grupos devem usar **Protocol Buffers** [1] na implementação dos seus trabalhos.
- O trabalho deve ser entregue no *Blackboard* até dia 19/09/2017.
- A defesa do trabalho será na aula do dia 20/09/2017.
- Não use código que você não tenha escrito.
- Para resolver o problema, você precisará implementar a solução atual e depois adaptar essa solução para implementar o sistema de descontos.

[1] https://github.com/google/protobuf

- O trabalho tem 20% de peso na nota:
  - Cliente = 0,1;
  - Servidor de *frontend* = 0,1;
  - Servidor de catálogo = 0,2;
  - Servidor de encomenda = 0,2;
  - Sistema de descontos = 0,4;
  - Testes = 0,5;
  - Documentação = 0,5.