Laboratório de Programação Concorrente

Lab10 - Servidor para Loja Online - 24.2

Objetivo

Neste lab vocês irão desenvolver um servidor REST concorrente para uma loja online que gerencie produtos, compras e atualizações de estoque, utilizando estruturas de dados do pacote java.util.concurrent para garantir sincronização, exclusão mútua e alta performance. **Este laboratório pode ser desenvolvido em grupos de 4 alunos.**

Contexto do Problema

Com o aumento das compras online, plataformas de e-commerce precisam lidar com um grande volume de acessos simultâneos. Uma loja online recebe constantemente requisições para consultar produtos, realizar compras, atualizar estoques e registrar novos itens.

Manter a consistência dos dados em um ambiente de alta concorrência é um desafio fundamental. Problemas como condições de corrida, deadlocks e starvation podem comprometer a integridade dos registros e a experiência do usuário. Um sistema de vendas que permite compras concorrentes, por exemplo, pode resultar na venda de um item fora de estoque.

Além disso, o desempenho é essencial: a resposta ao cliente deve ser rápida, mesmo sob alta demanda. Por isso, um servidor concorrente, robusto e eficiente, é essencial para garantir o bom funcionamento da loja online.

Especificação

O servidor REST deve suportar as seguintes funcionalidades obedecendo as rotas e respostas esperadas.

1. Consulta de Produtos:

Rota: GET /products
Descrição: Retorna uma lista de todos os produtos cadastrados.
Resposta (200 OK):

```
[
  {
    "id": "1234",
    "name": "Smartphone",
    "price": 999.90,
    "quantity": 20
  },
  {
    "id": "5678",
    "name": "Fone de Ouvido",
    "price": 199.90,
    "quantity": 50
  }
]
```

Rota: GET /products/{id}
Descrição: Retorna os detalhes de um produto específico.
Resposta (200 OK):

```
{
  "id": "1234",
```

```
  "name": "Smartphone",
  "price": 999.90,
  "quantity": 20
}
```

Resposta (404 Not Found):

```
{
  "message": "Produto não encontrado."
}
```

2. Compra de Produto:

Rota: POST /purchase

Descrição: Realiza a compra de um produto, atualizando o estoque.

Corpo da Requisição:

```
{
  "id": "1234",
  "quantity": 2
}
```

Resposta (200 OK):

```
{
  "message": "Compra realizada com sucesso.",
  "product": {
    "id": "1234",
    "name": "Smartphone",
    "remainingStock": 18
  }
}
```

Resposta (400 Bad Request):

```
{
      "message": "Estoque insuficiente. Quantidade disponível: 1"
}
```

Resposta (404 Not Found):

```
{
  "message": "Produto não encontrado."
}
```

3. Cadastro de Produto:

Rota: POST /products
Descrição: Adiciona um novo produto ao catálogo.

Corpo da Requisição:

```
{
  "id": "9012",
  "name": "Teclado Mecânico",
  "price": 499.99,
  "quantity": 30
}
```

Resposta (201 Created):

```
{
  "message": "Produto cadastrado com sucesso.",
  "product": {
    "id": "9012",
    "name": "Teclado Mecânico"
  }
}
```

Resposta (409 Conflict):

```
{
  "message": "Produto com ID já existente."
}
```

4. Atualização de Estoque:

Rota: PUT /products/{id}/stock
Descrição: Atualiza a quantidade de um produto no estoque.

Corpo da Requisição:

```
{
  "quantity": 50
}
```

Resposta (200 OK):

```
{
  "message": "Estoque atualizado.",
  "remainingStock": 50
}
```

Resposta (404 Not Found):

```
{
  "message": "Produto não encontrado."
}
```

5. Relatório de Vendas:

Rota: GET /sales/report

Descrição: Gera um relatório com os produtos vendidos e as quantidades.

Resposta (200 OK):

```
{
  "totalSales": 25,
  "products": [
    {
      "id": "1234",
      "name": "Smartphone",
      "quantitySold": 10
    },
    {
      "id": "5678",
      "name": "Fone de Ouvido",
      "quantitySold": 15
    }
  ]
}
```

Gerador de Requisições para Teste

O projeto deve incluir um gerador de requisições concorrentes para testar o servidor, simulando múltiplos usuários realizando operações simultâneas.

O gerador deverá criar várias threads, cada uma executando uma sequência aleatória de requisições para as diferentes rotas.

As threads devem simular cenários como:

- Altas taxas de consultas de produtos.
- Compras concorrentes de um mesmo produto.
- Atualizações de estoque em simultâneo.

- Geração de relatórios durante o uso intenso da loja.

Entrega

Você deve criar e manter um repositório privado no GitHub com a sua solução. Este repositório deve conter ser bem documentado e o README deve conter o passo a passo para a execução de sua solução (tanto para o servidor REST quanto para o gerador de requisições). No entanto, a entrega do laboratório deverá ser realizada por meio de submissão online utilizando o script submit-answer.sh, disponibilizado na estrutura de arquivos do próprio laboratório. Uma vez que você tenha concluído sua resposta, seguem as instruções:

1) Crie um arquivo lab10_matr1_matr2_matr3_matr4.tar.gz com a sua solução. Para isso, supondo que o diretório raiz é Lab10, você deve executar:
   tar -cvzf lab10_matr1_matr2_matr3_matr4.tar.gz Lab10

2) Submeta o arquivo lab10_matr1_matr2_matr3_matr4.tar.gz usando o script submit-answer.sh, disponibilizado no mesmo repositório do laboratório:
   bash submit-answer.sh lab10 lab10_matr1_matr2_matr3_matr4.tar.gz

Prazo aberto para submissão

08/abr/25 das 14:00 às 16:00