Disciplina: Sistemas Distribuídos
Professora: Ana Cristina Barreiras Kochem Vendramin

Avaliação (valor 2,5)
Microsserviços, Sistema de Mensageria, REST ou gRPC, SSE, WebHook

Implemente uma aplicação de *e-commerce*.

(valor 0,7) *Frontend*

Os clientes irão interagir com a aplicação para visualizar produtos, inserir, atualizar e remover produtos do carrinho, realizar, excluir e consultar pedidos. O *Frontend* é uma aplicação web que deve ser implementada em uma **linguagem diferente** do *backend*. Essa aplicação consumirá a **API REST** ou **gRPC (valor 0,4)** e receberá notificações via **SSE (*Server Sent Events*) (valor 0,3)**. O SSE é uma tecnologia que permite que um servidor envie atualizações em tempo real para clientes via HTTP, utilizando um único canal de comunicação unidirecional. Diferente do *WebSocket*, que estabelece uma comunicação bidirecional, o SSE é projetado especificamente para enviar dados do servidor para os clientes, tornando-o ideal para aplicações que necessitam de notificações em tempo real ou atualizações contínuas.

(valor 1,6) *Backend*

O *backend* é composto por cinco **microsserviços**, os quais devem ser desenvolvidos de forma independente e se comunicar com outros microsserviços de maneira assíncrona e desacoplada através de um **sistema de mensageria**.

(valor 0,5) Microsserviço Principal (**API REST** ou **gRPC**): responsável por receber as requisições REST do *frontend* para visualizar produtos, inserir, atualizar e remover produtos do carrinho, realizar, excluir e consultar pedidos. Cada novo pedido recebido será publicado (*publisher*) como um evento no tópico **Pedidos_Criados**, cujas mensagens serão consumidas pelos microsserviços Estoque e Pagamento. O microsserviço Principal consumirá eventos (*subscriber*) dos **tópicos Pagamentos_Aprovados**, **Pagamentos_Recusados** e **Pedidos_Enviados** para atualizar o status de cada Pedido. Quando um cliente excluir um pedido ou quando o pagamento de um pedido for recusado, o microsserviço Principal publicará no tópico **Pedidos_Excluídos**.

(valor 0,35) Microsserviço Estoque: responsável por gerenciar o estoque de produtos. Ele consumirá eventos dos tópicos Pedidos_Criados e Pedidos_Excluídos. Quando um pedido for criado ou excluído, esse microsserviço precisará atualizar o estoque. Ele responderá requisições REST ou gRPC do Principal para enviar dados dos produtos em estoque.

(valor 0,2) Microsserviço Pagamento: responsável por gerenciar os pagamentos através da integração com um sistema externo de pagamento via *Webhook*. É necessário definir uma URL (isto é, um *endpoint*) que irá receber as notificações de pagamento aprovado ou recusado. Se o pagamento for aprovado, esse microsserviço publicará o evento no tópico Pagamentos_Aprovados. Caso o pagamento seja recusado, o microsserviço publicará o evento no tópico Pagamentos_Recusados para que o sistema cancele o pedido e atualize o estoque.

(valor 0,15) Microsserviço Entrega: responsável por gerenciar a emissão de notas e entrega dos produtos. Este serviço consome eventos do tópico Pagamentos_Aprovados e publica no tópico Pedidos_Enviados.

(valor 0,4) Microsserviço Notificação: envia notificações via SSE para o *frontend* sempre que houver alteração no status de pedidos (pedido criado, pagamento aprovado, pagamento recusado e pedido enviado). Essas notificações devem incluir o ID e o status do pedido. Este serviço consome eventos de todos os tópicos.

(valor 0,2) Sistema de Pagamento:

Um **sistema de pagamento** deve ser responsável por processar os pagamentos. Nesse sistema será configurado um ***Webhook***. Após a autorização ou recusa de cada pagamento, o sistema de pagamento enviará uma notificação assíncrona (HTTP POST) via *webhook* para o *endpoint* configurado no *backend* do *e-commerce*. O corpo da requisição POST conterá detalhes sobre o evento: ID da transação, status do pagamento (autorizado, recusado, estornado, etc.), valor e dados do comprador.

Webhooks são mais comumente usados para comunicação entre servidores de forma assíncrona, enquanto o REST é um padrão de comunicação cliente-servidor que geralmente envolve interações (operações CRUD) síncronas iniciadas e controladas pelo cliente. *Webhooks* são ideais para notificar um servidor sobre a ocorrência de um evento em outro servidor. Ele é especialmente útil quando uma aplicação não precisa de uma resposta imediata. Por exemplo, serviços de pagamento usam *Webhooks* para notificar sistemas sobre o status de pagamentos. Quando um pagamento é processado com sucesso, o serviço de pagamento notifica automaticamente (*callback*) o sistema de destino via uma requisição HTTP POST, informando o status da transação. O sistema de destino, então, pode atualizar o *status* do pedido com base nos eventos externos recebidos. Isso torna a comunicação mais eficiente e escalável, sem a necessidade de manter conexões abertas ou fazer chamadas repetitivas (*polling*) para saber se um evento ocorreu.

Observações:

- Desenvolva uma interface com recursos de interação apropriados.
- É obrigatória a defesa da aplicação para obter a nota.
- O desenvolvimento do sistema pode ser individual ou em dupla.