



Disciplina: Sistemas Distribuídos

Professora: Ana Cristina Barreiras Kochem Vendramin

Avaliação (valor 2,5)
Microsserviços, Middleware orientado a Mensagens,
REST, SSE, Webhook

Implemente uma aplicação web de Leilão e um sistema de pagamento, conforme ilustrado na Figura 1.

Considere os seguintes processos:

1. RabbitMQ

Middleware orientado a Mensagens (MOM – *Message Oriented Middleware*) responsável por organizar mensagens (eventos) em filas, onde os produtores (*publishers*) as enviam e os consumidores/assinantes (*subscribers*) as recebem. Será utilizado para comunicação assíncrona entre os microsserviços.

2. (0,6) Cliente (*Frontend*)

O **frontend** deve ser implementado em uma **linguagem diferente** daquela utilizada nos demais processos. O cliente interage exclusivamente com a aplicação do Leilão através do **API Gateway**, utilizando **requisições REST** e **conexões SSE** (*Server-Sent Events*).

- (0,2) O cliente envia requisições REST para:
 - **Criar leilões;**
 - Consultar leilões ativos;
 - Efetuar lances;
 - **Registrar interesse em notificações;**
 - **Cancelar interesse em notificações.**
- (0,2) O cliente recebe notificações via **SSE**, incluindo:
 - Novos lances válidos em leilões de interesse;
 - Encerramento de leilões e anúncio de vencedor;
 - **Geração de link de pagamento;**
 - **Resultado do pagamento (aprovado ou recusado).**
- (0,2) O cliente vencedor do leilão recebe via **SSE** o link de pagamento gerado pelo sistema e pode acessá-lo para concluir a compra.

3. (0,9) API Gateway (sub)

Responsável por atuar como ponto único de entrada para o *frontend*, expondo os *endpoints* REST e enviando requisições REST aos microsserviços internos. O API Gateway apenas consome eventos do RabbitMQ.

(0,5) Expõe a API REST para o cliente:

- (0,1) **Criar um leilão.** O API *gateway* receberá os dados do leilão (nome do produto, descrição, valor inicial, data e hora de início e fim do leilão) e encaminhará ao MS Leilão via REST;
- (0,1) **Consultar leilões ativos.** O API *gateway* consultará o MS Leilão (REST) para retornar os dados (nome do produto, descrição, valor inicial ou último lance, data e hora de início e fim do leilão) dos leilões ativos aos clientes;
- (0,1) **Efetuar um lance.** O API *gateway* receberá do cliente os dados do lance (ID do leilão, ID do usuário, valor do lance) e enviará esses dados ao MS Lance via REST;
- (0,1) **Registrar interesse em receber notificações (novos lances e vencedor) sobre um leilão.** O API *gateway* deve gerenciar os interesses ativos e os eventos gerados, a fim de enviar notificações apenas aos clientes interessados nesses eventos.
- (0,1) **Cancelar interesse em receber notificações (novos lances e vencedor) sobre um leilão.** O API *gateway* não deve mais enviar notificações aos clientes que cancelaram o registro de interesse em um leilão.

(0,4) **Responsável por manter conexões SSE com os clientes.** Com base nos eventos consumidos *lance_validado*, *lance_invalidado*, *leilao_vencedor*, *link_pagamento*, *status_pagamento*, o API *gateway* enviará as seguintes notificações personalizadas (através de canais diferenciados pelo clientID):

- Novo lance válido (aos clientes registrados);
- Lance inválido;
- Vencedor do leilão (aos clientes registrados);
- Link de pagamento;
- Pagamento aprovado ou Pagamento recusado.

4. (0,2) MS Leilão (pub)

Responsável pela criação e gerenciamento dos leilões.

- (0,2) **Recebe requisições REST do gateway para:**
 - Criar leilões.
 - Consultar leilões.
- Controla o ciclo de vida dos leilões:
 - Ao atingir a data/hora de início → publica *leilao_iniciado*.
 - Ao atingir a data/hora de término → publica *leilao_finalizado*.

5. (0,1) MS Lance (pub/sub)

Responsável por gerenciar os lances realizados nos leilões.

- Ele consome os eventos *leilao_iniciado* e *leilao_finalizado*.
- Recebe requisições REST do gateway contendo um lance (ID do leilão; ID do usuário, valor do lance). Somente aceita o lance se o leilão estiver ativo e se o lance for maior que o último lance registrado.
 - Se o lance for válido, o MS Lance publica *lance_validado*.
 - (0,1) **Caso contrário, ele publicará o evento no *lance_invalidado*.**
- Quando o evento *leilao_finalizado* é recebido, o MS Lance determina o vencedor e publica *leilao_vencedor*, informando o ID do leilão, o ID do vencedor do leilão e o valor negociado.

6. (0,5) MS Pagamento (pub/sub)

- (0,1) **Consome os eventos *leilao_vencedor* (ID do leilão, ID do vencedor, valor).**
- (0,2) **Para cada evento consumido, ele fará uma requisição REST ao sistema externo de pagamentos enviando os dados do pagamento (valor, moeda, informações do cliente) e, então, receberá um link de pagamento que será publicado em *link_pagamento*.**
- (0,2) **Ele define um *endpoint* que recebe notificações assíncronas do sistema externo indicando o status da transação (aprovada ou recusada). Com base nos eventos externos recebidos, ele publica eventos *status_pagamento*, para que o API Gateway notifique o cliente via SSE.**

O **MS Notificação** será removido. Sua função foi incorporada ao API Gateway, que agora gerencia as notificações via SSE.

7. (0,2) Sistema de pagamento externo

O sistema externo recebe requisições REST do MS Pagamento para iniciar a transação e retorna um link de pagamento. Após o processamento do pagamento, o sistema de pagamento externo envia uma notificação assíncrona via webhook (HTTP POST) ao *endpoint* configurado no MS Pagamento. Essa notificação inclui informações sobre o evento: ID da transação, status do pagamento (aprovado ou recusado), valor e dados do comprador.

Observações:

- Não há necessidade de assinar as mensagens digitalmente;
- 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.

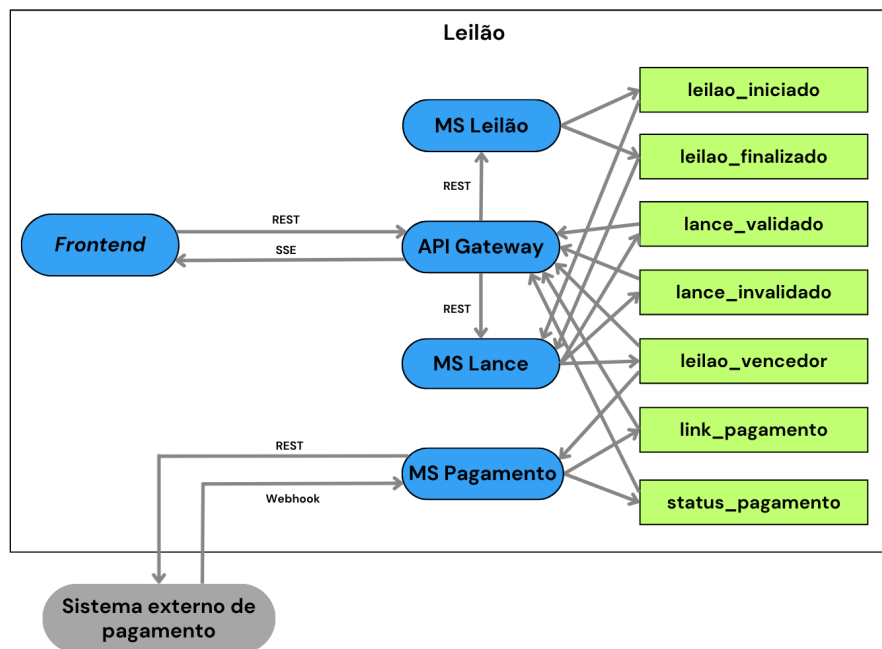


Figura 1 – Arquitetura.

Conceitos:

SSE (Server-Sent Events) é uma tecnologia que permite ao servidor enviar atualizações em tempo real para os clientes por meio de uma conexão HTTP, utilizando um canal de comunicação unidirecional. Diferentemente do **WebSocket**, que estabelece uma comunicação bidirecional, o SSE é projetado especificamente para o envio de dados do servidor para os clientes. Isso o torna especialmente adequado para aplicações que demandam notificações em tempo real ou atualizações contínuas, como alertas de promoções.

Webhooks (ou *APIs reversas* ou *APIs de push*) são mais comumente usados para comunicação entre servidores de forma assíncrona, enquanto REST é um padrão de comunicação cliente-servidor que geralmente envolve interações síncronas, iniciadas e controladas pelo cliente. *Webhooks* são ideais para notificar um servidor sobre a ocorrência de um evento em outro servidor. Isso torna a comunicação mais eficiente e escalável, sem a necessidade de manter conexões abertas ou realizar chamadas repetitivas (*polling*) para verificar se um evento ocorreu.