

Desenvolvimento Web II

Aula 07 - Uso de Mensageria e Filas de Mensagens

Prof. Fabricio Bizotto

Instituto Federal Catarinense
fabricio.bizotto@ifc.edu.br

Ciência da Computação
3 de abril de 2024

- 1 Chamadas Síncronas e Assíncronas
- 2 Mensageria
- 3 Exemplo de Uso
- 4 Mensagem
- 5 Quando Utilizar?
- 6 Message Broker
- 7 Nomenclatura
- 8 Material Complementar
- 9 Simulação
- 10 Experimentos

Exemplo em JavaScript - Síncrono

```
const response = await fetch('https://api.github.com/users/fabriziobizotto');  
const data = await response.json();  
console.log(data);
```

Exemplo em JavaScript - Assíncrono

```
fetch('https://api.github.com/users/fabriziobizotto')  
.then(response => response.json())  
.then(data => console.log(data));
```

"Mensageria é um conceito que define que sistemas distribuídos, possam se comunicar por meio de troca de mensagens (evento), sendo estas mensagens "gerenciadas" por um Message Broker (servidor/módulo de mensagens)."

- A mensageria é um padrão de comunicação **assíncrona** entre aplicações.
- A comunicação assíncrona é feita por meio de mensagens que são enviadas para uma **fila de mensagens**.
- A fila de mensagens é gerenciada por um *broker*, que é responsável por garantir que as mensagens sejam entregues e processadas.

Exemplo de Uso

Imagine que na organização na qual você trabalha, surge a necessidade de realizar uma integração com algum sistema parceiro que agrega valor ao seu negócio. Parece ser algo simples e você logo pensa:

Bom só preciso me preocupar, em desenvolver uma comunicação com a Web API do sistema que desejo integrar e está tudo certo!

Exemplo de Uso

Imagine que na organização na qual você trabalha, surge a necessidade de realizar uma integração com algum sistema parceiro que agrega valor ao seu negócio. Parece ser algo simples e você logo pensa:

Bom só preciso me preocupar, em desenvolver uma comunicação com a Web API do sistema que desejo integrar e está tudo certo!

Problema

De certa forma, é basicamente isso! Porém, vamos analisar alguns pontos importantes no cenário apresentado:

- E se o sistema parceiro estiver fora do ar?
- E se o sistema parceiro estiver lento?
- E se o sistema parceiro estiver sobrecarregado?
- E se o sistema parceiro estiver em manutenção?
- E se o sistema parceiro estiver com problemas de rede?

Exemplo de Uso

Imagine que na organização na qual você trabalha, surge a necessidade de realizar uma integração com algum sistema parceiro que agrega valor ao seu negócio. Parece ser algo simples e você logo pensa:

Bom só preciso me preocupar, em desenvolver uma comunicação com a Web API do sistema que desejo integrar e está tudo certo!

Possíveis Soluções

- Criar vários ifs para tratar cada um dos problemas apresentados anteriormente.
- Criar um mecanismo de *retry* para tentar novamente a comunicação com o sistema parceiro.
- Criar um mecanismo de *cache* para armazenar os dados que foram enviados para o sistema parceiro.
- Registrar em um banco de dados os dados que o registro não foi integrado e posteriormente tentar novamente.
- Utilizar um mecanismo de mensageria para realizar a comunicação com o sistema parceiro.

Definição

- São estruturas com informações trocadas entre sistemas.
- Pode representar um evento, uma notificação, um comando, etc.
- Deveriam ser pequenas e conter apenas as informações necessárias.
- Deve-se evitar o uso de dados sensíveis.
- Possui um formato bem definido com um cabeçalho (*header*) e um corpo (*payload*).

Exemplos

- **Pedido de Compra:** é um objeto que contém os dados de um pedido de compra.
- **Notificação de Pagamento:** é um objeto que contém os dados de um pagamento realizado.
- **Notificação de Entrega:** é um objeto que contém os dados de uma entrega realizada.
- **E-mail de Confirmação:** é um objeto que contém os dados de um e-mail que será enviado.

- **Comunicação Assíncrona:** o sistema não precisa esperar a resposta do sistema parceiro.
- **Sistemas Distribuídos/Desacoplados:** caso o sistema parceiro esteja fora do ar, o sistema não deve parar de funcionar.
- **Escalabilidade:** evita que o sistema fique sobrecarregado.
- **Resiliência:** em caso de erro, o sistema pode tentar novamente mais tarde.
- **Background:** avisar o usuário que o pedido foi realizado, mas não precisa esperar a resposta do sistema parceiro.

Definição

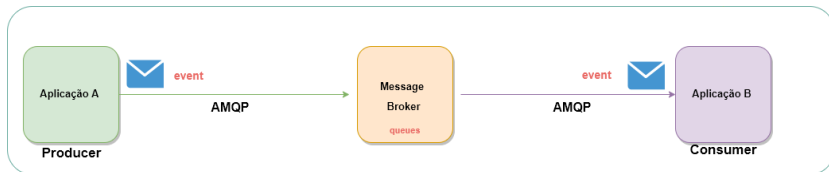
Um Message Broker nada mais é que um **servidor de mensagens**, responsável por garantir que a mensagem seja enfileirada e armazenada em disco (opcional), garantindo que ela fique lá enquanto necessário até que alguém consuma a mensagem.

Ferramentas

- RabbitMQ
- ActiveMQ
- Amazon SQS
- Azure Service Bus
- Google Cloud Pub/Sub
- Apache Kafka
- Redis

Básico

- **Producer/Publisher:** é o sistema que envia a mensagem para o *message broker*.
- **Event:** é o evento que será enviado para o *message broker*. Pode ser um pedido de compra, uma notificação de pagamento, etc.
- **Queue:** é uma fila que recebe as mensagens geradas por um producer. As mensagens ficarão dentro da fila até que alguma aplicação consumidora (consumer) retire a mensagem da fila.
- **Consumer:** é o sistema que recebe a mensagem do *message broker*.

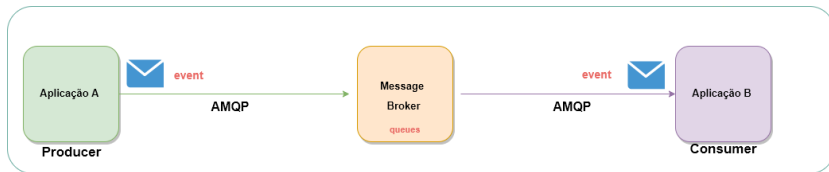





AMQP - *Advanced Message Queuing Protocol*

Protocolo de rede que permite a comunicação com um middleware (*message broker*) para troca de mensagens assíncronas.

Producer > Exchange > Binding > Queue > Consumer


- **Exchange:** é a porta de entrada das mensagens no *message broker*.
- **Binding:** é o vínculo entre uma fila de mensagens e uma exchange. Regras de roteamento.



-  **Playlist - Mensageria.**
Canal **Gabriel Faraday**.
-  **Apache Kafka e Spring Boot.**
Livro **Casa do Código**.
-  **Aprendendo sobre mensageria.**
Canal **Daniele Leão**.

Objetivo

Simular o envio de uma mensagem para uma fila de mensagens utilizando o RabbitMQ.

 RabbitMQ Simulator Tool

Experimento 1

Acesse o repositório RabbitMQ Tutorials, escolha uma linguagem de programação e realize os experimentos propostos.

Observações

Pré-requisitos: instalar o RabbitMQ.

Experimento 2

Acesse os detalhes para este experimento no repositório através do link abaixo: 