Arquitetura Orientada a Eventos para Cadastro e Validação de Dados Utilizando RabbitMQ
Sistema Bancário

Magdiel Prestes Rodrigues[1];Gian Lucca D.T.A.V. e Novais[2];Wesley Dos Reis Bezerra[3]

[1] Estudante de Graduação em Sistemas de Informação, IFC - Campus Rio do Sul. E-mail: magdielprestes@gmail.com

[2] Estudante de Graduação em Sistemas de Informação, IFC - Campus Rio do Sul. E-mail: gianluccavalle13@gmail.com

[3] Orientador, Professor EBTT, IFC - Campus Rio do Sul. E-mail: wesley.bezerra@ifc.edu.br

RESUMO

Este trabalho propõe uma arquitetura orientada a eventos para cadastro e validação de dados, utilizando o RabbitMQ como middleware de comunicação entre componentes. O sistema inicia com um formulário web que envia dados em JSON para uma fila de mensagens (Fila 1). Um consumidor de serviço valida as regras de negócio (CPF, e-mail, telefone) e encaminha os dados válidos para uma segunda fila (Fila 2). Um consumidor de banco processa essas mensagens, persistindo os dados em um banco MySQL e enviando informações para uma terceira fila (Fila 3). A metodologia incluiu a modelagem do fluxo de dados, implementação com Python (Flask e Pika), e containerização com Docker para garantir portabilidade. Os resultados demonstraram a eficácia do RabbitMQ na orquestração assíncrona de processos, com tratamento de erros robustos via filas dedicadas. Conclui-se que a arquitetura melhora a escalabilidade e resiliência do sistema, permitindo a integração modular de novos serviços.

Palavras-chave: Validação de dados; Sistemas distribuídos; Comunicação assíncrona; Integração de serviços.

INTRODUÇÃO

A complexidade crescente de sistemas distribuídos demanda arquiteturas que garantam desacoplamento e tolerância a falhas. Neste contexto, este trabalho explora o uso de filas de mensagens (RabbitMQ) para coordenar o fluxo de cadastro de usuários em um sistema bancário simulado. O processo envolve três estágios principais: envio de dados via interface web, validação de regras de negócio, e persistência em banco de dados. A solução busca responder à problemática de como garantir a consistência e rastreabilidade em operações assíncronas, utilizando padrões de message brokering. Justifica-se pela necessidade de sistemas resilientes em cenários com alta demanda concorrente, onde falhas pontuais não devem comprometer o fluxo global.

PROCEDIMENTOS METODOLÓGICOS

O desenvolvimento do sistema foi estruturado em quatro etapas principais, visando garantir uma implementação robusta e organizada. A primeira etapa consistiu na modelagem arquitetural, onde foi definido o fluxo de dados entre o formulário web, as filas do RabbitMQ (Fila 1, Fila 2 e Fila 3) e o banco de dados. Essa modelagem permitiu visualizar claramente como as informações seriam processadas, desde a captação dos dados até a persistência final.

ISSN 2318-2342

Na etapa de implementação, o frontend foi desenvolvido utilizando HTML para o formulário de cadastro, com validações básicas no lado do cliente, enquanto o backend foi construído com Flask para receber e enviar os dados em formato JSON. Para o processamento das mensagens, foram criados dois consumidores em Python utilizando a biblioteca Pika: o Consumer Service, responsável pela validação das regras de negócio (como CPF, e-mail e telefone), e o Database Consumer, encarregado de persistir os dados validados no banco de dados MySQL. A estrutura do banco de dados foi projetada com tabelas para usuários e transações, garantindo a integridade e rastreabilidade das informações.

A integração dos componentes foi realizada com a configuração do RabbitMQ, onde foram criadas filas duráveis para garantir que as mensagens não fossem perdidas em caso de falhas. Políticas de retenção foram implementadas para lidar com erros temporários, assegurando a resiliência do sistema. Por fim, a conteinerização foi feita utilizando Docker Compose, que permitiu orquestrar os serviços de forma isolada e replicável, incluindo o MySQL, RabbitMQ e os serviços Python. Essa abordagem facilitou a implantação e o teste do sistema em diferentes ambientes.

RESULTADOS E DISCUSSÃO

O sistema demonstrou resultados consistentes e eficientes no processamento de dados. O fluxo de trabalho, ilustrado na Figura 1, seguiu a sequência: [Formulário] → (Fila 1) → [Validação] → (Fila 2) → [Persistência] → (Fila 3). Durante a fase de validação, o Consumer Service aplicou regras rigorosas, como a validação de CPF utilizando o algoritmo de dígitos verificadores e a verificação do formato de e-mail com expressões regulares (regex). Essas validações resultaram na rejeição de 15% das entradas de teste, garantindo que apenas dados corretos fossem encaminhados para a próxima etapa.

A persistência dos dados foi realizada com atomicidade, ou seja, as transações só foram confirmadas após a validação completa e a inserção no banco de dados, evitando inconsistências. Em caso de falhas durante a persistência, as mensagens foram redirecionadas para a Fila 3, funcionando como um mecanismo de dead letter para análise posterior. Essa abordagem garantiu a resiliência do sistema, permitindo a recuperação de erros sem comprometer o fluxo principal.

As métricas de desempenho, apresentadas na Tabela 1, mostraram que o tempo médio de processamento na fase de validação foi de 120ms, com uma taxa de sucesso de 85%. Já na etapa de persistência, a latência média foi de 250ms, com uma taxa de sucesso de 98%. Esses resultados comprovam a eficácia da arquitetura proposta, que conseguiu balancear desempenho e confiabilidade.

CONSIDERAÇÕES FINAIS

A arquitetura orientada a eventos proposta mostrou-se altamente eficaz no gerenciamento de fluxos assíncronos, destacando-se pela facilidade de depuração e escalabilidade. O uso de filas duráveis no RabbitMQ garantiu a resiliência do sistema, enquanto a conteinerização com Docker Compose simplificou a implantação e a replicação do ambiente.

Entre os desafios enfrentados, destaca-se a necessidade de otimizar o consumo paralelo de mensagens para melhorar o desempenho em cenários de alta demanda. Além

ISSN 2318-2342

disso, a implementação de dead letter exchanges poderia automatizar o tratamento de erros, reduzindo a necessidade de intervenção manual.

Como trabalho futuro, sugere-se a adição de um serviço de notificação (e-mail/SMS) que consuma mensagens da Fila 3, proporcionando feedback imediato aos usuários sobre o status de seus cadastros. Essa funcionalidade aumentaria a usabilidade do sistema e completaria o ciclo de comunicação assíncrona.

REFERÊNCIAS

COCHRAN-SMITH, M.; LYTLE, S. L. Relationship of knowledge and practice: Teacher learning in the communities. **Review of Research in Education**, n. 24, p. 249-305, 1999.

DEMO, P. **Educar pela pesquisa**. Campinas: Autores Associados, 2000.

ISSN 2318-2342