Arquitetura de Microsserviços com Spring Cloud

|  |  |
| --- | --- |
| **Élder Rezende**  eldertec@gmail.com | **Fredrico Hernany**  fredericoh@gmail.com |

|  |  |
| --- | --- |
| Resumo | //TODO  **Palavras-Chave:** TODO |

|  |  |
| --- | --- |
| Abstract | //TODO  **Keywords:** TODO |

1 Introdução

Pela necessidade de disponibilidade, na busca de maior escalonamento e o desejo de incluir novas linguagens e tecnologias, grandes aplicações como o Uber, tem decomposto seus sistemas monolíticos em microsserviços e encarado o árduo desafio de gerí-los.

2 Modelo de Negócio

Nesse tópico descreveremos o que são microsserviços, onde e quando devem ser aplicados, um breve comparativo entre a arquitetura monolítica e a arquitetura de microsserviços, as vantagens e desafios da arquitetura de microsserviços.

3 Tecnologias

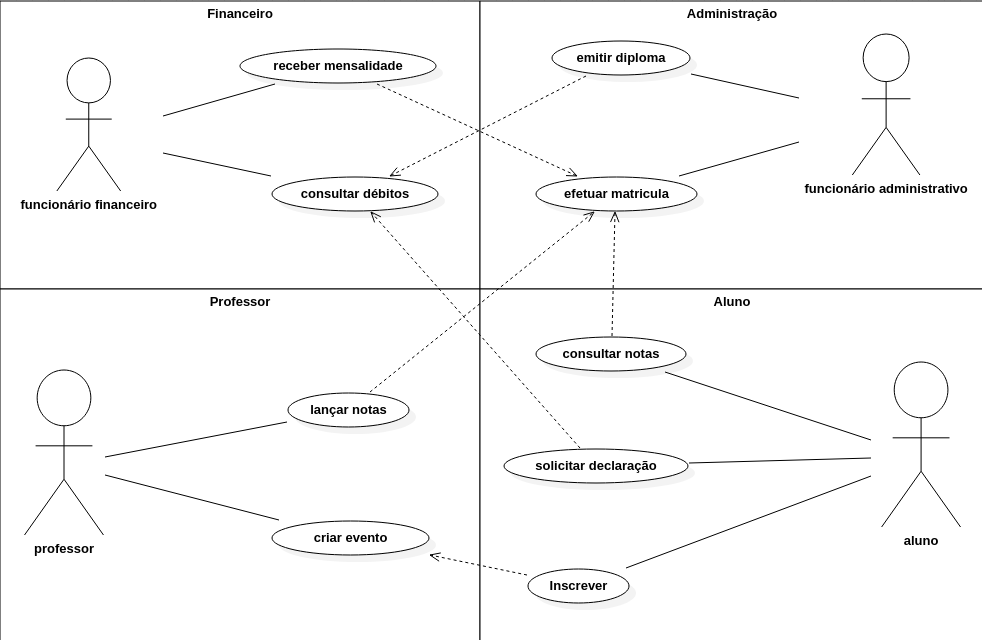
Serão utilizadas as ferramentas disponibilizadas pelo Spring Cloud, para a configuração do projeto, registro e descoberta de serviços, chamadas entre serviços e API Gateway.

4 Estudo de Caso

4.1 Objetivo Geral

Nosso estudo de caso foi baseado na decomposição de microsserviços a partir da abstração de domínios (DDD) em um cenário educacional. Com isso originando cinco domínios, são eles: Administração, Financeiro, Professor, Evento e Aluno. E suas funcionalidades estão sendo descritas no digrama de caso de uso da Imagem 1.

Imagem 1 - Diagrama de Caso de Uso



Fonte – Própria, estudo de caso deste artigo

4.2 Detalhamento dos Microserviços

Os domínios foram implementados de forma independente e cada serviço possui seu próprio banco de dados. Sendo 4 deles com bancos relacionais, onde 2 utilizam o MySql e 2 o Postgresql, e o Evento com banco não relacional utilizando o MongoDB.

Todos serviços foram implemantados utilizando o ecossistema Spring na versão 2.3.1 e a linguagem Java na versão 11. O Maven foi a ferramenta escolhida pra build e gerenciamento das dependências.

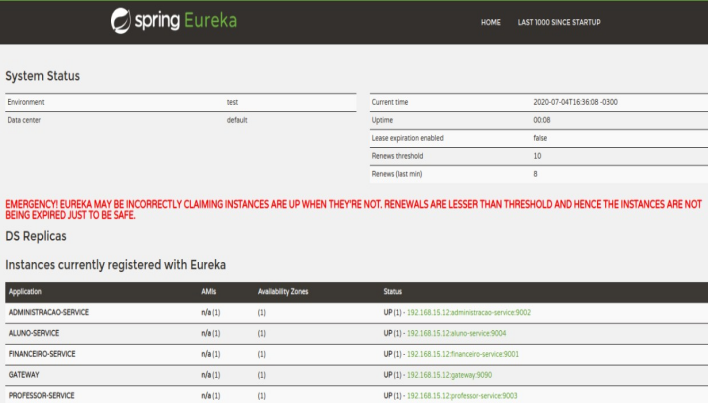
Foi implementado o servidor Eureka, que tem a função de descoberta e registro dos serviços. A API Gateway, sendo o proxy da aplicação ficando responsável pelo gerenciamento das rotas. Implementamos a documentação centralizada dos serviços utilizando o Swagger.

Para a comunicação entre os microserviços foi utilizado o padrão REST que utiliza o protocolo HTTP para comunicação síncrona.

4.3 Resultados

Como resultados temos os serviços registrados e gerenciados pelo Eureka Server sendo exibidos na Imagem 2.

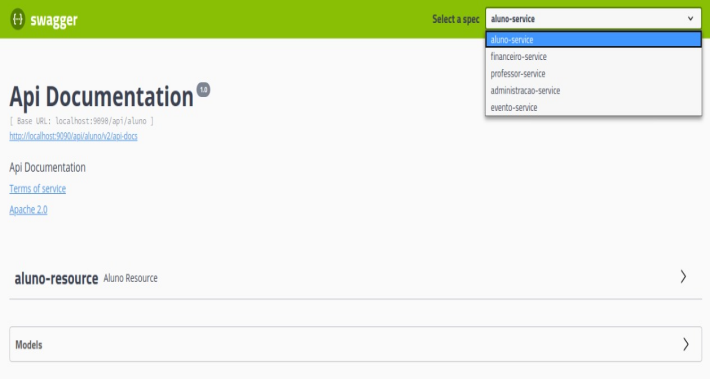
Imagem 2 - Eureka Server



Fonte – Própria, estudo de caso deste artigo

Na Imagem 3 temos as requisições filtradas pelo proxy e centralizadas na porta 9090 tornando acessível toda a documentação da API em um único endpoint.

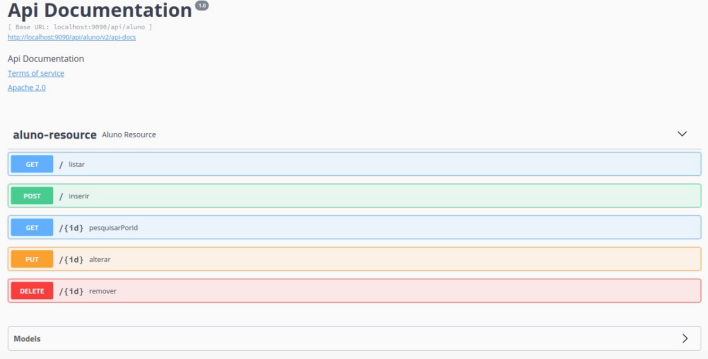
Imagem 3 - Documentação com Swagger



Fonte – Própria, estudo de caso deste artigo

A Imagem 4 exibe, pelo Swagger, a implementação do CRUD genérico do Aluno e seus endpoits estão documentados e disponíveis para testes.

Imagem 4 - Aluno Resource



Fonte – Própria, estudo de caso deste artigo

5 Considerações Finais

Serão apresentadas as dificuldades enfrentadas, as conclusões obtidas e as sugestões para futuras continuações do artigo.

Referências

|  |  |
| --- | --- |
| **[1]** | Fowler, Susan J. Microsserviços prontos para a produção. São Paulo, Novatec, 2017. |
| **[2]** | Newman, Sam. Migrando sistemas monolíticos para microsserviços. São Paulo, Novatec,  2020. |
| **[3]** | Martin, Robert. Arquitetura Limpa: O Guia do Artesão para Estrutura e Design de Software. Rio de Janeiro, Alta Books, 2018. |
| **[4]** | Documentação do Spring Cloud. Disponível em: <https://spring.io/projects/spring-cloud#overview.> Acesso em 17/05/2020. |