MICROMORPH: INTEGRADOR PARA DADOS HETEROGÊNEOS

Renan Mathias Gomes da Rosa

RESUMO

A integração de dados heterogêneos entre ambientes é uma ocorrência comum entre sistemas web modernos, geralmente exigindo a necessidade de construir *middlewares* para realizar a integração entre esses distintos sistemas. Ao decorrer deste artigo será construído o micro serviço Micromorph que poderá ser utilizado como substituto para essas integrações específicas. Esse serviço, será projetado de forma schemaless, permitindo a inserção de qualquer payload, o gerenciamento de dados utiliza metadados, os quais podem ser inseridos pelo cliente ou gerados pela aplicação durante a execução. O Micromorph disponibiliza diferentes métodos de integração, incluindo upload manual, integração via *https* e mensagens assíncronas e visando facilitar a utilização será desenvolvido uma documentação compreensível e a aplicação será disponibilizada como imagem docker. Os testes realizados se baseiam em diferentes possibilidades de arquitetura, com o objetivo de demonstrar a utilidade e elasticidade do serviço. O artigo a seguir apresentará a implementação desse serviço, como utilizar e futuras melhorias que poderiam ser realizadas ao processo.

Palavras-chave: Middleware, Micro-serviço, Banco de dados não relacional, Metadados.

ABSTRACT

The integration of heterogeneous data between environments is a common occurrence among modern web systems, usually requiring the need to build middleware to perform the integration between these different systems. Throughout this article, the Micromorph microservice will be built, which can be used as a replacement for these specific integrations. This service will be designed in a schemaless way, allowing the insertion of any payload. The data management will be done using metadata, which can be inserted by the client or generated by the application during execution. Micromorph provides different integration methods, including manual upload, integration using https and asynchronous messages. In order to facilitate use, understandable documentation will be developed and the application will be made available as a Docker image. The tests performed are based on different architectural possibilities, with the aim of demonstrating the usefulness and elasticity of the service. The following article will present the implementation of this service, how to use it and future improvements that could be made to the process.

Keywords: Middleware, Micro service, Non Relational databases, Metadata.

1 INTRODUÇÃO

Atualmente a expansão da área de ciência de dados, gera o desafio em lidar com uma grande quantidade de informações heterogêneas provenientes de fontes diversas. Nesse contexto surge a necessidade de persistir e homogeneizar esses dados. Este artigo propõe o desenvolvimento do Micromorph, um serviço web genérico capaz de processar esses dados e armazenar de forma centralizada, com o foco em diminuir o tempo de elaboração de serviços middleware. Será construído uma base de dados não relacional baseada nos metadados fornecidos pelos diferentes tipos de dados enviados. A existência deste projeto surge na necessidade de simplificar e otimizar o tratamento de dados heterogêneos, visto que integrações de software frequentemente gera a criação de serviços singulares para a realização da tarefa de higienização. Sendo assim o Micromorph surge como uma maneira alternativa, eliminando a necessidade de serviços middleware especializados.

Como tema tem-se a criação de um serviço web para o armazenamento e disponibilização eficiente de dados heterogêneos para a área de ciência de dados. Este serviço recebe arquivos de variados formatos por meio de entradas genéricas e os organiza em uma base centralizada, construída com base em metadados, com o propósito de facilitar sua utilização na ciência de dados. Se dá como problemas de pesquisa a possibilidade construir um serviço genérico de fácil acoplagem em arquiteturas de micro serviço para homogeneizar dados que vêm de diferentes do produto? Se o cliente poderá utilizar o Micromorph como substituto ao desenvolvimento de um serviço middleware singular para realizar uma integração? Há o que se refere a objetivos seria, construir um serviço web de fácil acoplagem para arquiteturas de micro serviços cloud, que recebe arquivos nos formatos (csv, json) e suporta diferentes tipos de entradas (filas/brokers, api e upload manual) permitindo disponibilizar esses arquivos para utilização na área da ciência de dados. Esse serviço deve ser disponibilizado em forma de imagem (docker) para ser facilmente acoplado em arquiteturas cloud. Os seguintes objetivos específicos foram planejados para o projeto:

- Pesquisar e compreender a implementação de data lakes;
- Modelar a arquitetura utilizando UML;
- Construir um serviço web que gere sua própria base de dados utilizando os metadados;
- Testar e validar o software;
- Criar casos de uso;
- Documentar a utilização do sistema;
- Disponibilizar como uma imagem docker, para a sua distribuição.

2 REFERENCIAL TEÓRICO

A implementação e utilização de um serviço como o Micromorph se assemelha, a elaboração de um data lake, foi necessário buscar literatura que remete a implementação de data lake, construção de micro serviços web e banco de dados não relacional.

Se vê a necessidade da construção do projeto em formato de micro serviço devido à ideia de se acoplar em arquiteturas já utilizadas, o conceito de trabalhar com micro serviços surge com a possibilidade de uma melhor escalabilidade, disponibilidade e também visto que ele necessita ser genérico, não seria viável a

elaboração dele em uma estrutura monolítica, como ele será somente um módulo em uma estrutura de cliente, os desafios de arquiteturas de micro serviço, como o gerenciamento de código fonte e a complexidade do todo, não será aplicável.(Souza Emmanuel, 2016)

A utilização de bancos de dados não relacionais é um paradigma necessário para a construção de um sistema desse tipo. Bancos de dados relacionais utilizam o modelo de tabela e relacionamento, o qual são estruturas de dados definidas, diferentemente de bancos não relacionais, que podem armazenar dados em diferentes formatos, como chave/valor e documentos.(Conceição Marcus, 2023)
Para a implantação de um gerenciador de dados heterogêneos, é de grande importância a criação de um catálogo, que fornece descrições, necessária para a utilização e gerenciamento dos dados, elas serão os metadados, as quais incluem informações como fornecedor da fonte de dados, propriedade das informações, modo de aquisição, detalhes de acesso e estrutura de dados, dependendo da forma que o dado é fornecido a busca desses metadados mudará, caso nao informado, o próprio Micromorph deverá criar. Saneamento de dados é outro ponto importante, visando a melhoria da qualidade de dados. São identificados quatro problemas: outliers(valores discrepantes, dados com uma extrema diferença um do outro), duplicatas (repetição exata da mesma informação), violação de regras e violação de padrões. A violação de regras e padrões, deverão ser implementadas de acordo com os metadados, pois fontes e tipos diferentes de dados terão regras diferentes aplicadas em cima.(IEEE 35th international conference on data engineering workshops, 2019)

3 FUNDAMENTOS

O software construído neste artigo é um micro serviço middleware, isso consiste em um software utilizado para ser a ponte de comunicação entre diferentes aplicações, ele fornece um padrão de API(*application programming interface*) para realizar o gerenciamento dos dados, esse formato abstrai os componentes internos, tornando invisíveis para os seus usuários, mas retorna um resultado determinado, facilitando o desenvolvimento de outras aplicações.(AWS, 2023) Dessa forma permitindo ao usuário que está utilizando a API, não se preocupar com detalhes e empenhar-se na sua regra de negócio.

Uma arquitetura de *microservices*(micro serviços) consiste em um único aplicativo ser composto por um ou mais serviços, que fornecem recursos limitados, essa arquitetura permite o escalonamento individual de recursos, caso um serviço está sendo mais utilizado que o outro e também permite que cada serviço seja construído usando tecnologias diferentes, pois cada serviço precisa ser contido nele mesmo.(Souza Emmanuel, 2016) O Micromorph é construído nesse formato de micro serviço middleware para se tornar parte de um ecossistema do usuário.

Docker é uma plataforma aberta que permite separar um software de sua infraestrutura, ele empacota e executa o aplicativo em um ambiente isolado chamado de container, um container contém tudo que é necessário para a execução, esse container podem ser compartilhados, permitindo que outros usuários utilizem o mesmo container que funcionará da mesma maneira.(Docker documentation, 2024) A utilização dessa plataforma se torna ideal para a entrega da solução, devido o isolamento do software com infraestrutura, possibilitando a implementação dele em qualquer infraestrutura cloud.

Outro conceito necessário para compreensão são os metadados, eles representam informações sobre os dados, ”os dados dos dados”. Os metadados facilitam a busca, organização e aplicação de dados, podendo ser utilizado tanto em softwares quanto em livros ou produtos físicos. Eles abrangem diversos aspectos dos dados, como sua fonte, localização, formato e qualidade, podendo ser organizados e referenciados de várias maneiras.(Volle Adam, 2024) Eles serão utilizados para a gestão do banco de dados.

4 ESCOLHA DE TECNOLOGIAS.

O primeiro passo para o desenvolvimento da aplicação foi selecionar as tecnologias que seriam utilizadas, isso inclui tanto a linguagem de programação, framework, agentes terceiros e ferramentas necessárias para testar. Atualmente existem muitas opções que possibilitam o desenvolvimento de softwares com bancos de dados não relacionais e talvez a principal escolha da aplicação seria qual banco de dados utilizaria.

O Núcleo da aplicação, será utilizado Java 17(LTS) com a utilização do *framework* springboot. O *springboot* auxilia a construção de uma aplicação web autônoma, isso quer dizer uma aplicação que funciona sozinha sem a necessidade de servidores web externos, como o *Apache Tomcat*. Isso me permite abstrair as necessidades de configuração de um projeto java, focando assim na construção do programa.

Para o Micromorph poder se tornar dinâmico, e possibilitar utilizá-lo em diferentes contextos e arquiteturas, para diferentes motivos, acaba impossibilitando a utilização de um banco de dados relacional, outro importante ponto na escolha do sistema foi a licença, sendo priorizado softwares *open source*, ou com licença de uso aberto, então inicialmente seria escolhido o banco de dados MongoDB.
MongoDB é orientado a documentos, esse padrão define que os dados e documentos são autocontidos e auto descritivos isso implica que o documento em si já define como ele deve ser apresentado e qual é o significado dos dados armazenados.(Coimbra Vinicius , 2018) Após pesquisas acabou se tornando viável a utilização de outro banco de dados, o Elasticsearch, diferentemente do MongoDB, não é orientado a documentos, e sim um *search engine*, baseado no Apache Lucene. Ambos os bancos permitem guardar dados no formato de JSON, o qual é desejado para a aplicação, porém como o Elasticsearch consegue retornar grandes quantidades de dados em pouco tempo, acaba se tornando uma solução mais interessante para o projeto.

A aplicação disponibiliza uma interface gráfica para prover a documentação e o envio manual de documentos. Essa interface foi construída usando a biblioteca *bootstrap* com javascript, essa biblioteca facilita o desenvolvimento de interfaces web responsivas, a documentação da *API* é disponível também através de um *postman collection* no código fonte da aplicação.

Outras ferramentas foram utilizadas para a construção dos exemplos, como o Drawlo, para a construção do material gráfico, *Mockaroo* para gerar dados para os exemplos e o Power BI para a construção dos dashboards de exemplo.

5 ARQUITETURA DA APLICAÇÃO.

O Micromorph apresenta uma arquitetura *schemaless*, isso significa que o *payload* enviado não precisa aderir a um formato padronizado. No entanto existe a

necessidade de armazenar metadados para viabilizar a busca de forma eficaz e disponibilizar contexto aos dados, eles podem ser fornecidos pelo usuário no *payload*, caso não informado, será gerado em cima das informações coletadas da requisição, para fornecer os metadados é necessário adicionar os seguintes dados a requisição:

Chave	Valor Padrão
micromorphMetaData.source	API
micromorphMetaData.documentFormat	json
micromorphMetaData.name	UUID
micromorphMetaData.labels	Vazio

Os campos *name* e *labels* podem ser utilizados pelos usuários para poder classificar suas requisições. O diagrama presente em Apêndice A - Figura 1, exibe que a aplicação é dividida em três camadas distintas, sendo elas a camada de controle, a camada de serviço e a camada de persistência.

A camada de controle é a qual lida com a entrada das requisições HTTPS(*Hyper Text Transfer Protocol Secure*) e AMQP(*Advanced Message Queuing Protocol*) transformando o *payload* em um objeto de metadados do Micromorph, se a requisição não incluir metadados explícitos, a camada de controle encaminha o objeto para a camada de serviço, onde os metadados necessários serão criados automaticamente com base nas informações presentes no payload. Esse processo garante que todos os dados processados pela aplicação sejam acompanhados por metadados apropriados, facilitando a indexação e seu uso posterior. O protocolo HTTPS é utilizado para a comunicação segura via web, garantindo que os dados transmitidos entre o cliente e o servidor sejam criptografados, protegendo assim a integridade e a confidencialidade das informações. O protocolo AMQP é utilizado principalmente em sistemas de mensagens assíncronas, permitindo a comunicação entre diferentes partes de um sistema distribuído de maneira confiável e eficiente.

Camada de serviço, possuem as regras de negócio da aplicação, caso o objeto informado não esteja de acordo com mínimas restrições(formato suportado e tamanho) é retornado para a camada de controle como exceção.
Caso positivo ela processa o objeto e cria metadados internos, tamanho do arquivo, *hash256,* o *unix timestamp*, além disso é responsável por todas as transformações de dados que ocorrem durante uma solicitação de busca, Isso inclui a construção da paginação, que é a divisão dos resultados em partes menores para facilitar a visualização e o gerenciamento, especialmente em casos de grandes volumes de dados. A paginação melhora a usabilidade da aplicação, permitindo que os usuários naveguem pelos resultados de forma mais eficiente(Bilal Ahmad, 2021).

Camada de persistência consiste em tudo relacionado aos processos de indexação e busca, sendo necessário fornecer proteção as exceções possíveis durante a etapa de indexação, e também ferramentas para a construção de consultas(*queries*) complexas utilizando os metadados anteriormente indexados.
Durante a etapa de indexação, os dados são organizados e preparados para futuras consultas, no entanto, esse processo pode encontrar diversas exceções e erros, como problemas de conectividade ou falhas no sistema de indexação.

6 DOCUMENTAÇÃO

Sendo o foco do Micromorph a sua fácil utilização em sistemas já existentes, é de alta prioridade uma documentação, que apresenta os seus recursos e como se integrar no sistema. O guia de como instalar está disponível no *readme* do projeto, e como utilizar suas funcionalidades estão disponíveis tanto no fonte quanto na página web servida pela aplicação, isso torna essencial para garantir que os desenvolvedores possam integrar e utilizar de forma eficiente, contribuindo para a adoção bem-sucedida da ferramenta.

Para realizar a implementação utilizando o docker basta realizar pull da imagem utilizando “docker pull rgrosa/micromorph:latest” como apresentado em apêndice B - Figura 1. Para executar com sucesso é necessário uma instância do banco de dados Elasticsearch, no *readme* presente no projeto, existe um exemplo de como configurar uma instância utilizando o Elasticsearch.

Ao executar a aplicação com sucesso, o servidor servirá uma página web que documenta como utilizar ela, os mesmos exemplos presentes, são apresentados em versão de “postman-collection” no código fonte do projeto, disponível no apêndice B - Figura 2.

A interface web servida pode ser utilizada como documentação de uso e também como forma de envio de dados, as Figuras 3, 4, 5 e 6 do apêndice B apresentam cada uma das telas disponíveis.

- Figura 3 - Apresenta a página inicial, com as funcionalidades disponíveis na versão atual
- Figura 4 - Disponibiliza uma forma de upload manual no formato csv e json
- Figura 5 - Documenta cada um dos endpoints do serviço, explicando suas funções e como utilizá-los, com exemplos no formato *curl*
- Figura 6 - Oferece um guia para configurar a integração do sistema de mensageria com a aplicação

7 EXEMPLO DE USO PARA O MICROMORPH

Durante a formação do projeto foram criados dois exemplos de utilização, onde o Micromorph poderia ser inserido, esses cenários foram criados pensando em casos reais de implementação em ambientes corporativos, onde eu necessitei criar serviços novos, para controlar a integração de dados entre um sistema externo, com o sistema interno do cliente que eu estava encarregado.

O Primeiro exemplo está apresentado em Apêndice C - Figura 1, nesse exemplo possuímos um cliente externo que disponibiliza dados de consulta para um cliente interno, o mesmo deseja ver as informações do cliente externo em um dashboard. Se o cliente externo possuir uma API de leitura, depender dela não seria ideal devido a diversos fatores. Primeiramente, a estabilidade do serviço externo está fora do nosso controle, o que pode resultar em indisponibilidade inesperada. Além disso, serviços externos geralmente possuem limites de banda de leitura, que não são transparentes para os usuários e podem impactar a aplicação final. Para diminuir os riscos, o ideal seria coletar os dados do serviço externo e trazê-los para um ambiente interno, onde possuímos o total controle sobre a disponibilidade dos dados. O Micromorph pode ser inserido nesse contexto para abstrair o uso de um novo middleware, esse processo se torna simplificado porque o é aceito qualquer

tipo de dado, permitindo que o serviço externo forneça informações sem precisar modificar seu payload. Assim, o cliente interno pode utilizar os dados da forma desejada dentro de seu próprio ambiente, garantindo maior segurança, eficiência e controle.

Após os dados serem indexados no Micromorph pode então ser montadas consultas para a busca, utilizando um dos endpoints disponíveis na documentação, com isso pode ser construído um dashboard. Pode ser visto na figura 3 - Apêndice C, um dashboard construído com dados mock, eles foram todos indexados utilizando o Micromorph, dentro do fonte do projeto foi deixado o exemplo, um "*script python"* que busca esses dados da ferramenta "Mockaroo", e insere no servidor, sem realizar nenhum processamento.

O Segundo exemplo (Figura 2 - Apêndice C) apresenta um caso onde se usa em *cluster.* Para a utilização é necessário apenas uma única instância do Elasticsearch, a implementação em cluster permite subir várias instâncias do Micromorph. Cada uma dessas instâncias utiliza o mesmo banco de dados centralizado, mas com índices singulares e específicos para cada instância, essa implementação apresenta dois principais benefícios: redução de custo com infraestrutura e centralização no gerenciamento de dados. Nesse exemplo foi utilizado diferentes formas de consumo, na Figura 4 - apêndice B, apresenta um dashboard de controle da aplicação, criado com as informações dos metadados consumidos.

Isso demonstra que além de utilizar os dados que foram ingeridos pela aplicação, pode se utilizar os metadados gerados para proporcionar uma visão clara do funcionamento da aplicação, facilitando a tomada de decisões para futuras melhorias no projeto.

8 TRABALHOS FUTUROS

Mesmo com a implementação finalizada e o serviço já sendo disponibilizado livremente, alguns pontos foram levantados que poderiam proporcionar uma melhoria na utilização da aplicação, abrindo novas possibilidades de utilização. Essas melhorias deverão ser estudadas com um certo cuidado para não afetar o desempenho da aplicação e não ferir a ideia original de ser um serviço genérico:

Validar os dados que são recebidos. Hoje só é realizada uma validação simples em cima do payload, levando em consideração o peso e se os dados não são vazios, não existe validação em cima dos dados pois isso impactaria na regra de negócio. Um trabalho futuro seria a adição de um módulo, onde o usuário poderia fornecer um JSON Schema. O JSON Schema é uma notação que permite validar documentos json, nele é escrito como os dados de um objeto json devem se comportar, como todos os dados antes de serem indexados são transformados em json, essa notação pode ser utilizada. Existe uma ampla gama de ferramentas que suportam a notação, incluindo para a linguagem de programação Java, sendo assim possível suportar.(JSON SCHEMA documentation, 2024). Lembrando que essa funcionalidade deve ser opcional para o cliente, pois a base do Micromorph é ser genérico.

Permitir o envio de dados. O Micromorph é um sistema passivo, somente recebendo dados, uma melhoria seria implementar um sistema para enviar dados para um local específico. Sugere-se a implementação de um serviço de *callback,* que poderia ser configurado tanto na configuração geral da aplicação quanto em payloads específicos durante a indexação dos dados, isso pode ser útil caso seja

desejado persistir em um database final. Além do serviço de callback, poderia ser implementada a produção de mensagens, visto que hoje já é possível realizar a leitura de filas do RabbitMQ.

Fornecer Implementações de database distintas. Para o funcionamento da aplicação é necessário possuir uma instância de elasticsearch para poder realizar as consultas e a indexação dos dados. Como se pode ver na figura A - Apêndice D, todo o serviço responsável pela persistência está descrito em uma classe de interface, abrindo a possibilidade de criar diferentes implementações, isso aumentaria o escopo de clientes que poderiam utilizar a aplicação, dessa forma teríamos diferentes implementações do micromorph para bancos de dados distintos. Seria necessário um estudo mais aprofundado para analisar o impacto de performance ao utilizar outro banco de dados não relacional.

9 CONSIDERAÇÕES FINAIS

Em conclusão, foi possível concretizar o projeto desejado, demonstrando que o Micromorph suporta os formatos inicialmente previstos e pode ser utilizado para integrações simples de dados. Nos testes realizados, o Micromorph mostrou-se eficaz no gerenciamento de dados de diferentes fontes, mantendo um bom desempenho nas buscas e possibilitando a geração de diferentes resultados a partir dos dados indexados. Exemplos desses resultados incluem um dashboard de monitoramento da aplicação e um dashboard de vendas. Também foi possível desenvolver uma documentação que auxilia na utilização dos endpoints e disponibilizar o serviço como uma imagem no Docker.

Entretanto, não foi possível realizar testes concretos em ambientes reais corporativos ou acadêmicos, por longos períodos para comprovar a estabilidade do software. Todos os testes foram realizados com dados mock por um período de alguns dias. Para realizar testes mais abrangentes, seria necessária a parceria com terceiros que utilizariam o Micromorph em cenários reais. No futuro, serão buscadas organizações que possam se beneficiar do uso desse serviço. O projeto pode se beneficiar de melhorias, especialmente na adição de novas funcionalidades que permitam aos usuários utilizá-lo de novas maneiras.

REFERÊNCIAS

AWS, O que é um middleware ? 2023
Disponível em: https://aws.amazon.com/pt/what-is/middleware/l Acesso em: 14 Apr. 2024.

BILAL, Ahmad, What is API Pagination? 2021 Disponível em: https://rapidapi.com/guides/api-pagination Acesso em: 20 maio. 2024

COIMBRA, Vinícius, Comparando: Elasticsearch vs MongoDB, 2018
Disponível em: https://medium.com/data-hackers/comparando-elasticsearch-vs-mongodb-4b5932c613d9
Acesso em: 15 Apr. 2024.

CONCEIÇÃO, Marcus. Bancos de Dados Relacionais vs Não Relacionais: Qual o melhor afinal? Dio 2023. Disponível em: https://www.dio.me/articles/bancos-de-dados-relacionais-vs-nao-relacionais-qual-o-melhor-afinal Acesso em: 24 Nov. 2023.

DOCKER, documentation, Docker Overview , 2024
Disponível em: https://docs.docker.com/get-started/overview/
Acesso em: 15 Apr. 2024.

IEEE 35TH INTERNATIONAL CONFERENCE ON DATA ENGINEERING WORKSHOPS (ICDEW), 35, 2019,Macao. Implementing big data lake for heterogeneous data sources Macao China: IEEE, 2019. 37 - 44 disponível em: http://jultika.oulu.fi/files/nbnfi-fe2019082024798.pdf Acesso em: 16 out 2023

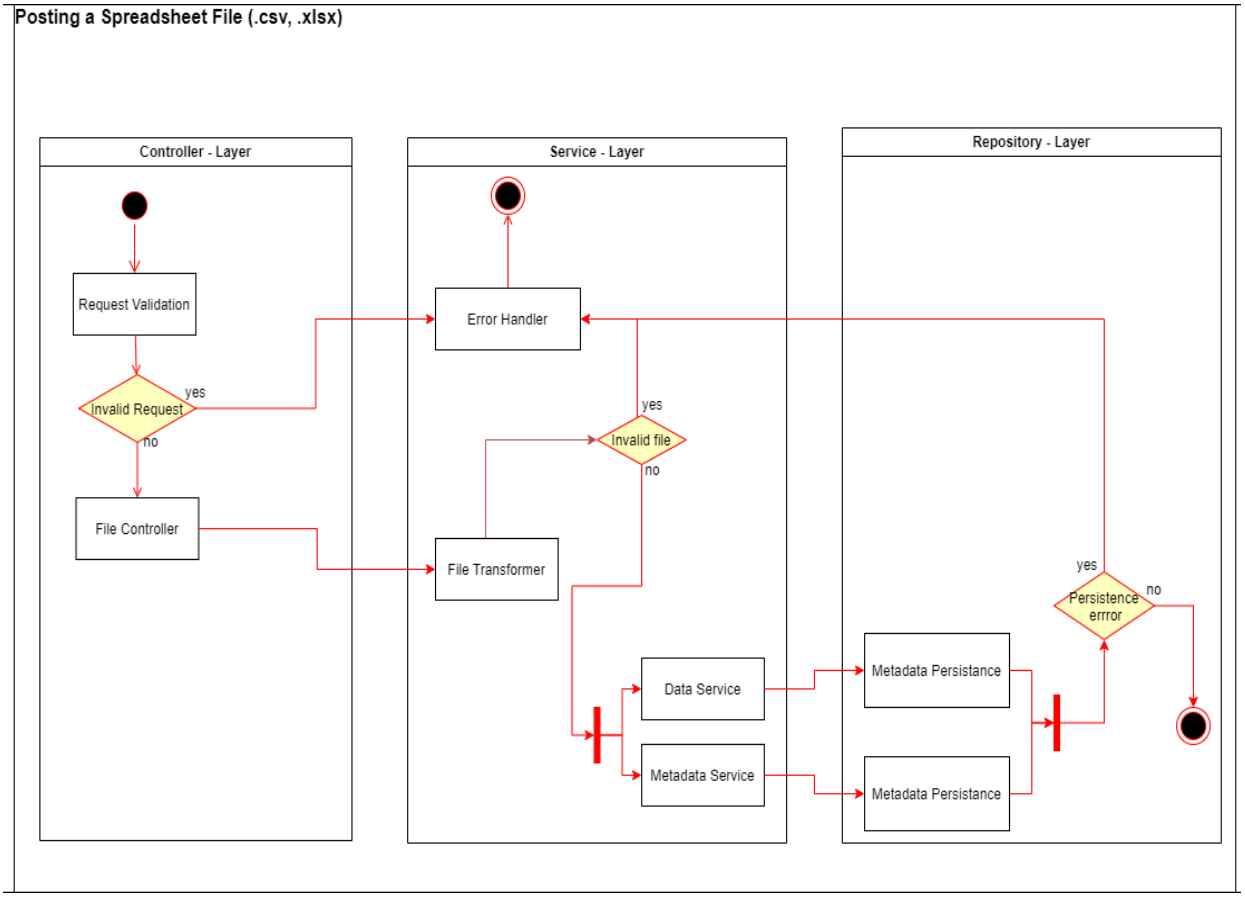
JSON SCHEMA, documentation, Introduction to JSON Schema, 2024
Disponível em: https://json-schema.org/learn/getting-started-step-by-step
Acesso em: 10 Jun. 2024

SOUZA, Emmanuel Neri De Souza. Um comparativo de arquitetura de software de micro serviços em relação às arquiteturas monolíticas em ambiente corporativo. 2016. Dissertação (Mestrado Profissional em Desenvolvimento de Tecnologia) - Instituto de Tecnologia para o Desenvolvimento, Curitiba 2016. Disponível em: https://sucupira.capes.gov.br/sucupira/public/consultas/coleta/trabalhoConclusao/viewTrabalhoConclusao.jsf?popup=true&id_trabalho=6090473#
Acesso em: 16 out. 2023.

VOLLE, Adam. metadata, 2024 Disponível em: https://www.britannica.com/technology/metadata Acesso em: 29 Apr. 2024.

APÊNDICE A - ARQUITETURA

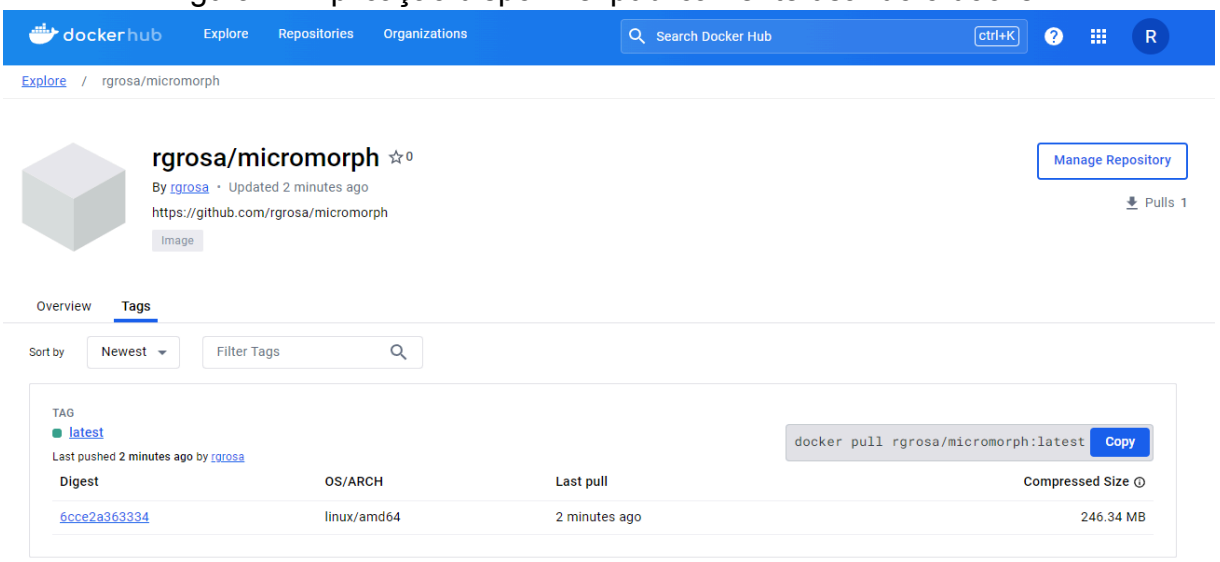
Figura 1 - Demonstra as diferentes camadas da aplicação



Fonte: Criado pelo autor (2023)

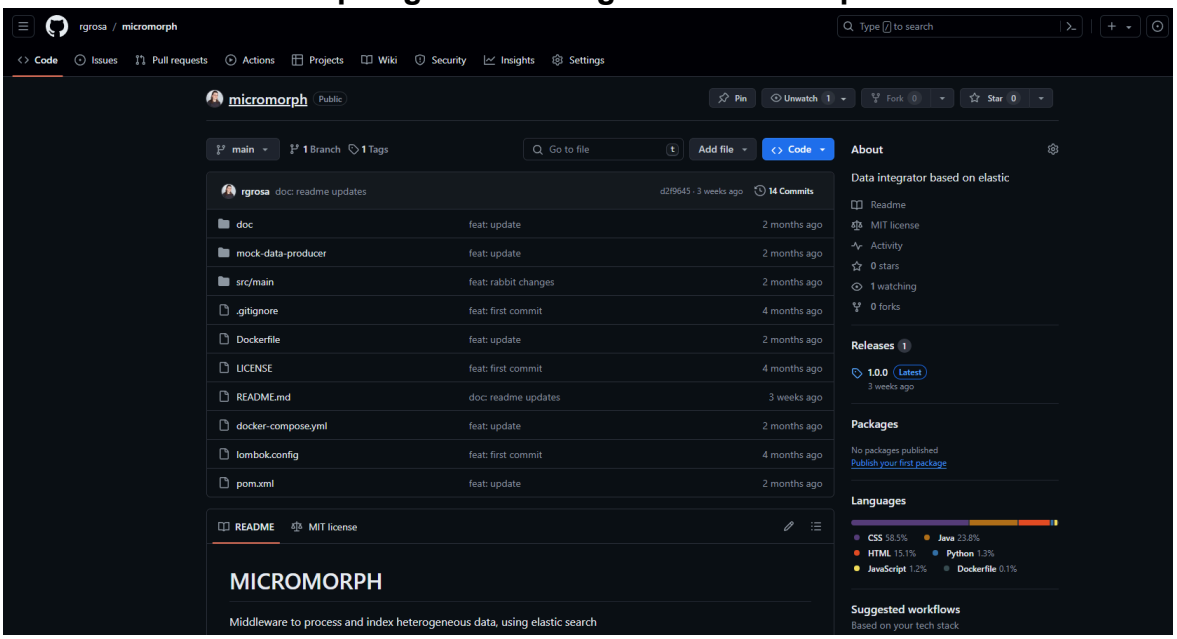
APÊNDICE B - DOCUMENTAÇÃO

Figura 1 - Aplicação disponível publicamente usando o docker



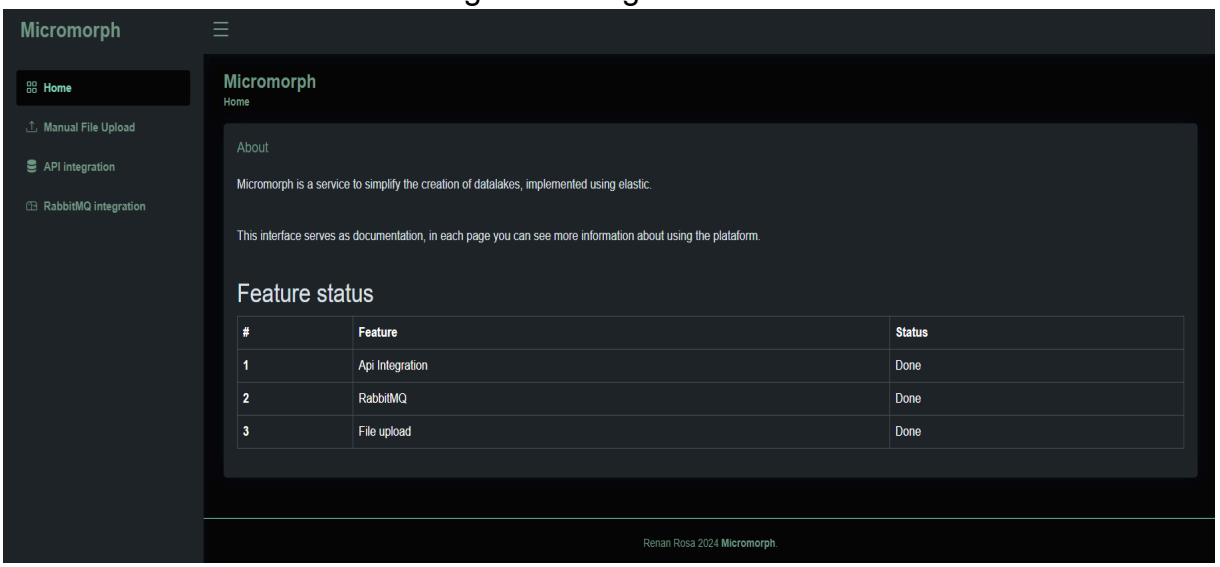
Fonte: Criado pelo autor (2024)

Figura 2 - Repositório do projeto disponivel em **https://github.com/rgrosa/micromorph**



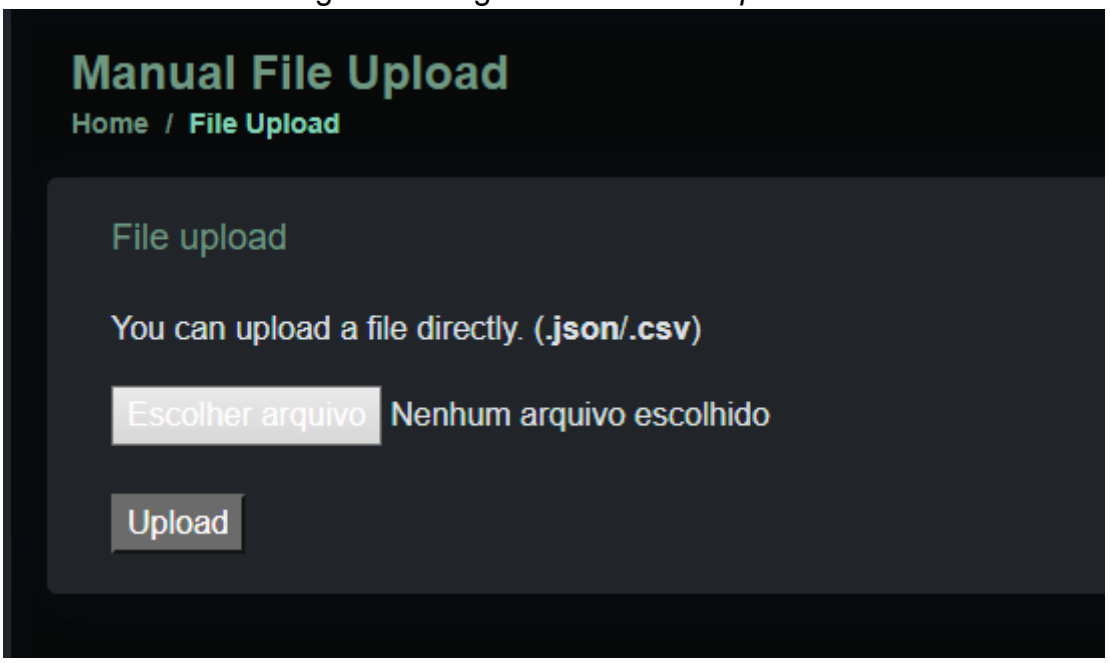
Fonte: Criado pelo autor (2024)

Figura 3 - Página home



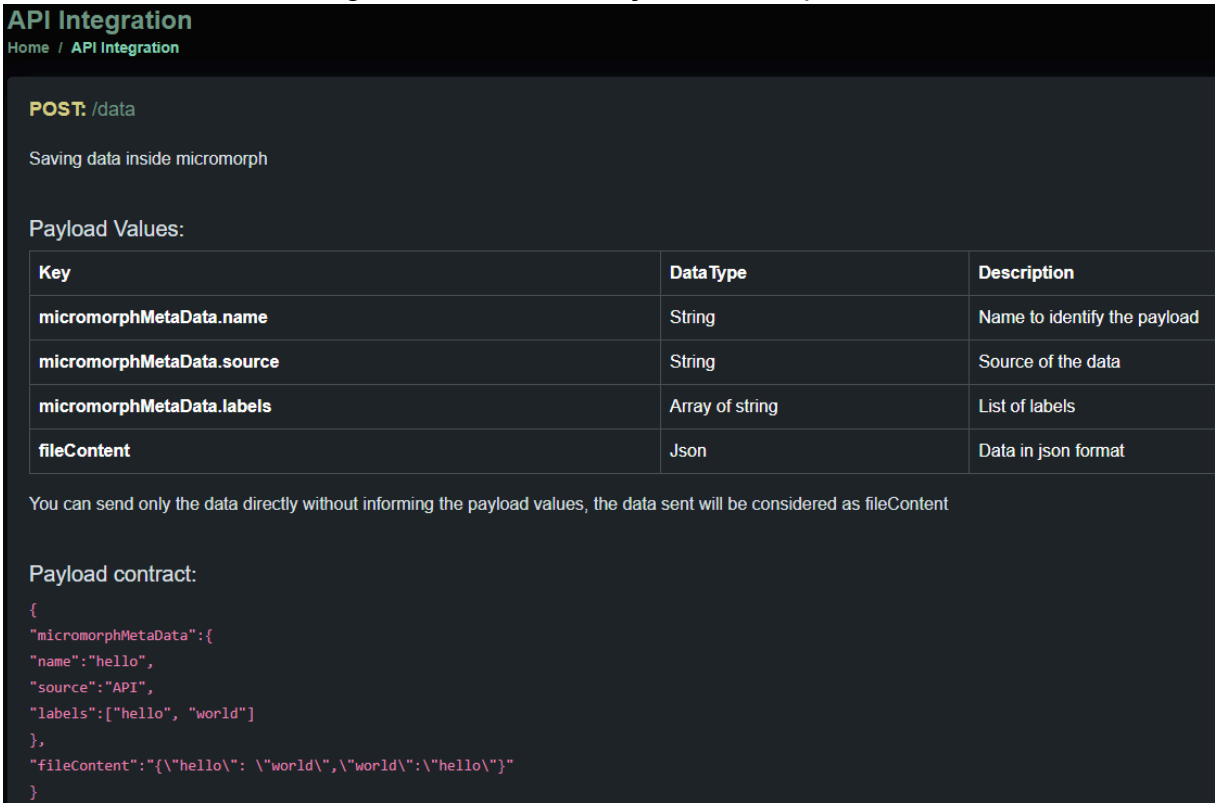
Fonte: Criado pelo autor (2024)

Figura 4 - Página *Manual File Upload*



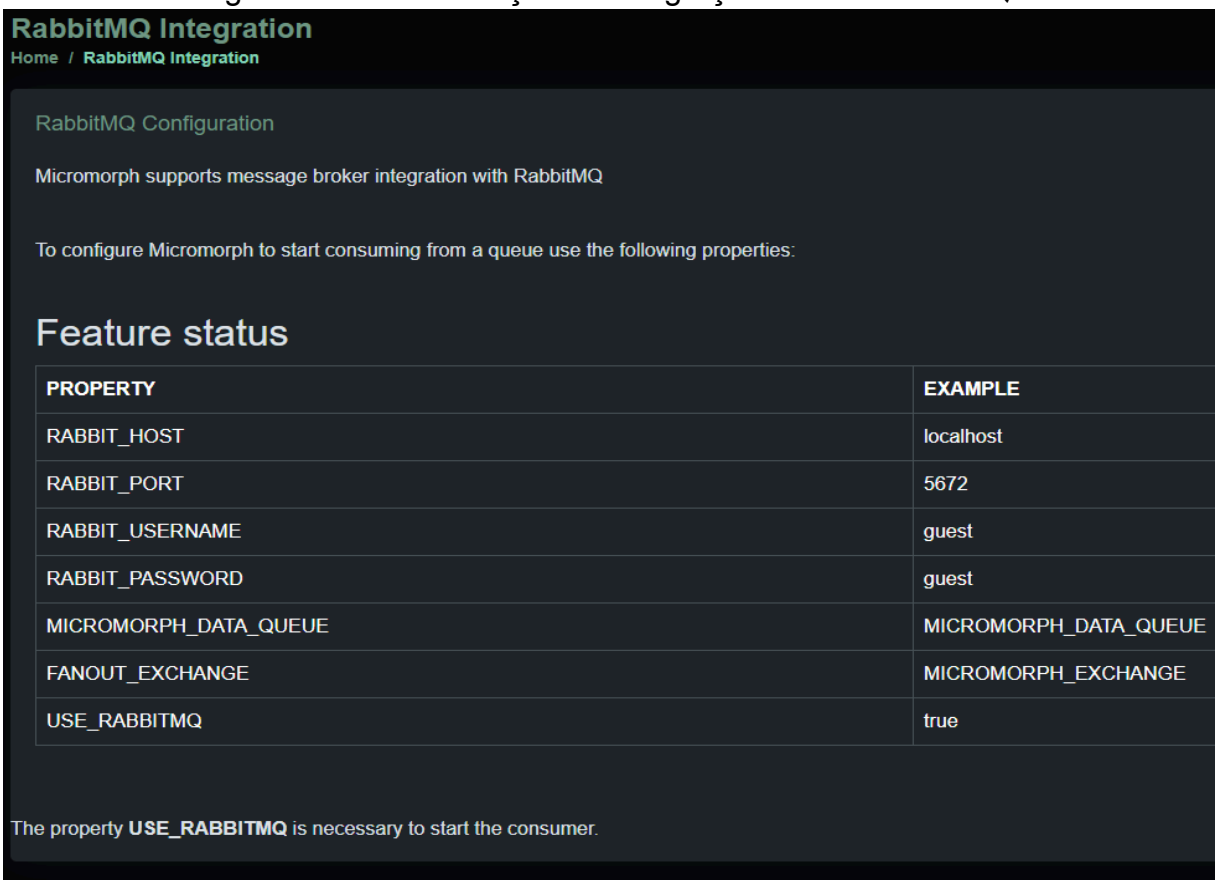
Fonte: Criado pelo autor (2024)

Figura 5 - Documentação dos *Endpoints*



Fonte: Criado pelo autor (2024)

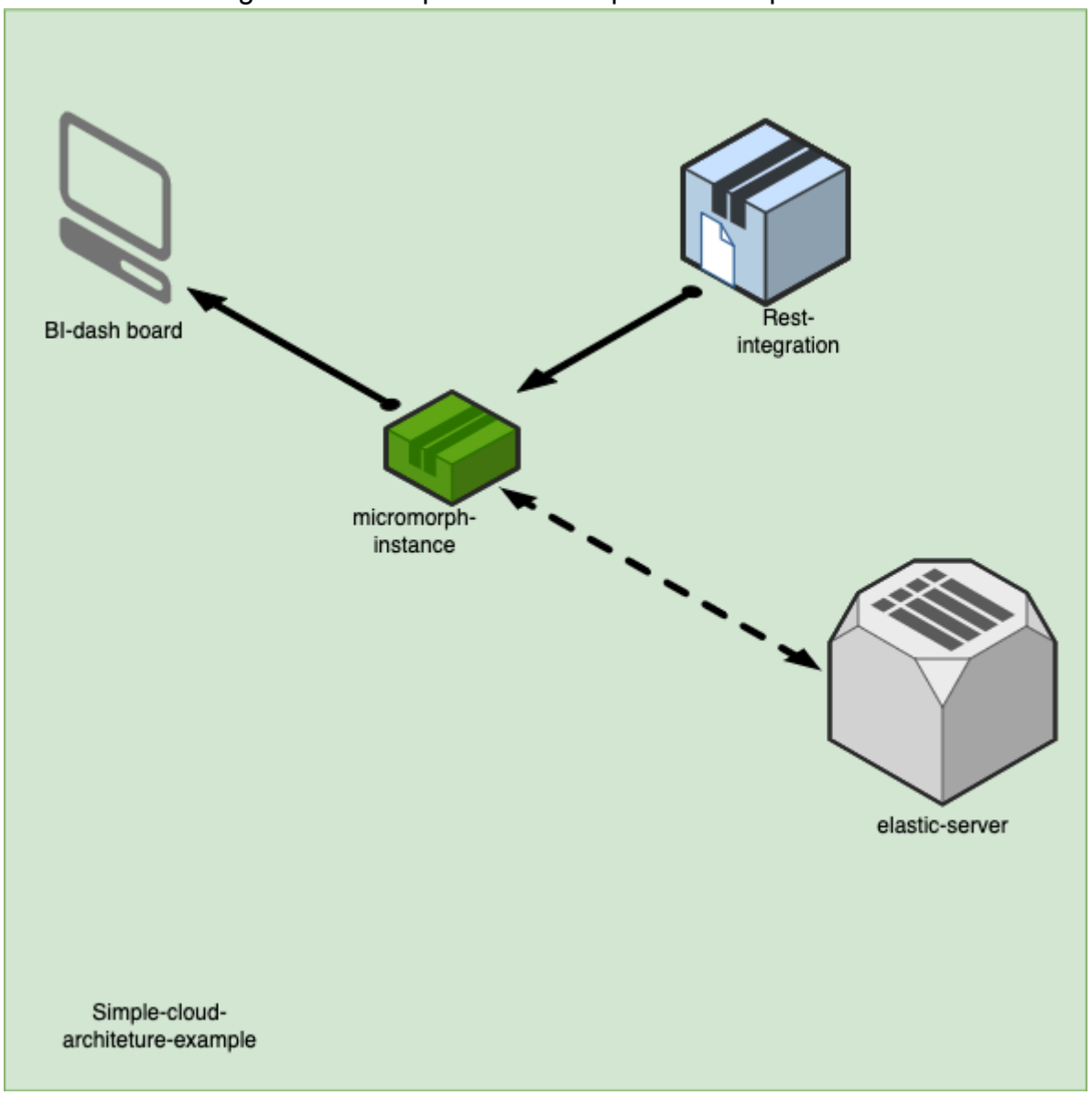
Figura 6 - Documentação da integração com o *RabbitMQ*



Fonte: Criado pelo autor (2024)

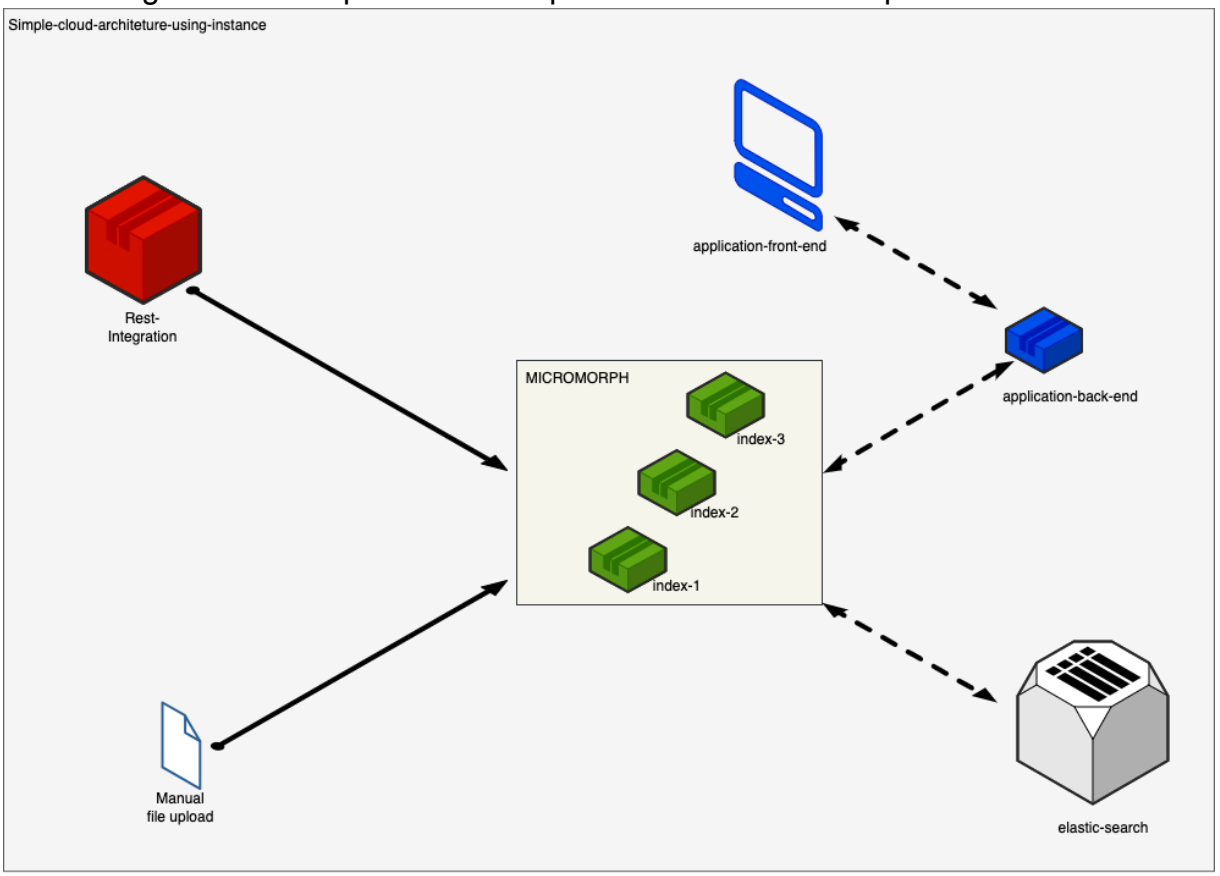
APÊNDICE C - EXEMPLO DE USO PARA O MICROMORPH

Figura 1 - Exemplo de uma arquitetura simplificada



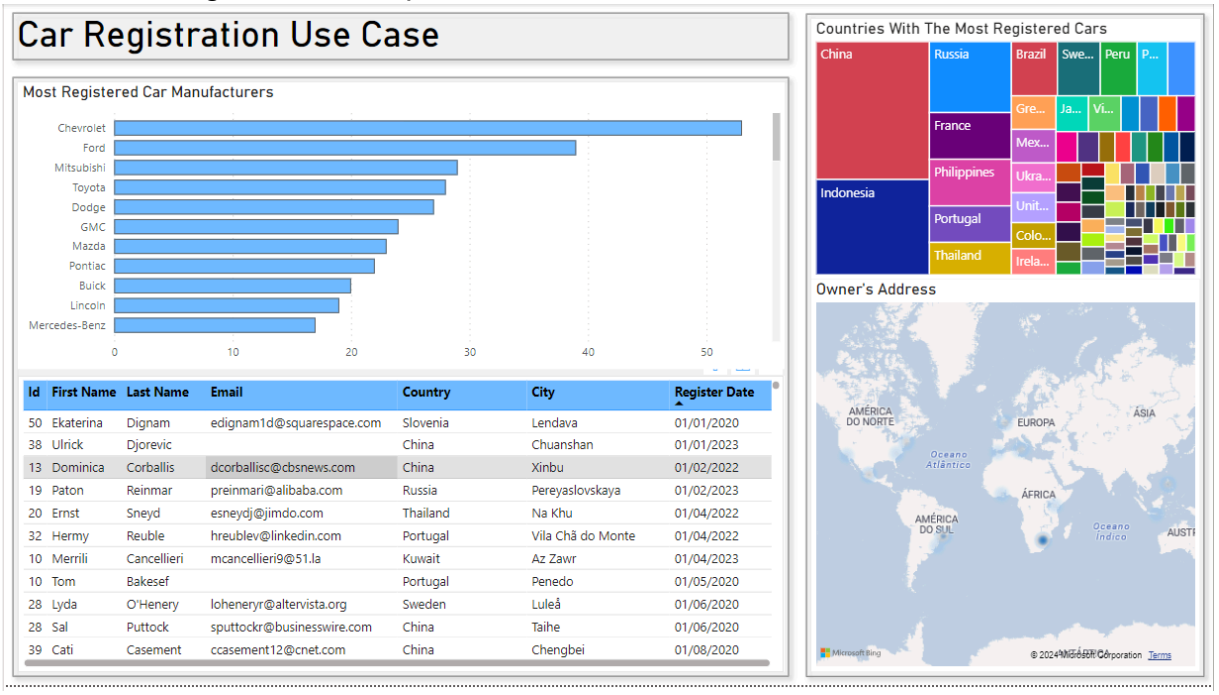
Fonte: Criado pelo autor (2024)

Figura 2 - Exemplo de uma arquitetura utilizando múltiplas instâncias



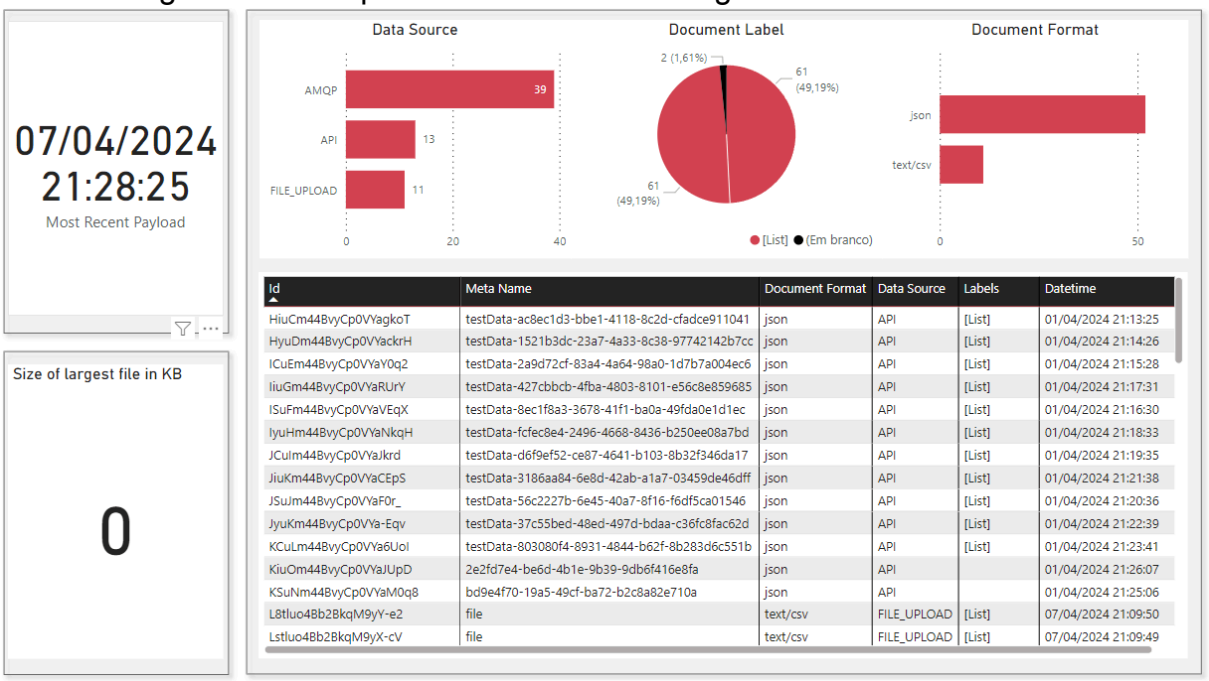
Fonte: Criado pelo autor (2024)

Figura 3 - Exemplo de um dashboard com dados de usuário



Fonte: Criado pelo autor (2024)

Figura 4 - Exemplo de um dashboard de gerenciamento de dados



Fonte: Criado pelo autor (2024)

APÊNDICE D - TRABALHO FUTURO

Figura 1 - Classe de interface da persistência de dados

```java
package br.com.micromorph.domain.service;

import br.com.micromorph.domain.dto.RequestByMetadataDTO;
import br.com.micromorph.domain.entity.Data;
import br.com.micromorph.infrasctructure.exception.PersistenceDeserializationException;
import org.springframework.data.domain.Page;
import org.springframework.data.elasticsearch.core.SearchHits;

import java.io.IOException;
import java.util.List;

4 usages  1 implementation  rgrosa
public interface DataServicePersistence {

    1 usage  1 implementation  rgrosa
    Data insertIntoDataIndex(Data data) throws IOException, PersistenceDeserializationException;

    1 usage  1 implementation  rgrosa
    void insertBatchIntoDataIndex(List<Data> data) throws IOException, PersistenceDeserializationException;

    1 usage  1 implementation  rgrosa
    Page<Data> findAll(Integer page);

    1 usage  1 implementation  rgrosa
    SearchHits<Data> findAllByMetadata(RequestByMetadataDTO requestByMetadata);

    1 usage  1 implementation  rgrosa
    void deleteDataById(String id) throws PersistenceDeserializationException;

    1 usage  1 implementation  rgrosa
    Data findById(String id);

}
```

Fonte: Criado pelo autor (2024)