Microsserviços para gerenciamento de imagens dos projetos NIMPI e MIV

Vilhena - RO

2022

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE RONDÔNIA – CAMPUS VILHENA

Microsserviços para gerenciamento de imagens dos projetos NIMPI e MIV

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – campus Vilhena, realizado em cumprimento de requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Marco Antônio Augusto de Andrade

Vilhena - RO

2022

FICHA CATALOGRÁFICA

Biblioteca IFRO – Campus Vilhena

S232m
SANT'ANA, Gabriel Amorim
Microsserviços para gerenciamento de imagens dos projetos NIMP e MIV / Gabriel Amorim Sant'Ana – Vilhena, Rondônia, 2022.

48f. : il.
Orientador : Prof. Me. Marco Antonio Augusto de Andrade

Trabalho de Conclusão de Curso (Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO

1. Microsserviços 2. Extração de textos 3. Extração de imagens 4. Kanban I. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – IFRO II. Título

005.13

Bibliotecária responsável Rosilene Maria do Couto Marques CRB 11/321

Microsserviços para gerenciamento de imagens dos projetos NIMPI e MIV

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia – campus Vilhena, realizado em cumprimento de requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Trabalho aprovado. Vilhena - RO, 29 de novembro de 2022

gov.br Documento assinado digitalmente
MARCO ANTONIO AUGUSTO DE ANDRADE
Data: 08/12/2022 08:28:39-0300
Verifique em https://verificador.iti.br

Msc. Marco Antonio Augusto de Andrade
Orientador

gov.br Documento assinado digitalmente
ERICK LEONARDO WEIL
Data: 08/12/2022 15:17:32-0300
Verifique em https://verificador.iti.br

Esp. Erick Leonardo Weil
Convidado 1

gov.br Documento assinado digitalmente
ROSA MARIA DA SILVA GONCALVES
Data: 08/12/2022 14:10:39-0300
Verifique em https://verificador.iti.br

Msc. Rosa Maria da Silva Gonçalves
Convidado 2

Vilhena - RO
2022

Este trabalho é dedicado às pessoas inocentes que tiveram suas vidas dizimadas pelos horrores da guerra.

Agradecimentos

A toda minha família por sempre acreditar no poder emancipador da educação. Aos meus pais, Eloisa e Luiz, pela perseverança em propiciar o conforto e as condições necessárias para que seus filhos tivessem uma educação de qualidade. Aos meus avós, Elio, Moema, Rubem e Teresinha pela sabedoria e amor transmitidos durante o ciclo de suas vidas. À minha esposa Katia por estar sempre ao meu lado e por ter me incentivado a ingressar no curso de Análise e Desenvolvimento de Sistemas.

A todos os professores do Instituto Federal de Educação, Ciência e Tecnologia de Rondônia, Campus Vilhena, pelos conhecimentos passados durante os últimos cinco anos. Ao professor Marco Antônio por possibilitar meu engajamento nos projetos do NIMPI e do MIV e por me orientar durante o desenvolvimento deste trabalho.

Aos amigos conquistados, durante o curso, que contribuíram para minha edificação como pessoa. Aos meus amigos Evaldo, Ericky, Rodrigo e Willian por suas contribuições no desenvolvimento deste trabalho.

Nas grandes batalhas da vida, o primeiro passo para a vitória é o desejo de vencer.

Mahatma Gandhi

Resumo

Esta monografia apresenta o processo de desenvolvimento de microsserviços para as plataformas do NIMPI e do MIV. O objetivo deste projeto é aprimorar a arquitetura utilizada atualmente pelas plataformas, fornecendo maior capacidade de processamento e propiciando uma maior organização e estruturação de suas funcionalidades. A criação dos diagramas presentes, neste trabalho, seguiram os princípios UML (Unified Modeling Language). Para o desenvolvimento dos serviços, foram utilizados o método Kanban para o gerenciamento do processo e a plataforma GitLab para o versionamento da aplicação. Ao término do projeto, os serviços desenvolvidos foram entregues à Fábrica de Software do IFRO (FSLab) para serem integrados às plataformas supracitadas.

Palavras-chave: microsserviços; extração de textos; extração de imagens; redimensionamento de imagens; Kanban; Vilhena.

Abstract

This monograph presents the microservices development process for the NIMPI and MIV platforms. The objective of this project is to improve the architecture currently used by the platforms, providing greater processing capacity and providing greater organization and structuring its features. The creation of the diagrams present, in this work, followed the UML (Unified Modeling Language) principles. For the development of services, the Kanban method was used to manage the process and the platform GitLab for application versioning. At the end of the project, the services developed were delivered to the IFRO Software Factory (FSLab) to be integrated into the aforementioned platforms.

Keywords: Microservices; Text extraction; Image extraction; Image resizing; Kanban; Vilhena.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

PIL	Python Image Library
PDF	Portable Document Format
OCR	Optical Character Recognition
UML	Unified Modeling Language
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
API	Application Programming Interface
MIV	Museu da Imagem de Vilhena
NIMPI	Núcleo Informatizado de Memória e Pesquisa do IFRO
MIT	Massachussetts Institute of Technology
IFRO	Instituto Federal de Rondônia
UTF-8	8-bit Unicode Transformation Format

Sumário

1 Introdução

1.1 Contexto e problema

O Núcleo Informatizado de Memória e Pesquisa do IFRO (NIMPI) é um laboratório de pesquisa, digitalização, preservação e divulgação de documentos históricos da memória de Rondônia e da Amazônia. Criado em 2013, junto ao Departamento de Extensão, o NIMPI é orientado pelos princípios da Ciência Aberta e tem como um de seus pilares o acesso aberto aos dados. Seu ideal é de que o conhecimento é um bem público de todos e para todos. O objetivo do NIMPI é democratizar as informações histórico-culturais regionais, preservando as memórias por meio da digitalização de documentos antigos. O NIMPI também fornece gratuitamente ao público, por intermédio de seu portal, a busca dos documentos digitalizados com o auxílio da tecnologia de Reconhecimento Óptico de Caracteres (Optical Character Recognition - OCR) (NIMPI, 2018).

O Museu da Imagem de Vilhena (MIV) é o primeiro museu virtual de Rondônia bem como a maior iniciativa de resgate imagético histórico de Vilhena. Criado no ano de 2021, pela Prefeitura de Vilhena em parceria com o IFRO, o MIV tem como foco transformar arquivos guardados em caixas e armários em uma história viva por meio da digitalização desses documentos. O MIV, ainda, disponibiliza a consulta pública dos arquivos digitalizados por meio de seu portal na rede mundial de computadores (MIV, 2022).

Ambos os projetos já se encontram operacionais e disponíveis ao público por meio da internet. No entanto, o processamento de novos documentos enviados às plataformas é limitado a um arquivo por vez. Dessa forma, uma vez que, vários arquivos são enviados simultaneamente aos servidores, o tempo de espera para que o documento seja disponibilizado ao público aumenta exponencialmente a cada arquivo enviado. Considerando essa premissa, esta monografia irá expor a problemática: Como desenvolver microsserviços para gerenciar o serviço de imagens das plataformas do NIMPI e MIV?

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo geral deste trabalho é aprimorar as funcionalidades já existentes nas plataformas do NIMPI e do MIV, por meio do desenvolvimento de microsserviços, para gerenciar o tratamento das imagens de ambas as aplicações.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Entender questões técnicas relacionadas ao processamento de imagens;
- Estudar sobre como o processo de digitalização de imagens é realizado;
- Elicitar os requisitos necessários para desenvolver os serviços;
- Desenvolver as funcionalidades elencadas por meio do levantamento dos requisitos;
- Implementar testes automatizados para as funcionalidades desenvolvidas;

1.3 Justificativa

A migração da arquitetura atual do NIMPI e do MIV para a arquitetura de microsserviços é essencial para que as plataformas ampliem seus poderes de processamento e possam crescer de forma estruturada. Para alcançar esses objetivos, faz-se necessária a fragmentação da aplicação em pequenos serviços escaláveis independentes. A escalabilidade é uma das principais vantagens que aplicações orientadas a microsserviços possuem em relação a aplicações monolíticas.

A adoção da arquitetura de microsserviços em substituição à monolítica vem sendo motivada pelas dificuldades de escalabilidade, baixa eficiência, lentidão no desenvolvimento e alta complexidade na adoção de novas tecnologias em sistemas monolíticos complexos (FOWLER, 2017, p. 21).

Essa mesma pesquisadora reforça ainda que:

> com a arquitetura de microsserviços, uma aplicação pode ser facilmente escalada tanto horizontalmente quanto verticalmente, a produtividade e a velocidade do desenvolvedor aumentam dramaticamente e tecnologias antigas podem ser substituídas por novas facilmente. (Ibidem, p. 21)

Levando em consideração os fatores mencionados anteriormente, é plausível a implementação de uma nova arquitetura para as plataformas supracitadas frente aos ganhos de desempenho, velocidade de processamento e adoção de novas tecnologias, qualidades essenciais para qualquer aplicação moderna que vise um crescimento sólido.

Os ganhos apresentados com a nova arquitetura beneficiarão não só os desenvolvedores e entidades administradoras do NIMPI e do MIV bem como pesquisadores, cientistas e a sociedade em geral, que terão à sua disposição um espaço virtual para acessar e compartilhar as memórias da região, podendo também contribuir com suas próprias produções científicas. O código das funcionalidades desenvolvidas será de domínio

público, podendo contribuir para futuras melhorias dos sistemas referidos e/ou derivações de *softwares* semelhantes.

2 Fundamentação teórica

2.1 Microsserviços

Microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual a aplicação é dividida em pequenos serviços independentes que se comunicam por meio de APIs. A tendência por microsserviços surgiu nos últimos anos pela necessidade de grandes empresas acelerarem o ciclo de atualizações de suas aplicações, disponibilizando novos recursos a seus clientes em curtos espaços de tempo (ZIADé, 2017, p. 1-10). Aplicações construídas, baseadas na arquitetura de microsserviços, podem ter seus serviços rapidamente escalados. De acordo com a demanda de processamento da aplicação, a produtividade e a velocidade da equipe de desenvolvimento aumentam drasticamente e, com a quebra do monólito em pequenos serviços e tecnologias obsoletas, podem ser facilmente substituídas por novas (FOWLER, 2017, p. 21).

A crescente demanda por entregas mais ágeis e sólidas, no cenário moderno, exigem que os ambientes de negócios sejam cada vez mais dinâmicos. Na arquitetura de microsserviços, cada serviço é implantado e opera de forma autônoma. Essa condição faz com que haja um ganho na agilidade do desenvolvimento da aplicação, uma vez que cada serviço torna-se uma unidade independente (SILVA, 2020, p. 24).

Segundo (AWS, 2022), a arquitetura de microsserviços traz consigo alguns benefícios, tais como:

- **Autonomia:** Um serviço pode ser desenvolviddo, implantado e escalado sem afetar o funcionamento de outro. Não há a necessidade de compartilhamento de código entre serviços. As comunicações entre componentes ocorrem com o auxílio de APIs.
- **Especialização:** Cada serviço é especializado em solucionar problemas específicos. Caso sejam adicionadas mais funcionalidades ao longo do tempo, o serviço poderá ter suas responsabilidades fragmentadas em novos serviços.
- **Agilidade:** Microsserviços propiciam uma maior organização de equipes pequenas e independentes. As equipes trabalham dentro de um contexto pequeno bem alinhado e possuem autonomia para trabalhar de forma independente.
- **Escalabilidade flexível:** Microsserviços permitem que cada serviço seja escalado de forma independente de acordo com a demanda. Essa característica permite que as equipes dimensionem adequadamente as necessidades de infraestrutura, meçam com precisão os custos dos recursos e mantenham a disponibilidade do serviço em horários de pico.

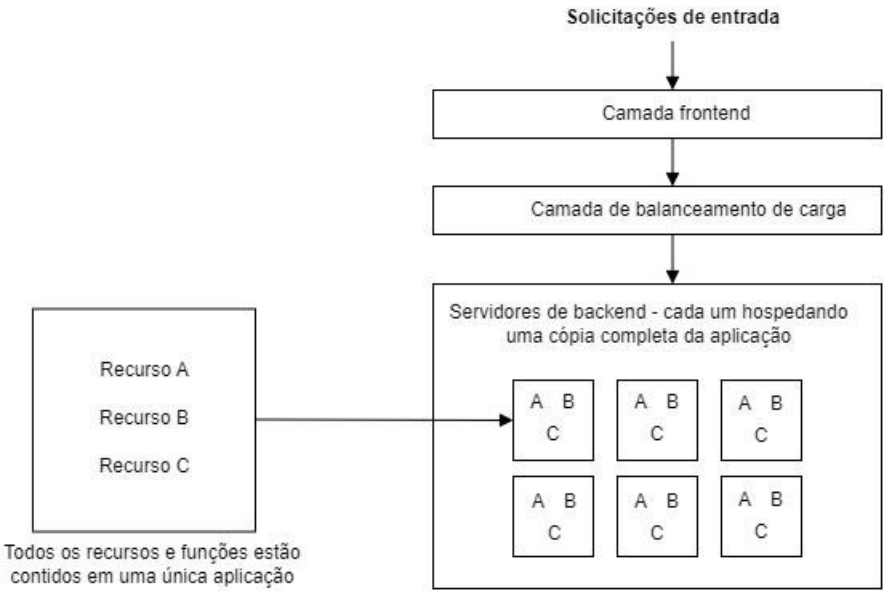
- **Liberdade tecnológica:** A arquitetura de microsserviços não possui abordagens engessadas. As equipes possuem liberdade para escolherem as melhores tecnologias de acordo com a necessidade. Uma aplicação baseada em microsserviços pode ter tecnologias diferentes entre seus serviços.
- **Código reutilizável:** A divisão do software em pequenos módulos permite a reutilização, de cada partícula, para outras finalidades. Um serviço desenvolvido para determinado objetivo, pode ser usado como componente básico para outro recurso.

2.2 Microserviços x monólitos

Uma aplicação monolítica possui todos seus recursos e funcionalidades implantados, ao mesmo tempo, em um grande bloco de código fonte único. Dessa forma, sempre que houver a necessidade de criação de uma nova instância da aplicação, uma cópia completa do sistema é executada no servidor hospedeiro (FOWLER, 2017, p. 27).

Na figura 1, podemos observar um exemplo de aplicação monolítica.

Figura 1 – Monólito



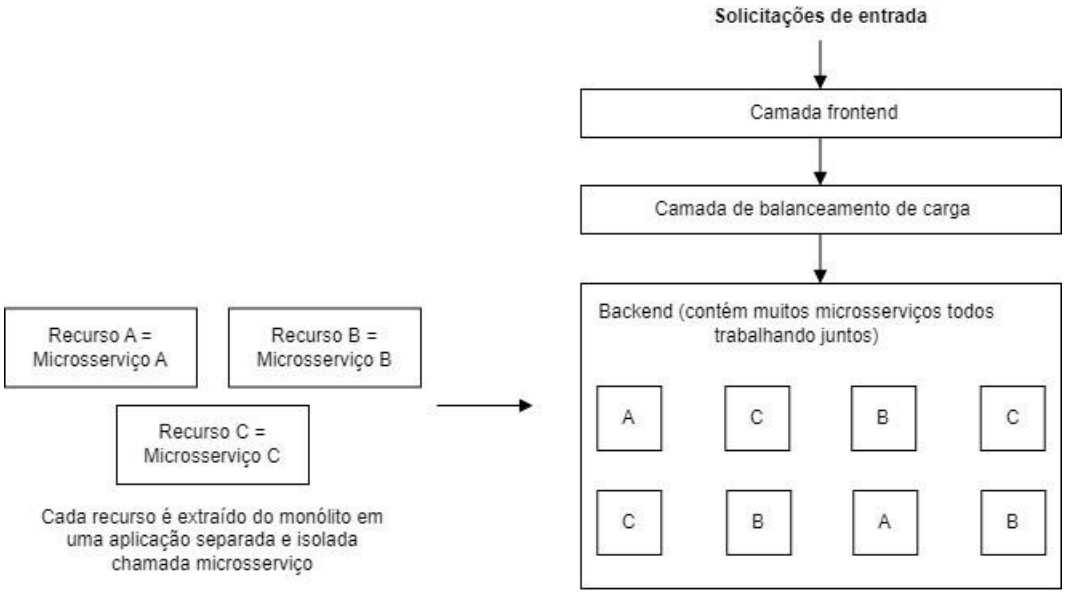
Fonte: Fowler (2017, p. 27)

Por outro lado, uma aplicação baseada na arquitetura de microsserviços, possui apenas um recurso ou função e é executada junto a outros microsserviços dentro de um ecossistema, cada uma possuindo responsabilidade única. Muitas organizações, geralmente, adotam essa arquitetura ao encontrarem desafios organizacionais e de escalabilidade após desenvolverem aplicações monolíticas (FOWLER, 2017, p. 27).

Ainda segundo a mesma autora, a divisão de uma aplicação monolítica em microsserviços nem sempre é uma tarefa fácil. A complexidade e os cuidados adicionais necessários para se obter sucesso aumentam de acordo com a quantidade de recursos existentes na aplicação atual. A principal regra, para se dividir uma aplicação monolítica em microsserviços, é identificar com precisão as principais funcionalidades e separá-las em pequenos componentes independentes. Esses componentes devem ser o mais simples possível, evitando a criação de monólitos menores. (Ibidem, p. 28-29)

A figura 2 exemplifica uma arquitetura baseada em microsserviços.

Figura 2 – Microsserviços



Fonte: Fowler (2017, p. 28)

2.3 Linguagem de Modelagem Unificada

A UML (Unified Modeling Language) é uma linguagem de modelagem utilizada internacionalmente pela indústria de engenharia de *software* para modelar aplicações orientadas a objetos (GUEDES, 2011, p. 19).

A modelagem de um *software* é essencial para que se possa estimar a quantidade de profissionais necessários para desenvolvê-lo, o tempo que levará para finalizá-lo, estimar custos e definir as tecnologias que serão utilizadas. Por mais simples que um sistema pareça ser, inicialmente, ele deve ser modelado para que possa crescer de forma estruturada (GUEDES, 2011, p. 20-23).

A utilização de diagramas UML possui uma série de vantagens as quais podem-se destacar (MARGARIDA, 2018, p. 42-45):

- Utilizada de forma eficaz em quase todas as áreas da Engenharia de Software.
- Existência de uma abundância de ferramentas disponíveis que suportam o padrão UML.
- Considerada a linguagem mais utilizada para modelar arquiteturas.
- Possibilita a modelagem de diferentes pontos de vista.
- Notação de fácil implementação e compreensão.
- Flexibilidade na adoção de diferentes estilos arquiteturais.

2.4 Processamento digital de imagens

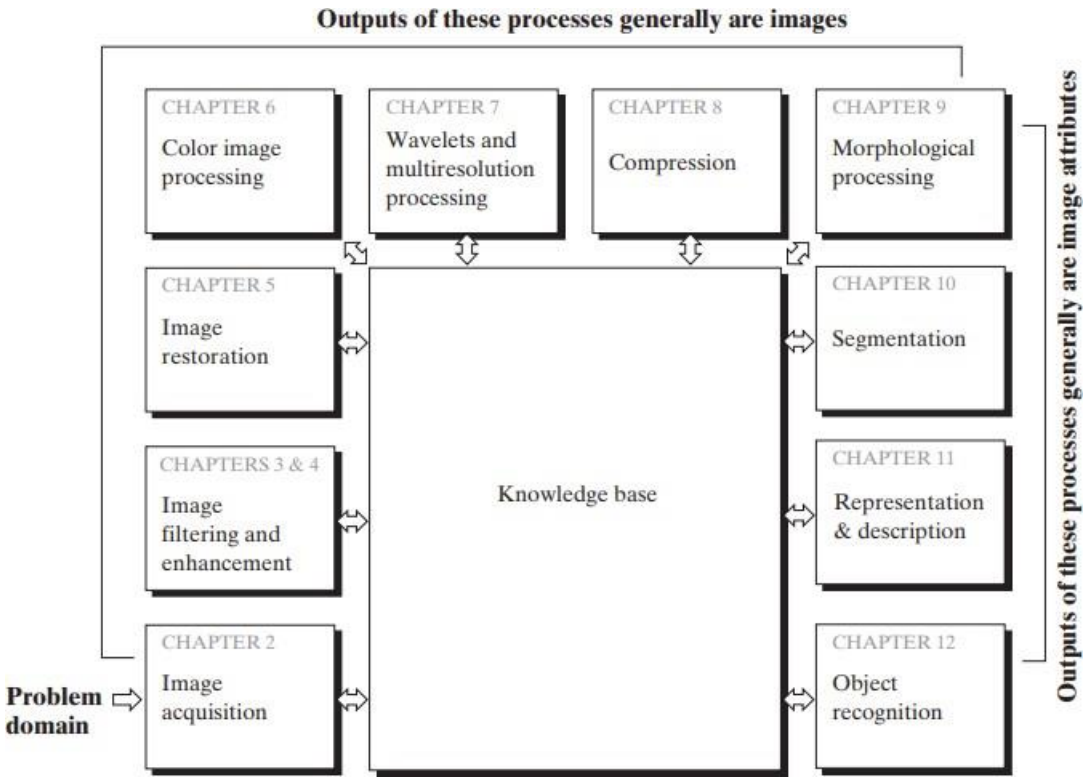
O campo do processamento digital de imagens refere-se a processos cujas entradas e saídas são imagens, além de processos que extraem atributos de imagens e processos que realizam o reconhecimento de objetos. Para exemplificar, podemos tomar como referência a área de análise automatizada de texto, cujos processos englobam: adquirir uma imagem da área em que o texto está contido, pré-processar a imagem, extrair os caracteres, descrever os elementos de forma apropriada para o seu correto processamento e, por fim, realizar o reconhecimento dos caracteres (GONZALES; WOODS, 2007, p. 23).

Não há limites claros no método de processamento de uma imagem. No entanto, um paradigma útil é o de considerar três categorias de processos computadorizados: baixo, médio e alto (GONZALES; WOODS, 2007, p. 24).

- Processos de baixo nível envolvem operações primitivas, como o pré-processamento de imagem para reduzir seu ruído ou o aprimoramento da nitidez e contraste da imagem. Um processo de baixo nível é caracterizado pelo fato de que suas entradas e saídas são imagens.
- Processos de nível médio envolvem tarefas como a segmentação (divisão da imagem em regiões ou objetos), descrição dos objetos para que possam ser processados de forma adequada e o reconhecimento de objetos. Um processo de nível médio é caracterizado pelo fato de que suas entradas, geralmente, são imagens, mas suas saídas são atributos extraídos dessas imagens (bordas, contornos e identificação de objetos).
- Processos de alto nível, envolvem "dar sentido"a um conjunto de objetos reconhecidos e realizar funções cognitivas relacionadas à visão humana.

A figura 3 mostra os passsos fundamentais no processamento digital de imagens:

Figura 3 – Passos fundamentais no processamento de imagens



Fonte: Gonzales e Woods (2007, p. 48)

2.5 Método Kanban

A palavra Kanban é oriunda do japonês e seu significado é cartão sinalizador. Uma determinada quantidade de kanbans define a capacidade de produção de um sistema. Uma nova tarefa só é iniciada quando há a existência de pelo menos um cartão. Assim que disponível, o cartão é vinculado a uma tarefa e a segue, à medida que, ela flui através do sistema. Novas tarefas devem esperar em uma fila até que um cartão seja finalizado (ANDERSON, 2011, p. 13).

Em desenvolvimento de software, o método Kanban é utilizado para limitar a quantidade de tarefas executadas simultaneamente em um projeto, definindo a capacidade e o equilíbrio da demanda de acordo com o que a equipe consegue produzir. Dessa forma, obtém-se um ritmo sustentável de desenvolvimento, fazendo com que todos os indivíduos possam alcançar um equilíbrio entre o trabalho e a vida pessoal (ANDERSON, 2011, p. 15).

A correta aplicação do Kanban faz com que organizações se tornem altamente colaborativas, confiáveis e se aprimorem constantemente, trazendo melhorias na produtividade, qualidade e prazos de entrega. Realizando entregas regulares com alto valor

agregado, há uma maior satisfação por parte do cliente. Existem também evidências de que organizações se tornam mais ágeis com a mudança cultural evolutiva proporcionada pelo Kanban (ANDERSON, 2011, p. 15).

2.6 Conteinerização

Container é uma unidade padrão de software que empacota o código e todas suas dependências por meio de um arquivo configurável para que a aplicação seja executada de forma rápida e confiável, abstraindo possíveis incompatibilidades de sistemas operacionais distintos. Diferentemente das máquinas virtuais que virtualizam o hardware da máquina, os containers virtualizam o sistema operacional, propiciando uma maior portabilidade e eficiência (DOCKER, 2022).

Os containers simplificam o desenvolvimento e a entrega de aplicações distribuídas e se tornam cada vez mais populares à medida que as empresas migram suas aplicações para a nuvem (IBM, 2022).

Para que microsserviços possam ser disponibilizados de forma escalável, é necessária sua encapsulação por meio de containers (IBM, 2022).

Para a conteinerização dos microsserviços desenvolvidos, neste trabalho, foi utilizado o *software* Docker.

É completamente possível criar containers sem o Docker com recursos integrados dos sistemas operacionais. No entanto, o Docker torna a conteinerização mais rápida, fácil e segura. Atualmente, mais de 13 milhões de desenvolvedores utilizam a plataforma (IBM, 2022).

2.7 Reconhecimento de caracteres

Reconhecimento de caracteres ópticos (OCR - Optical Character Recognition) é uma técnica de Visão Computacional que permite ao computador realizar o reconhecimento e processamento de textos contidos em imagens digitais. Existem algumas dificuldades para o reconhecimento exato das letras por meio de sistemas OCR. Letras com grafias semelhantes, textos manuscritos, imagens com baixa qualidade e imagens com baixa taxa de luminosidade tendem a ser dificultadores, durante o processo de extração de texto (SILVA, 2020, p. 24-26).

Neste trabalho, foi utilizada a biblioteca EasyOCR para realizar a detecção de textos das imagens digitalizadas pelos projetos do NIMPI e do MIV.

2.8 Versionamento de Software

O controle de versão provê uma maneira contínua e simples na construção de *softwares* possibilitando que equipes de desenvolvimento rastreiem as alterações realizadas no código fonte. Logo, é possível proteger o código matriz de danos irreparáveis, gerando assim maior liberdade para a equipe de desenvolvimento testar novas funcionalidades e conceitos sem causar danos ou conflitos no código funcional. Caso dois ou mais desenvolvedores criem alterações incompatíveis, ao codificarem simultaneamente, o controle de versão identifica as áreas com problemas para que a equipe possa rapidamente corrigi-las ou reverter o código para uma versão anterior. O controle de versão também é útil para que os desenvolvedores possam analisar versões anteriores do código e entender como uma solução evoluiu (GITLAB, 2022).

2.9 EasyOCR

EasyOCR é uma biblioteca pronta para uso, de fácil implementação, utilizada em conjunto com a linguagem de programação Python para extrair textos de imagens. A biblioteca consegue realizar a leitura de textos tanto por meio de fotos contendo cenas como de documentos de texto puro. Além de oferecer suporte a mais de 80 línguas incluindo o Português, possui mais de oitenta desenvolvedores contribuindo para seu constante aprimoramento e é utilizada por mais de mil e quinhentas pessoas (JAIDEDAI, 2022).

Suas principais vantagens são:

- possui uma comunidade ativa.
- documentação completa e atualizada.
- existência de diversos exemplos de utilização nos motores de busca.

2.10 Pillow

Pillow é uma poderosa biblioteca para Python que possui inúmeras funcionalidades para o tratamento de imagens. Possui suporte a diversos formatos de imagem e tem grande poder de processamento. Pode ser utilizada tanto para o redimensionamento quanto para a manipulação de propriedades e geração de imagens digitais (GEEKSFORGEEKS, 2021).

Vantagens da biblioteca:

- suporte a diversos formatos de imagem.
- exitência de diversas funcionalidades para o tratameto de imagens.

2.11 PDF2Image

PDF2Image é uma biblioteca desenvolvida com foco único a fim de realizar a extração das páginas de um arquivo PDF e transformá-las em objetos de imagem PIL. O projeto possui vinte e cinco contribuidores e é utilizado por mais de quatro mil e quinhentas pessoas (BELVAL, 2022).

Principais vantagens:

- documentação enxuta e atualizada.
- vasta gama de configurações.

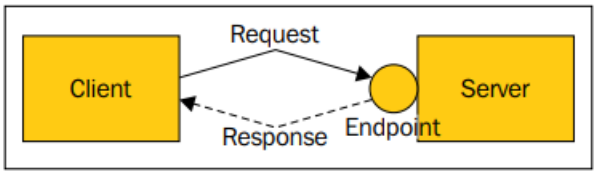
2.12 Mensageria

Mensageria é um conceito de comunicação por meio de troca de mensagens entre dois ou mais sistemas informatizados, permitindo que as aplicações funcionem de forma independente (BRITO, 2019). A troca de mensagens é controlada por um *software* intermediador que é responsável por receber e entregar as mensagens por meio de filas virtuais.

Sistemas de computadores, assim como os seres humanos, possuem a necessidade de trocar mensagens entre si para se comunicarem. Em alguns casos, um sistema informatizado precisa ter a confirmação de que a mensagem enviada chegou até o seu destinatário. Em outras situações, a aplicação precisa receber uma resposta imediata. Também há casos em que o *software* necessite receber mais de uma reposta. Devido à singularidade das necessidades de cada sistema, diferentes formas de comunicação entre aplicações emergiram nos últimos anos (DOSSOT, 2014, p. 7).

A figura 4 demonstra o modelo mais comum de comunicação entre sistemas. Neste formato, as aplicações comunicam-se entre si de forma direta. A aplicação cliente envia uma mensagem para o servidor e aguarda a resposta de forma síncrona. Por mais que, neste modelo, haja uma forma mais simples de se programar devido a sua estrutura procedural, o forte acoplamento das aplicações causa um impacto profundo na arquitetura do sistema em sua totalidade, gerando dificuldades de evolução e de escalabilidade das plataformas (DOSSOT, 2014, p. 8).

Figura 4 – Modelo de comunicação requisição-resposta

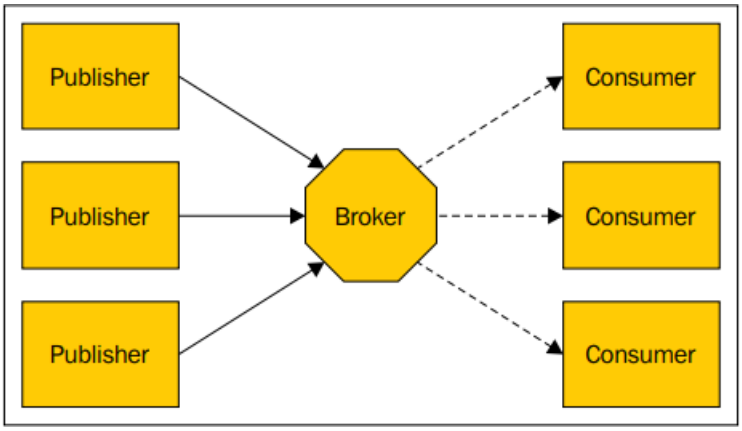


Fonte: Dossot (2014, p. 8)

Por outro lado, na abordagem de mensageria, as aplicações são fracamente acopladas. Nesse formato, não há uma comunicação engessada entre cliente e servidor, fazendo com que os sistemas possam evoluir de forma independente, sem causar impactos correlativos. O controle da recepção e entrega das mensagens são de responsabilidade de um *software* intermediário, também conhecido na área tecnológica como *broker* (DOSSOT, 2014, p. 9).

Na figura 5 podemos visualizar o modelo de comunicação por mensageria.

Figura 5 – Modelo de comunicação mensageria



Fonte: Dossot (2014, p. 9)

A arquitetura apresentada na figura 5 permite que:

- publicadores ou consumidores possam falhar sem afetar um ao outro.
- o desempenho de uma aplicação não afete o de outra.
- o número de publicadores e consumidores cresçam e diminuam de forma independente conforme a carga de trabalho.
- publicadores e consumidores não necessitem conhecer as tecnologias alheias.

2.13 Trabalhos similares

Os seguintes softwares proprietários para tratamento de imagens foram encontrados durante esta pesquisa:

- DocDigital[1]: plataforma para a preservação de documentos. A DocDigital trabalha com a preservação, estruturação e gestão de documentos históricos e antigos realizando a compactação dos arquivos sem que haja a perda na qualidade. A plataforma também possibilita ao cliente a conversão dos documentos para diversos formatos.
- FileStack[2]: plataforma para tratamento de áudio, vídeo, imagens e documentos. A FileStack possui uma API pela qual o usuário pode realizar o redimensionamento de imagens, aplicação de filtros e efeitos bem como a conversão entre formatos de arquivos. Possui bibliotecas para as principais tecnologias utilizadas no mercado, como JavaScript, Ruby, PHP, Python, Swift e Android.
- imgIX[3]: plataforma para tratamento de imagens em tempo real. A imgIX alivia a carga de processamento de sites e aplicativos ao realizar o tratamento de imagens e vídeos sob demanda e em tempo real. A plataforma possibilita o redimensionamento, corte, compressão e manipulação de elementos em arquivos de imagem e vídeo.

[1] Disponível em https://www.docdigital.net.br/
[2] Disponível em https://www.filestack.com/
[3] Disponível em https://imgix.com/

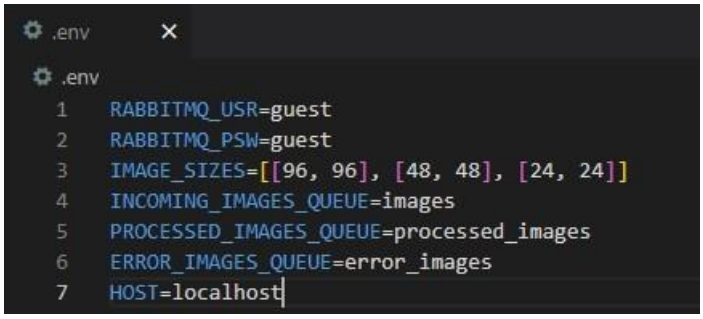
3 Materiais e métodos

3.1 Materiais

Para o desenvolvimento dos microsserviços, deste trabalho, foi utilizada a linguagem de programação Python[1]. A escolha do Python se deu pelo fato de ser uma linguagem extremamente versátil e posuir uma ampla gama de módulos prontos para a utilização.

Para cada serviço foi criado um arquivo de variáveis de ambiente denominado .env-example. Esse arquivo é utilizado pela aplicação para definir as configurações coletadas durante o levantamento de requisitos. Na figura 6 é possível visualizar o arquivo .env-example do serviço de imagens.

Figura 6 – Variáveis de ambiente



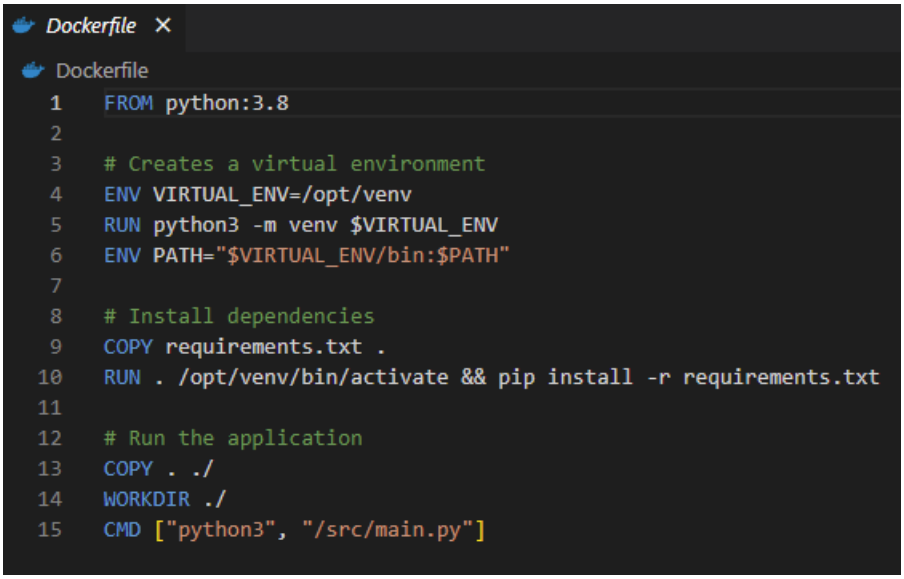
Fonte: Elaborado pelo autor (2022)

Para que os serviços pudessem ser escalados, de acordo com a demanda de processamento, foi utilizada a técnica de conteinerização por meio do *software* de código aberto Docker[2]. Em cada aplicação, criou-se um arquivo de configuração denominado DockerFile. Esse arquivo é responsável por criar uma imagem da aplicação encapsulando todas as funcionalidades e bibliotecas necessárias para o seu funcionamento.

[1] Disponível em https://www.python.org/
[2] Disponível em https://www.docker.com/

A figura 7 demonstra o arquivo DockerFile de um dos microsserviços desenvolvidos.

Figura 7 – DockerFile



Fonte: Elaborado pelo autor (2022)

Ainda sobre o conceito de *container*, houve a criação do arquivo docker-compose.yaml. Esse arquivo armazena as configurações necessárias para a geração de uma imagem completa da aplicação possibilitando também sua execução de forma automática. Neste arquivo, também foi realizada a configuração para a inicialização de uma instância do *software* RabbitMQ caso haja a necessidade por parte do utilizador.

Para a comunicação entre as APIs e os microsserviços desenvolvidos, utilizou-se o conceito de mensageria. O programa selecionado para gerir a troca de mensagens entre as aplicações foi o RabbitMQ.

Para simular um ambiente mais próximo ao de produção durante o processo de desenvolvimento dos serviços, criou-se uma instância do *software* de código aberto RabbitMQ[3] em um *container* Docker.

Para o recebimento, envio das mensagens e comunicação com o RabbitMQ, empregou-se a biblioteca Pika[4].

3.1.1 Microsserviços

Seguindo o levantamento de requisitos, foram desenvolvidos três microsserviços.

O serviço de imagens é responsável por redimensionar as imagens enviadas às plataformas do NIMPI e do MIV, respeitando as resoluções configuradas pelo administrador

[3] Disponível em https://www.rabbitmq.com/
[4] Disponível em https://pika.readthedocs.io/en/stable/intro.html

do sistema no arquivo de variáveis de ambiente do serviço. Durante o processamento da imagem, também é realizada a aplicação de um filtro para a remoção de bordas serrilhadas nas imagens. Para o desenvolvimento deste serviço, empregou-se a biblioteca Pillow[5].

O serviço de PDF é responsável pela conversão de arquivos PDF em imagens digitais. Para cada página existente no documento, o serviço executa extração da página e a converte para o formato JPEG (Joint Photographic Experts Group). Com o objetivo de obter um maior controle sobre a qualidade das imagens extraídas, criou-se uma propriedade no arquivo de variáveis de ambiente a qual pode ser configurada pelo administrador do sistema especificando a resolução da imagem em *pixels*. Para auxiliar na extração das imagens, utilizou-se a biblioteca PDF2Image[6].

O serviço de OCR é responsável por extrair os caracteres de texto contidos nas imagens digitalizadas e enviadas para as plataformas do NIMPI e do MIV. Nesse serviço criou-se uma variável de ambiente pela qual o administrador do sistema pode decidir se a leitura e extração dos textos será feita de forma integral ou por parágrafoss. Para a extração dos textos, foi desenvolvida uma funcionalidade que recebe a imagem a ser processada e faz a leitura dos caracteres existentes na imagem por meio da biblioteca EasyOCR[7].

3.2 Métodos

Nesta seção, serão descritos os métodos utilizados para o desenvolvimento dos microsserviços supracitados.

3.2.1 Kanban

Para o gerenciamento das tarefas realizadas, durante o processo de desenvolvimento deste trabalho, utilizou-se o método Kanban. Para o desenvolvimento dos serviços, foram criados 21 cartões. No título de cada cartão especificou-se a tarefa a ser realizada. Em cartões que possuíam tarefas com maior complexidade, criou-se uma lista de passos a serem realizados para a conclusão da tarefa. Ao final da execução de cada tarefa, descreveram-se as alterações realizadas na aplicação e os materiais utilizados.

A utilização do método Kanban propiciou um grande auxílio na organização e desenvolvimento das tarefas elencadas por meio do levantamento de requisitos.

[5] Disponível em https://pillow.readthedocs.io/en/stable/
[6] Disponível em https://pdf2image.readthedocs.io/en/latest/index.html
[7] Disponível em https://github.com/JaidedAI/EasyOCR

3.2.2 Versionamento

Para o versionamento da aplicação, utilizou-se três repositórios distintos para cada serviço. Criou-se os repositórios do serviço de imagens[8], do serviço de PDF[9] e do serviço de OCR[10] na plataforma do GitLab da Fábrica de Software da instituição. Para cada cartão de tarefa criado no quadro kanban, criou-se uma *branch*(ramificação) da *branch development* na qual foi desenvolvida a funcionalidade, descrita em cada cartão. Após a finalização do desenvolvimento de cada funcionalidade realizou-se um *merge request* - processo de unificação entre duas *branches* em que houve a unificação da *branch* em progresso com a *branch development*.

A utilização de uma ferramenta para versionamento auxiliou no acompanhamento da evolução das funciondalidades da aplicação. A plataforma também possibilitou a rápida detecção de erros existentes em novos códigos inseridos por meio da comparação entre as versões.

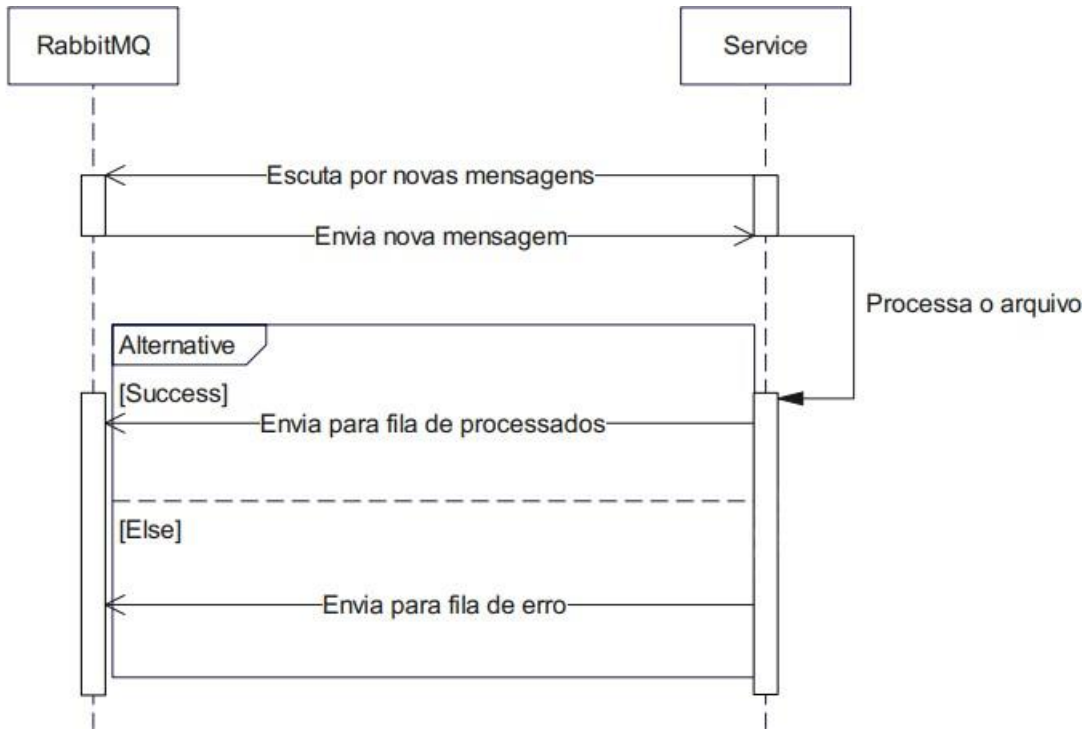
3.3 Modelagem

No decorrer da documentação deste trabalho, elaborou-se alguns diagramas UML. Considerando que todos os serviços possuem a mesma ordem de execução, criou-se apenas um diagrama de sequência o qual demostra a sucessão de procedimentos realizados durante o processamento de uma mensagem recebida.

Na figura 8, podemos observar a sequência de procedimentos realizada pelos serviços desenvolvidos.

[8] Disponível em https://gitlab.fslab.dev/fslab/nimpi-miv/imagem-service
[9] Disponível em https://gitlab.fslab.dev/fslab/nimpi-miv/pdf-service
[10] Disponível em https://gitlab.fslab.dev/fslab/nimpi-miv/ocr-service

Figura 8 – Diagrama de sequência



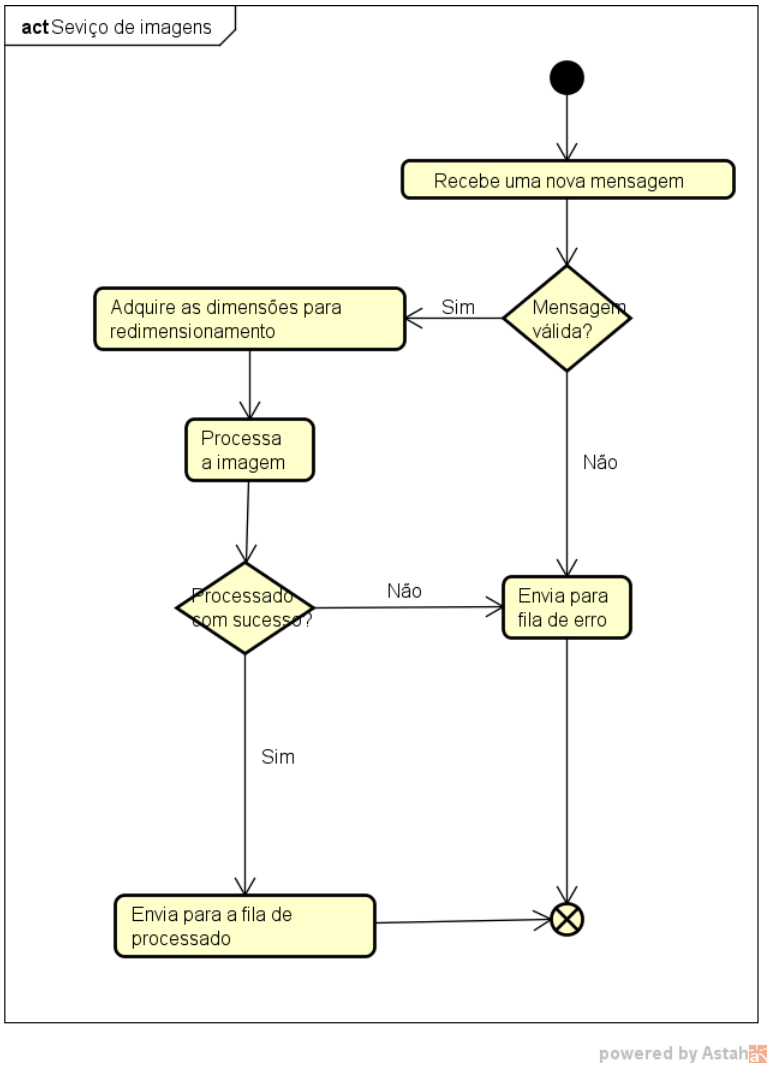
Fonte: Elaborado pelo autor (2022)

Para demonstrar o fluxo de execução dos serviços, elaborou-se dois diagramas de atividade. A figura 9 demonstra o diagrama de atividade do serviço de imagens.

Para os demais serviços, criou-se um único diagrama de atividade, pois ambos possuem a mesma lógica algorítmica.

A imagem 10 demonstra as etapas realizadas durante o processamento das mensagens nos serviços de PDF e OCR.

Figura 9 – Diagrama de atividade - Serviço de imagens

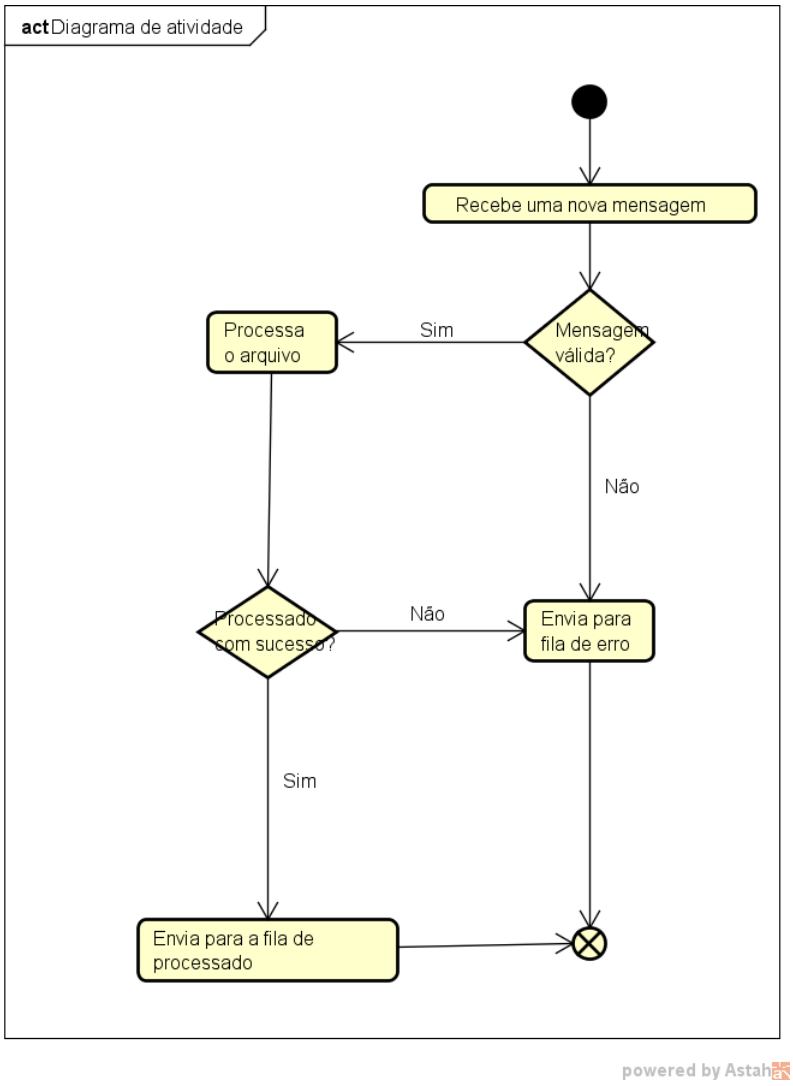


Fonte: Elaborado pelo autor (2022)

3.4 Requisitos

O levantamento de requisitos foi feito, ao final do mês de março, em reunião presencial realizada no Instituto Federal de Educação, Ciência e Tecnologia de Rondônia, Campus Vilhena, juntamente ao *product-owner* do projeto. Durante a reunião também foram definidas as tecnologias, as estruturas dos dados e a arquitetura do sistema. As tabelas 1 e 2 exibem os requisitos levantados.

Figura 10 – Diagrama de atividade - PDF e OCR



Fonte: Elaborado pelo autor (2022)

RF	Descrição
	Os serviços devem possibilitar a configuração de parâmetros de conexão por meio de um arquivo de configuração
	Os serviços devem receber os arquivos por uma fila criada no RabbitMQ
	Os serviços devem enviar os arquivos processados para uma fila no RabbitMQ
	O serviço de imagens deve realizar o redimensionamento de imagens
	O serviço de imagens deve permitir a configuração das resoluções de saída
	O serviço de imagens deve possibilitar o redimensionamento por thumbnails
	O serviço de PDF deve realizar a extração de imagens de todas as páginas
	Os serviço de OCR deve realizar a extração dos textos contidos na imagem recebida
	O serviço de OCR deve permitir configurar se a extração será integral ou por parágrafos

Tabela 1 – Requisitos funcionais

RQ	Descrição
	Os serviços devem ser desenvolvidos na linguagem de programação Python
	Os serviços devem ser desenvolvidos utilizando a metodologia de container

Tabela 2 – Requisitos não funcionais

3.5 Arquitetura de software

A arquitetura da aplicação desenvolvida baseou-se na Arquitetura de Microsserviços. Ao contrário da Arquitetura Monolítica em que a aplicação é construída em torno de um único bloco de execução, a arquitetura de microsserviços tem como conceito a quebra da aplicação em vários módulos e a divisão das responsabilidades. Essa arquitetura trouxe uma maior robustez para a plataforma e possibilitou o processamento de grandes quantidades de arquivos de forma simultânea. Para exemplificar a nova arquitetura, criou-se um diagrama C4Model[11] a nível de container.

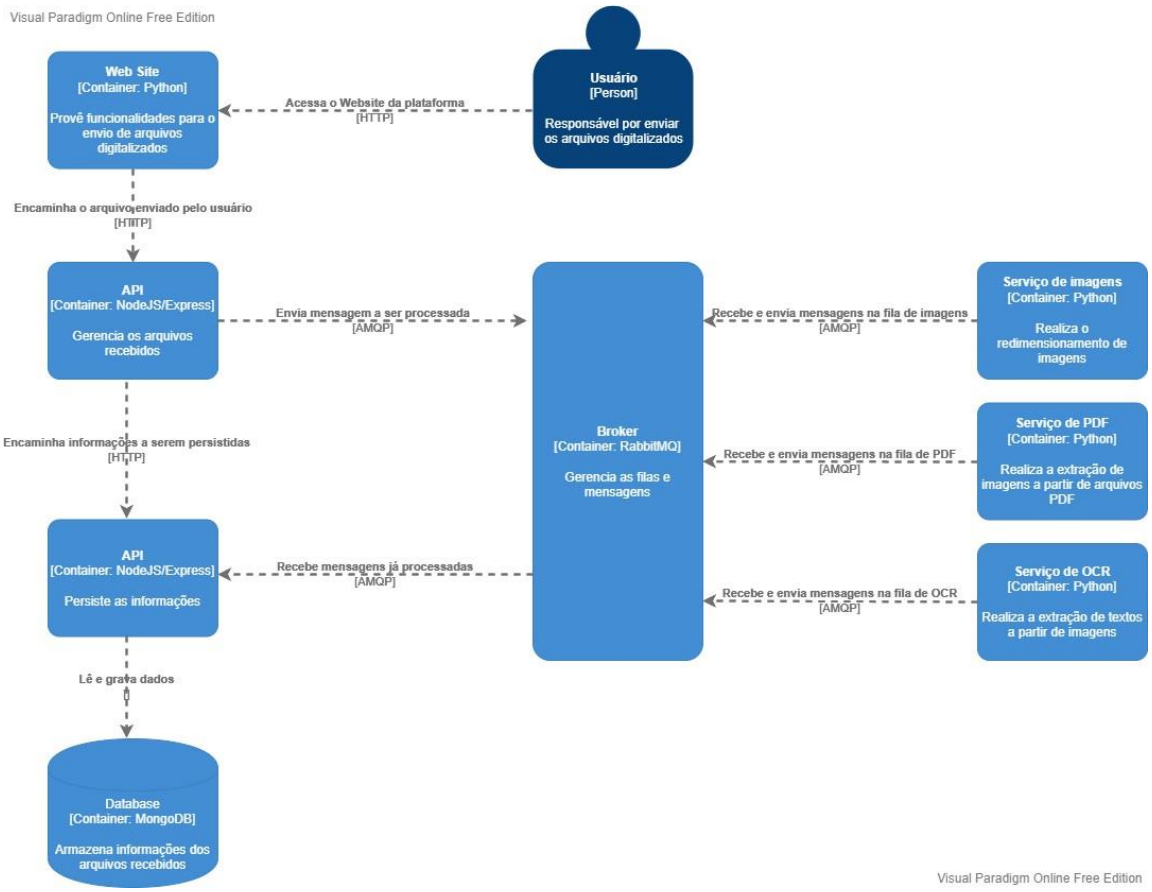
A figura 11 demonstra a arquitetura do sistema.

3.6 Licença de uso

A licença de uso utilizada para o desenvolvimento das aplicações foi a licença desenvolvida pelo Instituto de Tecnologia de Massachusetts MIT[12].

[11] Disponível em https://c4model.com/
[12] Disponível em https://opensource.org/licenses/MIT

Figura 11 – Arquitetura



Fonte: Elaborado pelo autor (2022)

4 Resultados e discussões

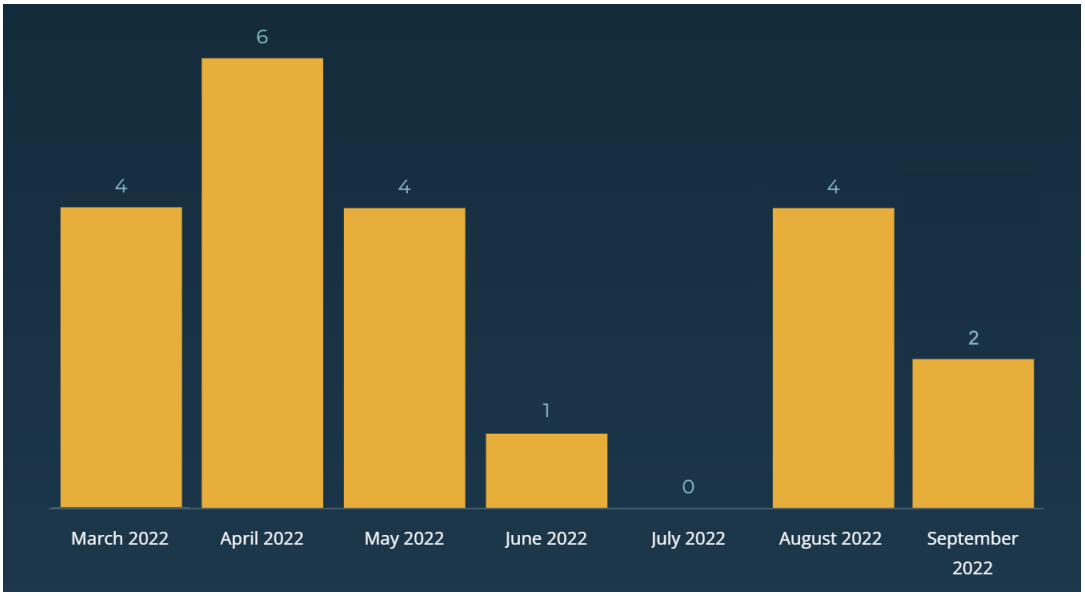
4.1 Gerenciamento de tarefas - Métricas

Para a geração dos gráficos de métricas utilizou-se a plataforma Screenful[1]. Essa plataforma realiza o mapeamento de todas as tarefas criadas no repositório do GitLab e gera os relatórios automaticamente.

4.1.1 Throughput

A métrica *throughput* demonstra a capacidade de entrega das tarefas dentro de um determinado ciclo de tempo. O ciclo, utilizado neste trabalho, para o cálculo da métrica *throughput* foi de 1 mês. A figura 12 exibe o índice de vazão média durante o processo de desenvolvimento de todos os serviços.

Figura 12 – Throughput



Fonte: Elaborado pelo autor (2022)

A iniciação do projeto aconteceu no mês de março, no qual foi realizada uma reunião junto ao Product Owner para definir os requisitos e os prazos necessários à execução das tarefas. No gráfico acima é possível observar uma queda abrupta na resolução das tarefas entre o mês de junho e julho. Essa queda foi motivada por problemas de saúde enfrentados

[1] Disponível em https://screenful.com/gitlab/dashboards-for-gitlab

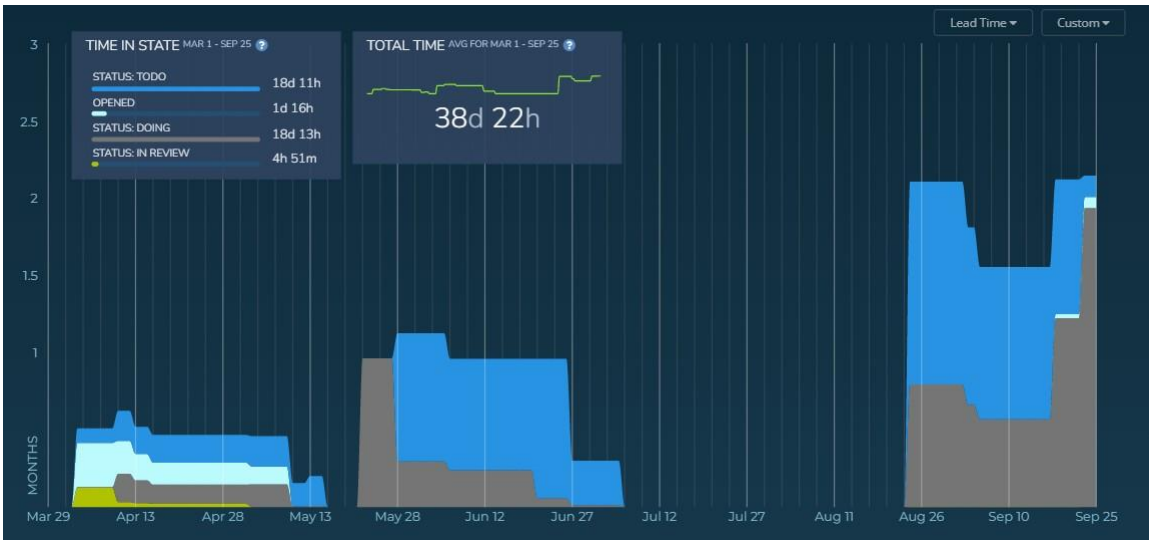
pelo autor da pesquisa, durante o período especificado outrora. A resolução das tarefas restantes foi retomada, no final do mês de agosto.

4.1.2 Lead Time e Cycle Time

Lead time e *cycle time* são métricas utilizadas para medir e aprimorar o desenvolvimento de um projeto. O *lead time* refere-se a diferença entre o momento da criação de uma tarefa e o momento em que ela entra em seu estado final. O *cycle time* é calculado por meio da diferença de tempo em que a tarefa iniciou seu desenvolvimento até a sua finalização.

As figuras 13 e 14 exibem o *lead time* e o *cycle time*, respectivamente, de todo o processo de desenvolvimento.

Figura 13 – Lead Time



Fonte: Elaborado pelo autor (2022)

Com base na figura 13, podemos observar um aumento significativo no *lead time* a partir do mês de maio. Esse acréscimo ocorreu devido às tarefas terem sido criadas no mês de abril e suas resoluções iniciadas no mês seguinte.

Houve também um aumento da métrica supracitada no desenvolvimento do serviço de OCR, acarretando em horas adicionais de estudos. As tarefas desse serviço foram criadas no mês de junho, o que também contribuiu para o crescimento da métrica.

No gráfico exibido pela figura 14, podemos observar que o *cycle time* do desenvolvimento da aplicação foi cinquenta por cento menor comparado ao *lead time*. Além dos motivos supracitados, a organização do tempo reservado para o desenvolvimento da aplicação foi um fator desafiador.

Figura 14 – Cycle Time

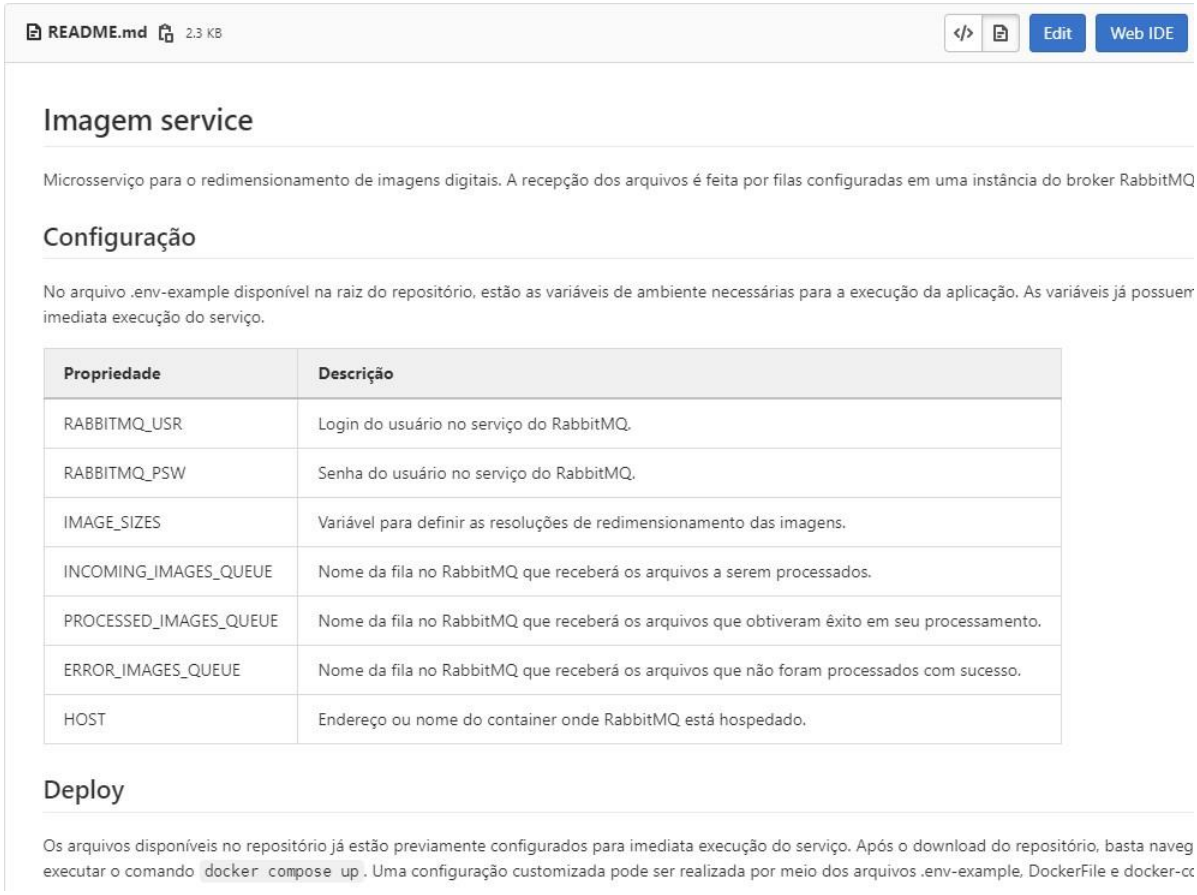
Fonte: Elaborado pelo autor (2022)

4.2 Documentação

Para auxiliar o administrador do sistema durante a configuração dos serviços, criou-se um passo a passo em um arquivo README.md o qual se encontra disponível no repositório de cada aplicação. A criação da documentação foi realizada utilizando a linguagem de *markdown* disponibilizada pelo próprio GitLab na edição de arquivos.

A figura 15 demonstra o arquivo com as instruções de configuração para o serviço de imagens.

Figura 15 – Documentação



README.md 2.3 KB

Edit Web IDE

Imagem service

Microsserviço para o redimensionamento de imagens digitais. A recepção dos arquivos é feita por filas configuradas em uma instância do broker RabbitMQ

Configuração

No arquivo .env-example disponível na raiz do repositório, estão as variáveis de ambiente necessárias para a execução da aplicação. As variáveis já possuem imediata execução do serviço.

Propriedade	Descrição
RABBITMQ_USR	Login do usuário no serviço do RabbitMQ.
RABBITMQ_PSW	Senha do usuário no serviço do RabbitMQ.
IMAGE_SIZES	Variável para definir as resoluções de redimensionamento das imagens.
INCOMING_IMAGES_QUEUE	Nome da fila no RabbitMQ que receberá os arquivos a serem processados.
PROCESSED_IMAGES_QUEUE	Nome da fila no RabbitMQ que receberá os arquivos que obtiveram êxito em seu processamento.
ERROR_IMAGES_QUEUE	Nome da fila no RabbitMQ que receberá os arquivos que não foram processados com sucesso.
HOST	Endereço ou nome do container onde RabbitMQ está hospedado.

Deploy

Os arquivos disponíveis no repositório já estão previamente configurados para imediata execução do serviço. Após o download do repositório, basta naveg executar o comando `docker compose up`. Uma configuração customizada pode ser realizada por meio dos arquivos .env-example, DockerFile e docker-cc

Fonte: Elaborado pelo autor (2022)

4.3 Testes automatizados

Os testes automatizados auxiliaram na obtenção de uma verificação mais veloz da integridade das aplicações durante o processo de desenvolvimento. Sua utilização mitigou o tempo despendido em validações manuais nas principais funcionalidades dos sistemas. O *framework* utilizado para a criação dos programas foi o pytest[2].

A escolha pelo Pytest foi motivada pela agilidade na criação de códigos enxutos e de fácil compreensão. A possibilidade da criação de programas elaborados também foi um fator decisivo para a sua escolha.

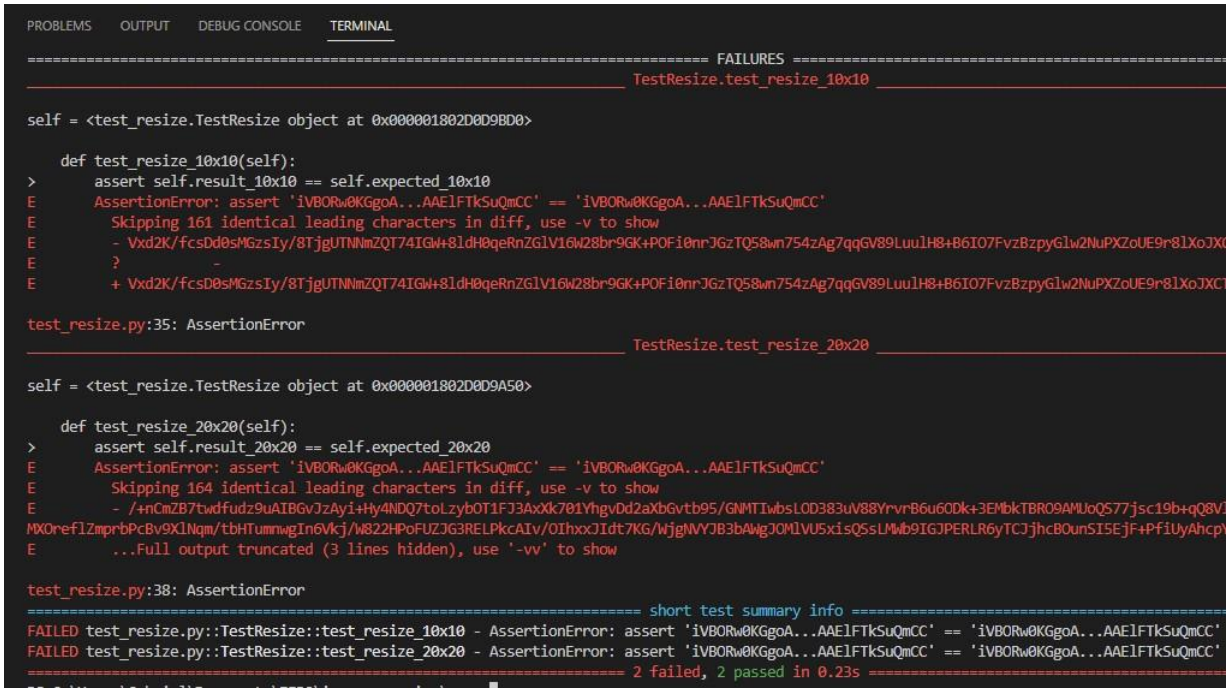
Para a programação, utilizou-se o padrão de classes, seguindo o exemplo disponível na documentação do *framework*. Esse modelo permite que diversos testes sejam executados, simultaneamente, em uma única chamada. A adoção desse padrão sobreveio pelo fato

[2] Disponível em https://docs.pytest.org/en/7.1.x/

do serviço de imagens exigir três testes para sua validação bem como para simplificar o desenvolvimento de novos testes que venham a surgir no futuro.

A figura 16 demonstra as mensagens de erro impressas em console de um teste automatizado que obteve duas falhas de validação.

Figura 16 – Teste automatizado

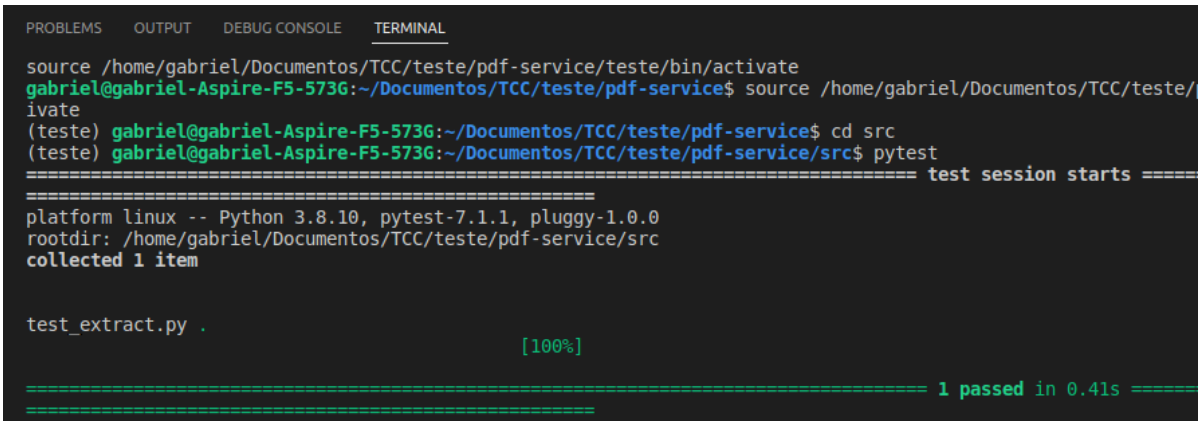


Fonte: Elaborado pelo autor (2022)

Uma das exigências do framework é que os nomes dos arquivos sejam criados com o prefixo test_ para que possam ser detectados durante a chamada. A chamada dos testes é feita com execução do comando *pytest*. Caso o teste seja realizado com sucesso uma mensagem na cor verde é exibida no console da aplicação. Caso falhe, uma mensagem de erro é exibida na cor vermelha.

A figura 17 demonstra a saída de um teste realizado com sucesso.

Figura 17 – Teste automatizado



Fonte: Elaborado pelo autor (2022)

4.4 Tolerância a falhas

Todo e qualquer sistema informatizado pode falhar por diversos motivos em algum ponto de sua execução. A prevenção dessas falhas é essencial para que a aplicação não pare de responder, repentinamente, devido a incompatibilidades não detectadas durante o processo de desenvolvimento. Para mitigar essas falhas catastróficas e evitar comprometer a disponibilidade dos serviços foram implementados blocos para tratamento de exceções durante todo o ciclo de vida dos serviços.

Na figura 18 é possível visualizar a implementação de um bloco de tratamento de exceções criado para a funcionalidade de envio de mensagens. Nesta imagem, a instrução *try* tenta executar o envio de uma mensagem para o serviço do RabbitMQ. Caso haja alguma falha inesperada, durante a comunicação entre os serviços, a instrução *except* previne a falha da aplicação, exibindo o motivo do erro por meio de uma mensagem no console do serviço.

Figura 18 – Tratamento de exceções

```
def SendMessage(message, queue_alias):
    load_dotenv()

    try:
        credentials = pika.PlainCredentials(os.getenv('RABBITMQ_USR'), os.getenv('RABBITMQ_PSW'))
        connection  = pika.BlockingConnection(pika.ConnectionParameters(host = os.getenv('HOST'),
        channel     = connection.channel()

        channel.queue_declare(queue = queue_alias, durable = True)

        channel.basic_publish(
                            exchange    = '',
                            routing_key = queue_alias,
                            body        = message,
                            properties  = pika.BasicProperties(
                                delivery_mode = pika.spec.PERSISTENT_DELIVERY_MODE
                            ))

        connection.close()
        print(" [+] Message sent to: " + queue_alias)
    except AMQPConnectionError as amqp:
        print(" [x] Failed to connect to RabbitMQ: ", repr(amqp))
```

Fonte: Elaborado pelo autor (2022)

4.5 Implantação

Para a implantação dos serviços foi utilizada a técnica de conteinerização por meio do *software* Docker. Devido a dependência dos serviços com outras APIs, desenvolvidas fora do escopo deste trabalho, a implantação foi realizada em um computador pessoal. Por meio do arquivo docker-compose.yaml foi definida a quantidade de *containers* que serão instanciados na inicialização de cada serviço.

Foi criado também um *script* na linguagem Python para simular o envio de mensagens para o RabbbitMQ. Esse *script* carrega uma imagem em memória e, em seguida, armazena-a em um objeto JSON que, posteriormente, é enviado ao *broker*.

4.6 Demonstração do *software*

O envio dos arquivos a serem processados é feito por uma API que não pertence ao escopo de desenvolvimento deste trabalho. Por esse motivo, durante o desenvolvimento dos serviços, criou-se uma funcionalidade para simular o envio. Conforme podemos, observar na figura 19, a mensagem deve conter um objeto JSON com a propriedade *"file"* na qual será armazenada a *string* com as informações do arquivo a ser processado. Essa *string* deve estar codificada em Base64 e UTF-8.

Figura 19 – Envio simulado

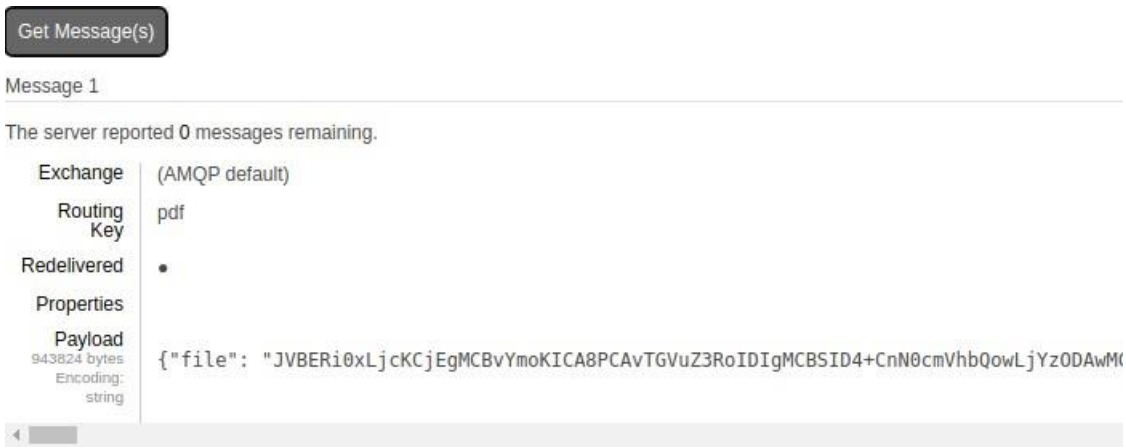
testePDF.py > ...

```python
import base64
import pika

with open("teste.pdf", "rb") as img_file:
    image = base64.b64encode(img_file.read()).decode('utf-8')

message = '{"file": "%s"}' % image

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()

channel.queue_declare(queue='pdf', durable = True)

channel.basic_publish(exchange='',
                      routing_key='pdf',
                      body = message)
print(" Requisição enviada.")

connection.close()
```

Fonte: Elaborado pelo autor (2022)

O RabbitMQ se encarrega de armazenar e enfileirar todas as mensagens recebidas até que sejam coletadas e processadas pelo serviço responsável por seu tratamento e, só então, são removidas da fila correspondente. A figura 20 demonstra a *interface* do RabbitMQ e a mensagem recebida.

Figura 20 – Interface RabbitMQ



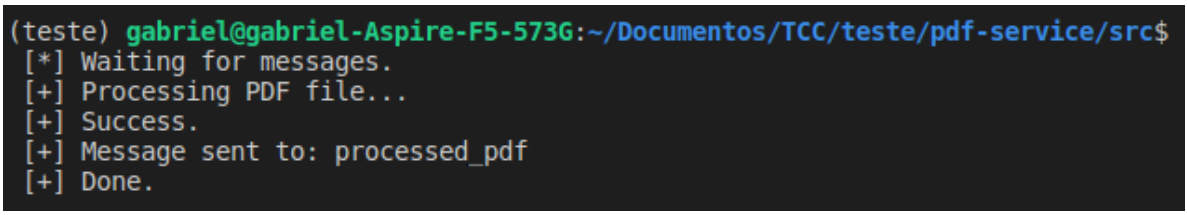
Fonte: Elaborado pelo autor (2022)

Todos os serviços possuem uma conexão contínua com o RabbitMQ e realizam a

escuta por novas mensagens em suas respectivas filas. Assim que uma nova mensagem é recebida pelo *broker*, o serviço responsável pelo aquivo faz a coleta e o processa. Caso a mensagem recebida esteja mal formada ou ocorra algum erro, durante o processamento, a mesma é enviada para uma fila de mensagens com erro. Caso a tarefa seja executada com sucesso, a mensagem é enviada à fila de mensagens processadas.

Na figura 21, podemos observar a sequência de mensagens impressas no console do serviço de PDF, durante o processamento de uma mensagem recebida.

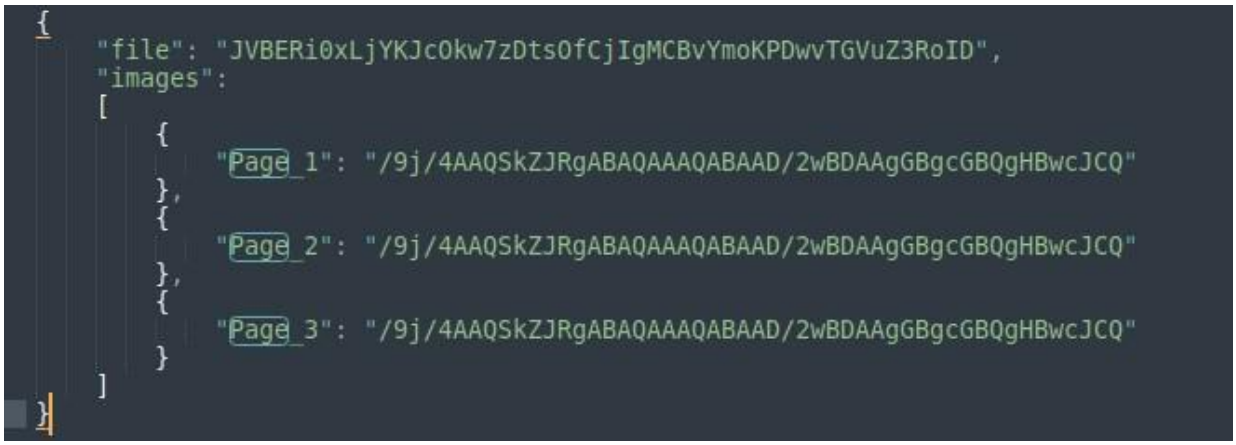
Figura 21 – Mensagens em console - Serviço de PDF



Fonte: Elaborado pelo autor (2022)

Os arquivos processados são alocados em objetos que são anexados posteriormente ao objeto original. Devido à grande quantidade de caracteres presentes nos objetos, não é possível exibir as informações armazenadas de forma completa. Por esse motivo, as *strings* foram limitadas a 50 caracteres para que a estrutura do objeto pudesse ser visualizada, neste exemplo. A imagem 22 demonstra a estrutura dos objetos retornados nas mensagens processadas com sucesso pelo serviço de PDF.

Figura 22 – Mensagem processada - Serviço de PDF



Fonte: Elaborado pelo autor (2022)

As estruturas dos objetos esperados e retornados pelas aplicações podem ser

verificadas no arquivo README.md disponível nos repositórios[3] de cada serviço no GitLab da Fábrica de Software do IFRO.

[3] Disponível em https://gitlab.fslab.dev/

5 Considerações finais

Este trabalho teve como objetivo o desenvolvimento de microsserviços para substituir a arquitetura dos projetos do NIMPI e do MIV. A arquitetura monolítica, presente nas plataformas, apresentava gargalos de processamento devido à crescente demanda de documentos enviados às aplicações. A divisão dos sistemas em microsserviços trouxe uma melhor utilização dos recursos de máquina. O emprego dos conceitos de mensageria e *container* elevaram a capacidade de entrega das plataformas possibilitando o processamento de vários arquivos de forma simultânea.

Esta monografia poderá ser utilizada como base para o aprimoramento dos serviços desenvolvidos para as plataformas supracitadas bem como para a criação de *softwares* semelhantes.

5.1 Trabalhos futuros

Após o desenvolvimento da solução proposta foram identificadas possibilidades aos seguintes trabalhos futuros:

- Desenvolver uma funcionalidade ou serviço para realizar o aprimoramento da qualidade das imagens digitalizadas.
- Criar um serviço que realize a localização dos textos pesquisados e os identifiquem nas imagens.
- Serviço para alertar o usuário sobre arquivos com erro de processamento.

Referências

ANDERSON, D. J. *Kanban - Mudança Evolucionária de Sucesso para Seu Negócio de Tecnologia*. [S.l.]: Blue Hole Press, 2011.

AWS, A. *O que são microsserviços?* 2022. Acessado em: 07/09/2022. Disponível em: https://aws.amazon.com/pt/microservices/.

BELVAL, E. *PDF2Image*. 2022. Acessado em: 08/09/2022. Disponível em: https://github.com/Belval/pdf2image.

BRITO, T. *Mensageria? Message Broker? O que é...* 2019. Acessado em: 17/09/2022. Disponível em: https://medium.com/@devbrito91/mensageria-1330c6032049.

DOCKER. *What is a container?* 2022. Acessado em: 07/09/2022. Disponível em: https://www.docker.com/resources/what-container/.

DOSSOT, D. *RabbitMQ Essentials*. [S.l.]: Packt Publishing Ltd., 2014.

FOWLER, S. J. *Microsserviços prontos para a produção*. [S.l.]: Novatec, 2017.

GEEKSFORGEEKS. *Python: Pillow (a fork of PIL)*. 2021. Acessado em: 08/09/2022. Disponível em: https://www.geeksforgeeks.org/python-pillow-a-fork-of-pil/.

GITLAB. *What is version control?* 2022. Acessado em: 12/09/2022. Disponível em: https://about.gitlab.com/topics/version-control/.

GONZALES, R. C.; WOODS, R. E. *Digital Image Processing*. [S.l.]: Pearson Education, 2007.

GUEDES, G. T. *UML 2 - Uma abordagem prática*. [S.l.]: Novatec, 2011.

IBM. *What is Docker?* 2022. Acessado em: 07/09/2022. Disponível em: https://www.ibm.com/cloud/learn/docker.

JAIDEDAI. *EasyOCR*. 2022. Acessado em: 08/09/2022. Disponível em: https://github.com/JaidedAI/EasyOCR.

MARGARIDA, M. de O. Avaliação da aplicabilidade da uml como uma adl de software. 2018.

MIV. *Museu de Imagem de Vilhena*. 2022. Acesso em: 07/11/2022. Disponível em: https://miv.fslab.dev/sobre.

NIMPI. *Núcleo Informatizado de Memória e Pesquisa do IFRO*. 2018. Acessado em: 09/11/2022. Disponível em: http://nimpi.ifro.edu.br/sobre/.

SILVA, I. do Nascimento Paulo da. Arquitetura de microsserviços para processamento de imagens relevantes em evidências de crimes digitais. 2020.

ZIADé, T. *Python Microservices Development*. [S.l.]: Packt Publishing Ltd., 2017.

ANEXO A – Licença MIT