CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

DAVID A.S.M. RUSYCKI

PROTÓTIPO DE SISTEMA DISTRIBUIDO PARA MONTAGEM DE DOCUMENTOS DIGITAIS

RIO DO SUL
2023

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

DAVID A.S.M. RUSYCKI

PROTÓTIPO DE SISTEMA DISTRIBUIDO PARA MONTAGEM DE DOCUMENTOS DIGITAIS

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí, como condição parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Orientador: M.e. Marciel de Liz Santos

RIO DO SUL
2023

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

DAVID A.S.M. RUSYCKI

PROTÓTIPO DE SISTEMA DISTRIBUIDO PARA MONTAGEM DE DOCUMENTOS DIGITAIS

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí- UNIDAVI, a ser apreciado pela Banca Examinadora, formada por:

_____

Professor Orientador: M.e. Marciel de Liz Santos

Banca Examinadora:

Professor Sandro Alencar Fernandes

_____

Prof.

Professor M.e Fernando Andrade Bastos

_____

Prof.

Rio do Sul, 3 de dezembro de 2023.

A ambição universal do homem é colher o que nunca plantou. (Adam Smith).

Dedico este trabalho a minha família, em especial minha mãe, Miriam, que sempre dispôs seus esforços e fez o possível para que eu pudesse chegar onde estou.

AGRADECIMENTOS

Agradeço a Deus por ter iluminado minha trajetória ao longo dos anos, além de ter me dado forças e uma família tão especial. Agradeço a minha mãe, Miriam, pois sempre fez o possível para me auxiliar nesse percurso, confiando a mim a oportunidade de aproveitar cada passo dessa jornada. Eternamente serei grato e te amarei.

Por fim agradeço a todos os professores que já tive na vida, pois todos foram peças fundamentais na minha formação acadêmica e também como pessoa. Em especial ao professor Marciel de Liz Santos, que aceitou orientar o presente trabalho, permitindo contribuições e troca de informações importantes para a elaboração de um bom trabalho.

RESUMO

Na última década a consolidação da internet e da computação no meio corporativo contribuiu muito para a digitalização de processos empresariais. Tendo isso em vista muitas funcionalidades passaram a ser executadas a partir de meios digitais, processos que envolvem apresentação de documentos, como, admissão de colaboradores e inscrições para bolsas de estudo, foram alguns dos principais afetados por essa digitalização. Porém para algumas pessoas se tornou confuso e até mesmo complexo reunir e entender as informações e documentos que devem compor determinada etapa do processo. Com isso em vista, o presente trabalho propõe a elaboração de um protótipo de aplicação WEB, distribuída, com recursos de observabilidade que permita total controle e segurança sobre o conteúdo dos documentos. Objetivando facilitar aos usuários finais encontrar e entender como os documentos necessários para o processo que ele está inserido devem ser unidos e entregues. A aplicação foi desenvolvida utilizando tecnologias e conceitos modernos. A sua parte visual (*frontend*) foi desenvolvida utilizando HTML, CSS e JavaScript. Somado ao NodeJs e a biblioteca VueJs, que permitiram a criação mais facilitada da interface e seus componentes. A estrutura de microsserviços *backend* foi desenvolvida utilizando Java e o framework Spring Boot, contando com uma estrutura de fila RabbitMq. O sistema ainda conta com serviços de agregação de logs e estatísticas com aplicações dedicadas para tais funcionalidades. A soma dessas tecnologias com o poder das bibliotecas *open-source* utilizadas, permitiram alcançar os objetivos definidos e realizar a criação de uma aplicação robusta e monitorável. Além disso verificou-se também ao fim do trabalho as necessidades futuras de melhorias para o sistema.

Palavras-Chave: Processos, Digitalização, Sistemas de informação.

ABSTRACT

In the last decade, the consolidation of the internet and computing in the corporate environment has contributed greatly to the digitalization of business processes. With this in mind, many functionalities began to be carried out using digital means, processes involving the presentation of documents, such as hiring employees and applying for scholarships, were some of the main ones affected by this digitalization. However, for some people it has become confusing and even complex to gather and understand the information and documents that must make up a certain stage of the process. With this in mind, this work proposes the development of a distributed WEB application prototype, with observability resources that allow total control and security over the content of documents. Aiming to make it easier for end users to find and understand how the documents necessary for the process they are involved in should be put together and delivered. The application was developed using modern technologies and concepts. Its visual part (frontend) was developed using HTML, CSS and JavaScript. Added to NodeJs and the VueJs library, which allowed for easier creation of the interface and its components. The backend microservices structure was developed using Java and the Spring Boot framework, relying on a RabbitMq queue structure. The system also has log and statistics aggregation services with dedicated applications for such functionalities. The sum of these technologies with the power of the open-source libraries used made it possible to achieve the defined objectives and create a robust and monitorable application. Furthermore, this will also be obtained at the end of the work, depending on future needs for improvements to the system.

Keywords: Processes, Digitization, Information Systems.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AMQP	*Advanced Message Queuing Protocol*
API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
DTO	*Data transfer object*
HTML	*Hyper Text Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
ID	*Identifier*
JSON	*JavaScript Object Notation*
OCR	*Optical character recognition*
PDF	*Portable Document Format*
REST	*Representational State Transfer*
RH	*Recursos Humanos*
SAAS	*Software as a service*
SQL	*Structured Query Language*
URL	*Uniform Resource Locator*
UUID	*Universally unique identifier*
WEB	*World Wide Web*

SUMÁRIO

1. INTRODUÇÃO

Na última década a internet e computação se tornaram fundamentais no meio corporativo. De acordo com CGI.br (2022), a quantidade de empresas que usam conexão via fibra óptica avançou de 67% em 2019 para 87% em 2021. O que demonstra um grande crescimento, justificado pela necessidade de presença online e uso de novas tecnologias nas empresas.

Desse modo as entidades passaram a adotar a internet não somente como meio de comunicação e sim como uma extensão e facilitador para o andamento das atividades do negócio. Muitas empresas tornaram algumas funcionalidades antes apenas presenciais, disponíveis agora via internet.

Em vista dessa modernização, processos como inscrições para projetos, programas de bolsas de estudos, processos admissionais, passaram a ser realizados quase que de maneira completa, através de plataformas e meios digitais, geralmente via internet.

Com essas facilidades, no entanto surgiram alguns problemas, como, a maneira de juntar os vários documentos digitais que são necessários nesses processos, através de plataformas que não se pode ter plena confiança. Além disso muitas vezes são documentos com dados sensíveis que não devem cair em posse de pessoas mal-intencionadas.

Ainda temos alguns cenários onde as informações e orientações de como os documentos devem ser montados, ou seja, se existe uma ordem, quais outros documentos devem compor um grande documento, entre outros. Por exemplo, um comprovante de renda de um grupo familiar. Esse documento exige folhas de pagamento de todas as pessoas do grupo familiar, essas folhas virão de vários locais diferentes e, geralmente, uma por uma. Para tanto seria necessário juntar todas elas e depois juntar com outras dos outros integrantes do grupo familiar. Sendo assim encontramos o problema citado anteriormente.

Dessa maneira, é importante uma plataforma WEB que permita as empresas cadastrar, as informações sobre os documentos e como eles devem ser montados, disponibilizando exemplos, tudo isso de maneira centralizada, o que geraria facilidade e segurança para os usuários, além de possibilitar a apresentação de qual o progresso atual que a pessoa já atingiu, fornecendo para ela uma melhor visualização de todo o processo.

1.1 PROBLEMA DE PESQUISA

Como facilitar a busca de informações e a montagem de documentos para inscrições e cadastros através da tecnologia da informação?

1.2 OBJETIVOS

Essa sessão apresenta os objetivos gerais e específicos do desenvolvimento do projeto.

1.2.1 Geral

- Prototipar aplicação WEB, para facilitar e aprimorar o processo de montagem e organização de documentos digitais.

1.2.2 Específicos

- Levantar requisitos Funcionais e Não Funcionais para a aplicação.
- Definir Tecnologias para o desenvolvimento.
- Prototipar aplicação WEB com arquitetura distribuída e monitorável.
- Permitir total controle e segurança sobre os documentos.

1.3 JUSTIFICATIVA

Observando dificuldades nos processos de cadastro para concorrer a bolsas de estudo, que geralmente exigem muitos documentos para comprovação das informações, o lançamento dessas acaba sendo complicado principalmente na montagem de documentos que exigem diversos outros “documentos” como os de comprovação de renda de um grupo familiar por exemplo, que exige folhas de pagamentos de todos os integrantes do grupo. Com várias regras e orientações específicas para diversos casos.

Pensando nisso percebe-se que as informações necessárias para montar a documentação, não ficam centralizadas em um local que também permita a junção desses documentos, assim é necessário recorrer a outras plataformas para realizar essa junção da documentação, onde não se pode ter segurança quanto a privacidade das informações contidas nos documentos. Além de atrapalhar no progresso pois é necessário ficar mudando de ferramentas e anotar o seu progresso em algum lugar para não se perder no meio dos vários documentos que podem ser necessários.

Esse tipo de situação acaba complicando o processo de cadastro e inscrição principalmente para acadêmicos novos que nunca fizeram esse procedimento, é muito comum

dúvidas básicas com questões específicas, o que demanda mais do pessoal responsável por cuidar e tirar dúvidas sobre o cadastro.

Desse modo uma plataforma WEB acessível de maneira facilitada vinculada as informações do processo de inscrição, permitiria centralizar esses dados e a junção confiável e segura de documentos. Podendo guiar o processo em passos a serem cadastrados, reunindo informações úteis além de exemplos de documentos preenchidos. Para facilitar o entendimento e progresso do processo principalmente para pessoas novas.

2. REFERENCIAL TEÓRICO

Neste capítulo será realizada a revisão da literatura abordando os conceitos e tecnologias que serão utilizadas no desenvolvimento.

2.1 ENGENHARIA DE SOFTWARE

Segundo Sommerville (2011, p. 3), “Engenharia de software é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado.”.

Ainda de acordo com Sommerville (2011), os engenheiros de software adotam as abordagens da engenharia de software para sistematizar e organizar o seu trabalho. Sendo uma maneira eficiente de produzir software de alta qualidade. O que é necessário com a alta demanda existente, tem-se ainda a necessidade de sermos capazes de produzir softwares confiáveis de forma econômica e rápida.

2.1.1 Engenharia de Requisitos

Sommerville (2011) indica que os requisitos são a descrição do que um sistema deve fazer. Fornecendo uma visão sobre os recursos e serviços, além das restrições do seu funcionamento. Para ele os requisitos devem refletir as necessidades dos clientes para um sistema, como controlar um dispositivo, colocar um pedido ou encontrar informações. Para Sommerville (2011, p. 57) “O processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado engenharia de requisitos.”.

Os requisitos são geralmente classificados como requisitos funcionais e requisitos não funcionais. Onde requisitos funcionais são declarações de serviços que o sistema deve fornecer, como ele deve reagir a determinadas entradas e como deve ser comportar em dadas situações. Podendo também explicitar o que o sistema não deve fazer. (SOMMERVILLE, 2011).

Já os requisitos não funcionais, são restrições aos serviços ou funções disponibilizadas pela aplicação. Incluindo também restrições de tempo de reposta assim como também restrições relacionadas a normas. Geralmente esse tipo de requisito se aplica a todo o sistema. (SOMMERVILLE, 2011).

2.2 UML

Consoante a Fowler (2011 p. 25), “UML (*Unified Modeling Language*) é uma família de notações gráficas, apoiada por um metamodelo único, que ajuda na descrição e no projeto de sistemas de software, particularmente daqueles construídos utilizando o estilo orientado a objetos (OO).”.

Medeiros (2004) declara que é em resumo uma forma de comunicar ideias, não sendo um processo de desenvolvimento, e sim uma das formas de comunicação que um processo pode utilizar. Paula Filho (2019) diz que em sua versão 2.5 é uma notação longa, que se baseia em um conjunto de quatorze diferentes tipos de diagramas, para as mais diversas representações.

2.2.1 Diagrama de Caso de Uso

Segundo Medeiros (2004), para entender o que é uma Diagrama de Caso de Uso é necessário primeiro entender o que é Ator e Caso de Uso. Nesse cenário um ator pode ser uma pessoa, um sistema ou uma Entidade Externa. Basicamente é alguém ou alguma coisa que interage com outras pessoas ou coisas, para que algo seja realizado.

O autor também cita que um Caso de Uso, pode ser visto como uma representação descrita de variadas ações para a realização de uma macroatividade. Enquanto as atividades menores que compõem a macroatividade tiverem ‘afinidade’ teremos um Caso de Uso. Desse modo o Diagrama de Caso de Uso é composto por Casos de Uso e Atores. Servindo para exemplificar a relação entre eles.

2.2.2 Diagrama de Atividades

De acordo com Paula Filho (2019, p.38):

> Dentro da UML, os diagramas de atividades são as representações mais próximas de alguns diagramas tradicionais, como os fluxogramas e os diagramas de fluxo de dados (DFDs). O elemento básico deles é a atividade, uma especificação de comportamento executável como execução sequencial e concorrente de unidades subordinadas, que podem incluir atividades aninhadas, até chegar às ações, que representam atividades atômicas no nível do modelo. Uma ação recebe um conjunto (possivelmente vazio) de entradas e o converte em um conjunto (possivelmente vazio) de saídas, possivelmente modificando o estado do sistema.

Ainda de acordo com Paula Filho (2019), o diagrama de Atividades pode ser dividido em algumas partições, equivalentes às raias, que podem representar camadas de responsabilidades pelas atividades.

2.2.3 Diagrama de Classes

De acordo com Fowler (2011, p.52):

> Um diagrama de classes descreve os tipos de objetos presentes no sistema e os vários tipos de relacionamentos estáticos existentes entre eles. Os diagramas de classes também mostram as propriedades e as operações de uma classe e as restrições que se aplicam à maneira como os objetos estão conectados.

Para Fowler (2011), os diagramas de classe são a espinha dorsal da UML e, portanto, são muito utilizados.

2.2.4 Diagrama de Pacotes

Fowler (2011, p.96) cita que "Um pacote é uma construção de agrupamento que permite a você pegar qualquer construção na UML e agrupar seus elementos em unidades de nível mais alto. Seu uso mais comum é o agrupamento de classes [...]".

> Em um modelo da UML, cada classe é membro de um único pacote. Os pacotes também podem ser membros de outros pacotes, de modo que você obtém uma estrutura hierárquica na qual os pacotes de nível superior são divididos em subpacotes que possuem seus próprios subpacotes e assim por diante, até que a hierarquia chegue nas classes. (FOWLER, 2011, p.96).

Ainda diz que em termos de programação são equivalentes aos pacotes da linguagem Java. Já nos diagramas, os pacotes, aparecem como uma pasta com guia, onde pode mostrar simplesmente o nome ou também o seu conteúdo. (FOWLER, 2011).

2.3 HTML

De acordo com Miletto (2014, p. 62), "HTML, ou *HyperText Markup Language*, é uma linguagem de marcação utilizada para criar páginas acessadas a partir de um navegador.".

Os elementos que compõem uma página da WEB, são posicionados por meio de recursos disponibilizados pela linguagem, eles são conhecidos como *tags* HTML. Cada *tag* é

uma palavra em inglês específica, envolta por sinais de “menor que” (<) e “maior que” (>). Por exemplo, para adicionar uma imagem na página, utilizaria (<*image*/>). Geralmente, as *tags* aparecem em pares, uma indicando o início e a outra indicando o fim da *tag* de marcação (MILETTO, 2014).

2.4 CSS

Cascading Style Sheets (CSS) ou em português Folhas de Estilo em Cascata, é um recurso da *WEB* que é muito útil no desenvolvimento de páginas, pois possibilitam a reutilização de códigos de formatação visual. Esses que podem ser difundidos em diversos arquivos de um site. Evitando assim a repetição desnecessária de código, que por consequência evita retrabalho. Ao utilizar esse recurso, somente alterando o estilo na folha de estilos irá surtir impacto em todos os lugares que o “referenciam” (ALVES, 2021).

Ressalta ainda que “Podemos considerar que a linguagem de folhas de estilo CSS diz como o conteúdo de um documento HTML deve ser exibido ao usuário, e não o que deve ser mostrado.”. (ALVES, 2021, p.28).

2.5 JAVASCRIPT

Em acordo com Oliveira e Zanetti (2020, p.47):

> JavaScript é uma linguagem de programação orientada a objetos, sendo interpretada e executada pelo navegador *WEB* (*server-side script*). Apresenta uma sintaxe similar à linguagem Java e tem como objetivo principal oferecer melhor interatividade às páginas. Uma característica importante da linguagem JavaScript é que ela não apresenta tipos de dados, isto é, qualquer variável definida é do tipo variante. Isso quer dizer que o tipo de dados é definido de acordo com a informação armazenada naquele momento. O tipo de dados de determinada variável também pode ser modificado ao longo da execução da aplicação, conforme seu conteúdo é alterado.

Posto isso, quanto a origem da linguagem segundo Flanagan (2013, p.19):

> JavaScript foi criada na Netscape na fase inicial da *WEB* e, tecnicamente, “JavaScript” é marca registrada, licenciada pela Sun Microsystems (agora Oracle), usada para descrever a implementação da linguagem pelo Netscape (agora Mozilla). A Netscape enviou a linguagem para a ECMA – *European Computer Manufacturer’s Association* – para padronização e, devido a questões relacionadas à marca registrada, a versão padronizada manteve o nome estranho “ECMAScript”. Pelos mesmos motivos ligados à marca registrada, a versão da Microsoft da linguagem é formalmente conhecida como “JScript”. Na prática, quase todo mundo chama a linguagem de JavaScript.

Flanagan (2013) relata que o JavaScript é utilizado na ampla maioria dos sites modernos e que os mais variados dispositivos incluem interpretadores da linguagem, principalmente em razão dos navegadores atuais, isso o torna “a linguagem de programação mais onipresente da história.” (FLANAGAN, 2013, p.18). O autor ainda considera que o “JavaScript faz parte da tríade de tecnologias que todos os desenvolvedores WEB devem conhecer [...]” (FLANAGAN, 2013, p.18).

2.6 VUEJS

Segundo Incau (2019), um engenheiro que trabalhava no *Google Creative Labs* chamado Evan You sentia a necessidade de aumentar sua produtividade ao prototipar seus projetos sem a repetição de códigos HTML. Com essa situação, Evan começo a verificar as ferramentas existentes no mercado, percebeu que não existia nenhuma biblioteca que permitisse trabalhar sem a repetição de código. Para resolver esse ele criou o VueJs, que inicialmente era uma ferramenta de prototipação rápida e foi evoluindo para o que temos hoje.

“VueJs, ou simplesmente Vue (pronuncia-se *View*), para os mais íntimos é uma lib JavaScript para o desenvolvimento de componentes reativos para WEB. Ele ganhou muita visibilidade no mercado após ser adotado como padrão pelo Laravel (famoso *framework* PHP).”. (INCAU, 2019, p.13).

2.6.1 Componentes reativos

De acordo com Incau (2019, p.16), “Os componentes são fragmentos menores que as páginas. Na verdade, eles servem para compor as páginas. Desse modo, temos um modo fácil e elegante de se reaproveitar código. Isto é o que temos de mais moderno para o desenvolvimento *front-end* hoje em dia”. Ele ainda fala sobre reatividade:

> [...] reatividade é tão simples quanto a definição de componentes. A ideia é que, quando a "informação" de um componente mudar (através do JavaScript), sua marcação (HTML) saiba reagir e se adaptar a isto. Um dos exemplos mais básicos é a adição de um item em uma lista no seu JavaScript. Neste momento, algo ocorre, um evento, então o seu HTML deve saber reagir a isto e adicionar também para a visualização do usuário. (INCAU, 2019, p.16).

Para ele os componentes reativos são partes de código de marcações HMTL, com seu estilo CSS e com o próprio comportamento JavaScript. Eles são semelhantes aos elementos que

já estamos acostumados a ver em páginas da WEB, com a capacidade de ter um comportamento dinâmico e interativo (INCAU, 2019).

2.6.2 O diferencial do vue.js

Segundo Incau (2019), a biblioteca VueJs possui o diferencial de não ser intrusiva que não impõe um padrão de codificação como por exemplo o Angular 2 com TypeScript. Grande parte do código VueJs é escrita em JavaScript puro, tornando sua sintaxe clara e limpa. O VueJs é altamente flexível e também performático, utilizando o conceito de Virtual DOM tendo como seu principal concorrente, o React.

2.7 JAVA

Linguagem de programação Java foi criada em 1991, por um grupo liderado por James Gosling e Patrick Naughton, integrantes da Sun Microsystems, com foco em dispositivos de uso popular, como televisores inteligentes, computadores e celulares. Foi projetado para ser simples com arquitetura que realmente pudesse ser executa por muitos dispositivos. (HORSTMANN, 2009).

Em maio de 1995 a Sun Microsystems apresentou o primeiro Java Development Kit (comumente conhecido como JDK) através da Internet, o que permitiu que desenvolvedores de todo o planeta tivessem como começar a usar a ferramenta. Algo importante da evolução do Java é que ela se transformou em um sistema de desenvolvimento geral, que não ficava restrito a ser usado no ambiente da WEB baseado em navegador, ela pode ser empregada no desenvolvimento de todo tipo de programa. Essa característica notável permitiu que ela se tornasse muito popular em um curto espaço de tempo. (WINDER, 2009).

2.8 SPRING FRAMEWORK

Spring Io (2023) relata que o *Spring framework* é um *framework* para facilitar a criação de aplicativos corporativos, originalmente em linguagem de programação Java. Surgiu em 2003, como uma forma alternativa à complexidade das primeiras especificações do J2EE. O *Spring* integra-se com algumas especificações do J2EE. São elas: API de *servlet*, API WebSocket, Utilitários de simultaneidade, API de ligação JSON, Validação de Bean, JPA, JMS, Coordenação de transações, Injeção de Dependência e também Anotações Comuns.

Conforme ainda indicado por *Spring* Io (2023), o *Spring* f*ramework* é baseado e dividido em módulos. Onde os aplicativos podem escolher quais os módulos que são necessários. Existem alguns módulos centrais como o de container, que inclui um modelo de configuração e opções de injeção de dependência. O *framework* oferece ainda suporte para diferentes arquiteturas, incluindo mensagens, transações de dados, persistência e WEB.

2. 8. 1. Spring Boot

O *Spring* Boot de acordo com *Spring* Io (2023), é um facilitador para criação de aplicações baseadas em *Spring framework*. Semelhante a um projeto pre configurado, o *Spring* Boot possui algumas características, como: Criação de aplicações *Spring* Independentes, Servidores como Tomcat, Jetty ou Undertow diretamente integrados (ou seja sem a necessidade de implantar arquivos), Configurações automáticas de bibliotecas *Spring*, Recursos de métricas e monitoramento para uso em produção.

2.9 BANCO DE DADOS

Segundo Alves (2014, p.16), “Um banco de dados é um conjunto lógico e ordenado de dados que possuem algum significado, e não uma coleção aleatória sem um fim ou objetivo específico.”. Ainda cita como característica que “Um banco de dados é construído e povoado com dados que têm um determinado objetivo, com usuários e aplicações desenvolvidas para manipulá-los.” (ALVES, 2014, p.16).

Ele conclui que “Para se ter um banco de dados, são necessários três ingredientes: uma fonte de informação, da qual os dados são derivados; uma interação com o mundo real e um público que demonstra interesse nos dados contidos no banco.” (ALVES, 2014, p.16).

2.9.1 Sgbd

Alves (2014, p.16) define SGBD como:

> Sistema de Gerenciamento de Banco de Dados (SGBD) é uma coleção de ferramentas e programas que permitem aos usuários a criação e manutenção do próprio banco de dados. Desta forma o SGBD pode ser considerado como um sofisticado *software* destinado à definição, construção e manipulação.

Ele acrescenta dizendo que “Um SGBD deve ainda permitir que bancos de dados sejam excluídos ou que sua estrutura seja alterada, como adição de novas tabelas/arquivos.”.

2.9.2 PostgreSQL

De acordo com PostgreSQL (2023), O PostgreSQL é um poderoso sistema de banco de dados de código aberto, que utiliza o modelo objeto-relacional, estende a linguagem SQL e dispõe uma combinação de muitos recursos, que são capazes de armazenar e dimensionar com segurança cargas de trabalho de dados mais complicadas.

A aplicação surgiu em 1986 como parte do projeto POSTGRES da Universidade da Califórnia em Berkeley. Atualmente tem mais de 35 anos de desenvolvimento ativo na principal plataforma. Durante os anos conquistou uma forte reputação por sua arquitetura comprovada, confiabilidade, integridade de dados e robusto conjunto de recursos. Além disso pode ser executado em todos os principais sistemas operacionais, e é compatível com *ACID* desde de 2001. (POSTGRESQL, 2023).

2.10 DOCKER

Segundo Romero (2015), “[...] Docker é uma ferramenta para criar e manter containers, ou seja, ele é responsável por armazenar vários serviços de forma isolada do SO host, como: WEB *server*, banco de dados, aplicação, *memcached* etc.”. Romero (2015) ainda relata que o Docker é uma alternativa para virtualização. Sendo completa e muito mais leve do que a virtualização que usa *hypervisors* KVM, Xen e VMware ESXi.

2.10.1 Containers

Containers são pacotes de software que são incorporadas em unidades padronizadas para desenvolvimento, envio e implantação. (DOCKER, 2023).

Um contêiner é uma unidade padrão de *software* que agrupa o código e todas as suas dependências. Permitindo assim que o aplicativo seja executado de forma rápida e confiável de um ambiente de computação para outro. Uma imagem de contêiner do Docker é um pacote de *software* leve, autônomo e executável que inclui tudo o que é necessário para executar um aplicativo: código, tempo de execução, ferramentas do sistema, bibliotecas do sistema e configurações. (DOCKER, 2023).

2.11 GIT

De acordo com Git (2023, n.p.), “O Git é um sistema de controle de versão distribuído gratuito e de código aberto projetado para lidar com tudo, desde projetos pequenos a muito grandes com velocidade e eficiência.”.

Ele ocupa pouco espaço com desempenho ultrarrápido, supera as ferramentas SCM como *Subversion*, CVS, *Perforce* e ClearCase com recursos como ramificação local barata, áreas de preparação convenientes e vários fluxos de trabalho. (GIT, 2023).

2.12 MAVEN

De acordo com Maven (2023), é uma ferramenta que pode ser usada para construir e gerenciar projetos baseados em Java. Tem como objetivo facilitar o processo de construção, fornecer um sistema de construção uniforme e incentivar as melhores práticas de desenvolvimento.

O projeto começou como uma tentativa de facilitar os processos de construção do Jakarta *Turbine*. Onde existiam vários projetos, cada um com seus próprios arquivos de construção Ant, onde eram todos ligeiramente diferentes. Buscava-se uma maneira padrão de construir os projetos, com uma definição clara do que consistia em o projeto, tornando mais fácil de publicar informações do projeto, além compartilhar as dependências entre vários projetos. (MAVEN, 2023).

2.13 PDFBOX

De acordo com PDFBox (2023), a biblioteca Apache PDFBox é uma ferramenta Java de código aberto para trabalhar com documentos PDF. Permite a criação de documento assim como manipulação de documentos já existentes. Também possui a capacidade de extrair conteúdo dos documentos. Ainda conta com vários utilitários de linha de comando. A ferramenta é capaz de dividir, mesclar, preencher formulários, validar, imprimir, transformar os arquivos em imagem e assinar digitalmente documentos PDF.

2.14 PROTOCOLO HTTP

De acordo com Behrouz (2010, p. 609):

> O HTTP (*Hypertext Transfer Protocol*) é um protocolo usado principalmente para acessar dados na *World Wide Web*. O HTTP funciona como uma combinação de FTP e SMTP. Ele é semelhante ao FTP porque transfere arquivos e usa os serviços do TCP. Entretanto, é muito mais simples do que o FTP, pois usa apenas uma conexão TCP. Não há nenhuma conexão de controle separada; somente dados são transferidos entre o cliente e o servidor. O HTTP é como o SMTP, porque os dados transferidos entre o cliente e o servidor parecem mensagens SMTP. Além disso, o formato das mensagens é controlado por cabeçalhos como o MIME. Ao contrário do SMTP, as mensagens HTTP não se destinam a serem lidas por seres humanos; elas são lidas e interpretadas pelo servidor HTTP e pelo cliente HTTP (*browser*). As mensagens SMTP são armazenadas e encaminhadas, mas as mensagens HTTP são transmitidas imediatamente. Os comandos do cliente ao servidor são incorporados em uma mensagem de pedido. O Conteúdo do arquivo solicitado ou de outras informações é incorporado em uma mensagem de resposta. O HTTP usa os serviços do TCP na porta conhecida 80.

Consoante a isso MDN (2023), diz que o HTTP é a base de qualquer troca de dados na WEB, sendo um protocolo cliente-servidor, assim as requisições são iniciadas pelo destinatário, comumente utilizando um navegar da WEB. MDN Ainda cita alguns dos principais métodos HTTP, que podem ser observados no Quadro 1.

Quadro 1- Métodos HTTP

Benefício	Descrição
DELETE	Esse método deve ser utilizado para requisições que tem por objetivo remover recursos.
GET	Esse método deve ser utilizado para requisições que tem por objetivo solicitar recursos, ou seja recuperar dados.
POST	Esse método deve ser utilizado para requisições que tem por objetivo gerar alterações nos recursos do servidor. Seja inserindo ou alterando dados. Pois permite passar um corpo de requisição personalizado.
PUT	Esse método deve ser utilizado para requisições que tem por objetivo criar ou substituir uma representação do recurso de destino com os novos dados.

Fonte: Elaborado a partir de MDN (2023).

2.15 API

Interface de programação de aplicação (API) substancialmente define as regras que devem ser seguidas para se comunicar com outros sistemas de software. As APIs são expostas ou criadas por desenvolvedores para que outras aplicações possam se comunicar programaticamente e utilizar os recursos disponibilizados pela sua aplicação. (AWS, 2023).

As APIs podem ser entendidas como um contrato entre um provedor e um usuário de informações, estabelecendo o conteúdo exigido pelo consumidor e o conteúdo exigido pelo

produtor. Elas são um mediador entre usuários e clientes com os recursos ou serviços WEB que eles desejam obter. (RED HAT, 2023).

2.15.1 Rest

Para AWS (2023), o *representational state transfer* (transferência de estado representacional) REST é uma arquitetura de software que define algumas condições sobre como uma API deve funcionar. O objetivo da sua criação foi a definição de uma diretriz para gerenciar a comunicação em uma rede complexa como a internet. Serviços da WEB que implementam e seguem as definições da arquitetura REST são conhecidos como serviços RESTFul.

Para a RED HAT (2023), a criação do cientista da computação Roy Fielding trata-se de um conjunto de restrições de arquitetura que os desenvolvedores podem implementar de maneiras variadas. A fim de atender o conjunto de diretrizes que tornam o REST mais rápido, leve e escalável que protocolos prescritos como o Protocolo Simples de Acesso a objetos (SOAP). Como consequência APIs RESTFul precisam utilizar os métodos HTTP que puderam ser vistos no Quadro 1.

2.16 MICROSSERVICOS

De acordo com a AWS (2023), Microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de softwares. Basicamente consiste em pequenos serviços independentes que se comunicam usando API’s. A arquitetura de microsserviços facilita a escalabilidade das aplicações, e agiliza o desenvolvimento das aplicações, reduzindo também o tempo para introdução de novos recursos.

2.16.1 Características

Os microsserviços diferente de projetos monolíticos possuem algumas características que os diferenciam. Nesse modelo, os serviços são autônomos, ou seja, podem ser desenvolvidos, implantados, operados e escalados sem afetar o funcionamento dos outros serviços. Eles não precisam compartilhar nenhum código ou implementação com outros serviços. Sendo assim todas as comunicações entre os componentes individuais ocorrem por meio de API’s bem definidas. (AWS, 2023).

Outra característica destacada pela AWS (2023), é que esses serviços são especializados, ou seja, são projetados para ter um conjunto de recursos voltados para a solução de um problema específico. Sendo assim com o passar do tempo e quando necessário os desenvolvedores podem acrescentarem mais código aos serviços, e de acordo com a complexidade ele poderá ser dividido em mais serviços menores.

2.16.2 Benefícios

Conforme abordado pela AWS (2023), existem alguns benefícios da arquitetura de microsserviços. Eles podem ser observados no Quadro 2.

Quadro 2 - Benefícios dos microsserviços

Benefício	Descrição
Agilidade	Os microsserviços permitem uma organização de equipes geralmente pequenas e independentes que são proprietárias de seus serviços. Elas atuam dentro de um contexto pequeno e claramente compreendido tendo autonomia para trabalhar de forma mais independente e rápida. Tendo como resultado aceleração dos ciclos de desenvolvimento.
Escalabilidade flexível	Essa abordagem permite que cada serviço seja escalado de forma independente para atender à demanda de recursos que a aplicação exige. Isso permite melhor dimensionamento das necessidades de infraestrutura.
Fácil implantação	Os microsserviços permitem a integração e a entrega contínuas, o que facilita o teste de novas ideias e sua reversão caso algo não funcione adequadamente. O custo mais baixo de falha permite a experimentação e facilita a atualização do código fonte, acelerando também o tempo de introdução de novos recursos no mercado.
Liberdade tecnológica	A arquitetura de microsserviços não segue uma abordagem padrão. Portanto as equipes ficam livres para escolher a melhor ferramenta para resolver os problemas.
Código reutilizável	A divisão do software em pequenos serviços bem definidos permite a utilização de funções para várias finalidades. Um serviço para uma determinada função pode ser usado como componente básico para outro recurso. Assim aplicativos são reutilizados pois são desenvolvidos novos recursos sem precisar escrever código.
Resiliência	A independência do serviço aumenta a resistência a falhas do aplicativo. Os microsserviços não geram falha total dos serviços. Sendo assim apenas partes da aplicação param de funcionar.

Fonte: Elaborado a partir de AWS (2023).

2.17 ECLIPSE

De acordo com a ECLIPSE FOUNDATION (2023), O Eclipse IDE é um ambiente de desenvolvimento integrado (*Integrated Development Environment*) para a linguagem de programação Java. Atualmente o Eclipse é mantido pela Função Eclipse, que vem fornecendo

para a comunidade global um ambiente maduro, escalável e amigável. Focado em software de código aberto, atualmente a fundação é responsável por mais de 415 projetos de código aberto, que incluem ferramentas e estruturas para uma ampla gama de domínios de tecnologia, como internet das coisas, automotivo, geoespacial, engenharia de sistema e muitos outros.

2.18 RABBITMQ

De acordo com RABBITMQ (2023), a ferramenta é um dos agentes de mensagem de código aberto mais populares. Sendo muito utilizado em pequenas *startups* e grandes empresas. Podendo ser implantado em configurações distribuídas e federadas para atender a requisitos de alta escala e alta disponibilidade. Existe uma ampla variedade de ferramentas de desenvolvedor para diversas linguagens populares. As principais caraterísticas do RabbitMq, podem ser vistas no Quadro 3.

Quadro 3 – Características do RABBITMQ

Característica	Descrição
Mensagens assíncronas	Suporta diversos protocolos de mensagens, enfileiramento das mesmas, confirmação de entrega, roteamento flexível para filas e vários tipos de troca.
Experiência do desenvolvedor	Implantação com Kubernetes, BOSH, Chef, Docker e Puppet. Além do desenvolvimento de mensagens entre linguagens com as principais linguagens do mercado.
Implantação Distribuída	Suporte a implantação como clusters para alta disponibilidade e rendimento, também a possibilidade de federar em diversas zonas e regiões de disponibilidade.
Pronto para empresa e nuvem	Oferece autenticação conectável, autorização, suporte a TLS e LDAP. Além de ser leve e fácil de implantar em nuvens públicas e privadas.
Ferramentas e plug-ins	Possui um conjunto diversificado de ferramentas e plug-ins, que oferecem suporte à integração contínua, métricas operacionais e integração com outros sistemas corporativos.
Gestão e Monitoramento	Disponibiliza uma API HTTP para consulta de métricas, além de ferramentas de linha de comando e UI para gerenciar e monitorar.

Fonte: Elaborado a partir de RABBITMQ (2023).

2.19 GRAFANA

De acordo com GRAFANA (2023), a ferramenta Grafana permite, coleta, correlação e visualização fácil de dados através de belos painéis (*Dashboards*). Trata-se de uma solução de visualização e monitoramento de dados de código aberto que orienta decisões através da informação, melhora o desempenho do sistema e agiliza a solução de problemas.

Conforme abordado por GRAFANA (2023), os dashboards podem ser montados a fim de evitar a unificação de bancos de dados, ou seja, eles adotam uma abordagem de fornecer um

único painel para visualização mais objetiva das informações. Ainda conta com recursos de compartilhamento onde os painéis podem ser vistos por qualquer pessoa.

A plataforma permite também configuração de alertas, a fim de possibilitar mais rapidamente a correção de problemas, o que possibilita transformar os dados em ideias em tempo real. Além disso conta com soluções como o *Grafana Cloud* que é uma plataforma de observabilidade aberta e combinável. Ou seja, as métricas de diversas outras ferramentas externas podem ser direcionadas para um *dashboard* no grafana. (GRAFANA, 2023).

2.20 GRAFANA LOKI

De acordo com GRAFANA LABS (2023), o Grafana Loki é um sistema de agregação de logs multilocatário, horizontalmente escalável e altamente disponível, inspirado no Prometheus. Os seus ideais foram desde de sua concepção a economia e a facilidade de operar. A ferramenta não indexa o conteúdo dos logs em si, mas sim um conjunto de metadados para cada fluxo de logs. Isso acaba por implicar que o Loki requer muito menos armazenamento do que outras soluções parecidas. Além de que os registros estarão disponíveis para consulta em milissegundos após serem recebidos pelo Loki.

O Loki é um projeto de código aberto, que é impulsionado pela comunidade e nasceu dentro do Grafana Labs em 2018. Sob a licença AGPLv3. O uso do Loki está disponível em 3 principais versões. A primeira é a tradicional onde é possível baixar, configurar e executar a aplicação em uma infraestrutura. Já a segunda disponibiliza registros em núvem. E a terceira é uma solução de registros empresariais autogerenciada que funciona com segurança em escala com suporte especializado do Grafana Labs. (GRAFANA LABS, 2023).

2.21 PROMETHEUS

De acordo com PROMETHEUS (2023), trata-se de um kit de ferramentas para geração de alerta e monitoramento de sistemas. Originalmente foi desenvolvido no SoundCloud, e desde a sua criação em 2012, muitas empresas e organizações adotaram o Prometheus, o projeto tem uma comunidade de desenvolvedores e usuário muito ativa. Atualmente é um projeto de código aberto mantido de forma independente por qualquer empesa.

A aplicação coleta e armazena suas métricas como dados de séries temporais, ou seja, as informações das métricas são armazenadas com carimbo de data e hora e em que foram registradas, juntamente com os pares de valores-chave opcionais chamados rótulos. As métricas armazenadas podem variar de aplicação para aplicação, ou seja, em um servidor WEB teremos

métricas que para um banco de dados não fazem sentido. Além disso os dados armazenados podem ser consultados a partir de uma linguagem proprietária conhecida como PromQL. (PROMETHEUS, 2023).

3. METODOLOGIA DA PESQUISA

O presente trabalho de conclusão de curso caracteriza-se como pesquisa aplicada e descritiva, pois tem como objetivo o desenvolvimento de um protótipo para facilitar a preparação de documentação para processos de inscrição de projetos e programas através da tecnologia da informação.

Após a finalização do levantamento bibliográfico, foi realizado o desenvolvimento do protótipo, buscando por utilizar meios e padrões de desenvolvimento modernos para sua realização.

Com base nos conhecimentos e experiências do autor foi optada pela IDE Spring *Tool Suite* e VSCODE, utilizando Spring framework em linguagem Java para desenvolvimento das API's *Back-end*. E VueJs para o desenvolvimento do *Front-end* em linguagem JavaScript. Utilizando boas práticas de desenvolvimento de software em camadas. Como solução de armazenamento de informações foi optado pelo PostgresSql em razão da sua larga utilização e comunidade. O fluxo do desenvolvimento pode ser observado na Figura 1.

Figura 1 – Fluxograma desenvolvimento

Fonte: Acervo do autor (2023).

3.1 ESTADO DA ARTE

Esta seção apresentará ferramentas que já existem no mercado e que possuem objetivos semelhantes ao projeto proposto pelo presente trabalho. Em conjunto será disponibilizado uma listagem comparativa de vantagens e desvantagens do projeto proposto em relação as ferramentas já existentes.

3.1.1 Pipefy

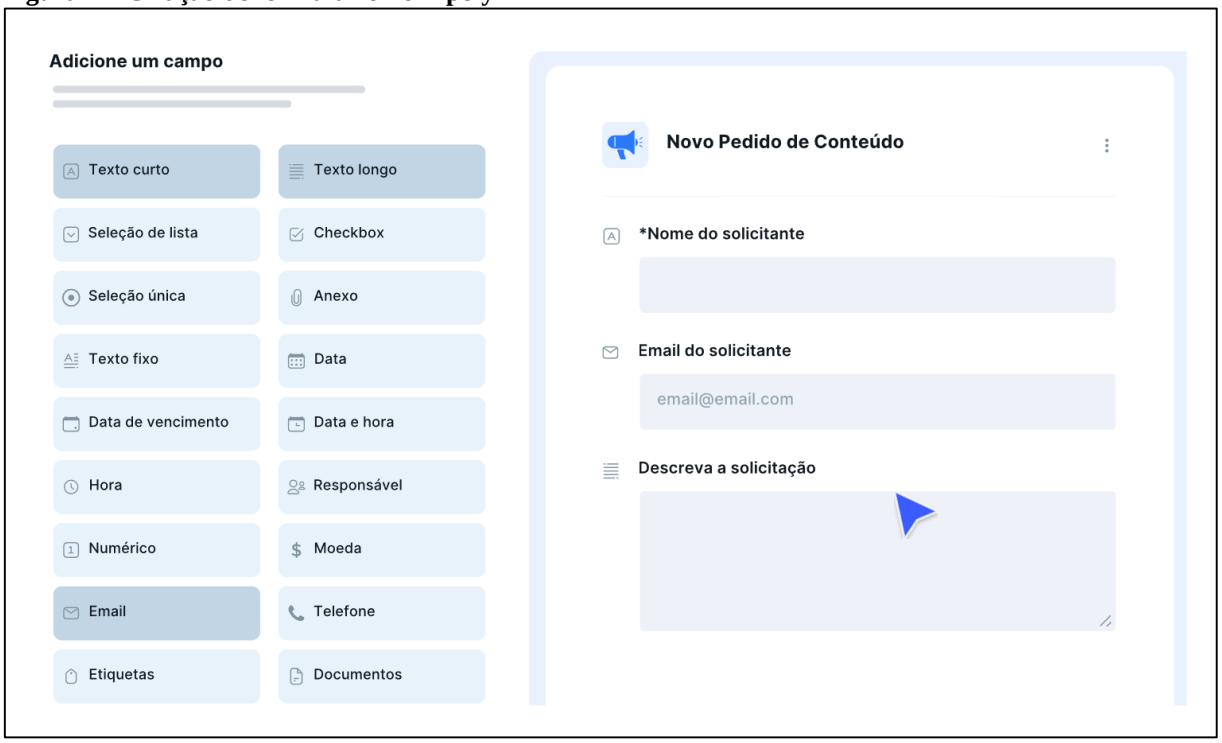
De acordo com Pipefy (2023) é uma ferramenta que permite implementar com rapidez novos fluxos de trabalho e aperfeiçoar constantemente as entregas da empresa. Disponibilizando funcionalidades desde a criação da demanda a execução e relatórios para análise de resultados, permitindo aumentar a eficiência em processos como gestão de RH, compras e *onboarding* de clientes.

Um dos recursos disponibilizados pela ferramenta é possibilidade de criar formulários para envio de documentos. Uma funcionalidade muito semelhante a proposta no presente trabalho. A interface para adição de campos em um formulário pode ser vista na Figura 2. Onde é possível adicionar campos de vários tipos para as informações necessárias. Desse modo podem ser adicionados vários campos para adicionar os documentos por exemplo.

Como vantagem do Pipefy existe a possibilidade da utilização dos outros recursos disponíveis e integrados a ferramenta. Como a linha de CRM, gerenciamento de compras e venda. Portanto é uma solução mais robusta em outros sentidos.

Como desvantagem está a questão de que um usuário ao responder um formulário gerado na plataforma, ainda teria que recorrer a recursos terceiros para realizar junção dos documentos PDF. Já que o Pipefy não disponibiliza esse recurso. Além disso ainda, o armazenamento dos documentos se dará na estrutura do Pipefy. Desse modo não é possível ter total controle sobre a segurança dos documentos.

Figura 2 – Criação de formulário no Pipefy



Fonte: Pipefy (2023).

3.1.2 LugarH

De acordo com LugarH (2023) trata-se de uma ferramenta que pretende otimizar o tempo no dia a dia dos funcionários de RH, para que os mesmos possam focar no trabalho que realmente faz a diferença. Sendo uma ferramenta focada em processos de admissão digital em empresas. Tornando o setor de RH mais rápido, prático e online.

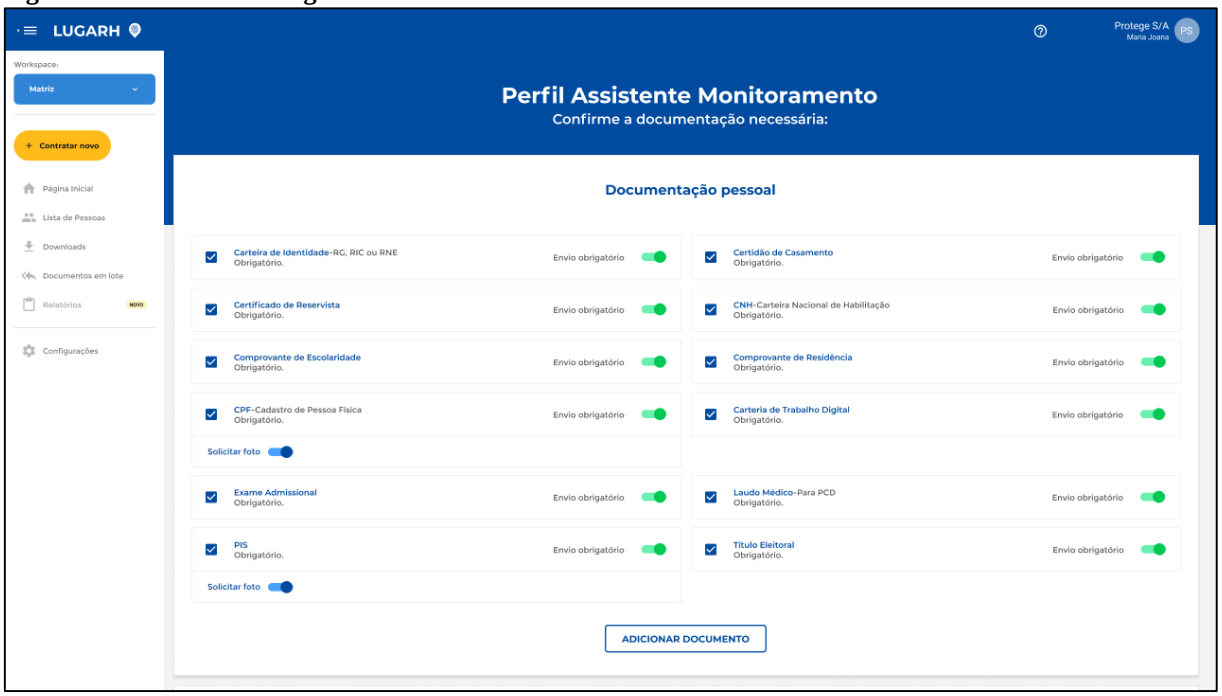
A ferramenta possui um sistema de envio de documentos bastante intuitivo, conforme proposto pelo trabalho, também pensado em ser uma espécie de passo a passo para que o usuário final tenha mais facilidade em realizar todo o procedimento. Na Figura 3, é possível observar a tela de configurações de documentação pessoal necessária para o processo de admissão da empresa. Nela é possível definir quais documentos são obrigatórios no processo além de solicitar fotos do documento ou pessoa por exemplo.

As vantagens que a ferramenta possui, assim como no Pipefy, é o conjunto extra de funcionalidades relacionadas que a mesma possui. Disponibilizando de assinatura digital, recursos de OCR, inteligência artificial, etc.

As desvantagens também estão relacionadas ao controle do conteúdo dos arquivos, que será realizado por eles, falta de recursos para junção dos PDFs enviados, e neste caso o fator

limitante de somente dar suporte para envio de documentos que estão no contexto de um processo admissional.

Figura 3 – Interface do LugarH



Fonte: LugarH (2023).

3.1.3 Comparação do protótipo com estado da arte

Conforme visto as ferramentas encontradas não possuem exatamente a mesma ideia do proposto trabalho. Porém realizam processos muito semelhantes para o seu contexto de aplicação. O Pipefy e LugarH são mais robustos, porém ainda carecem de alguns recursos. Essas diferenças podem ser melhor observadas no Quadro 4.

Quadro 4 – Comparação de recursos suportados

Funcionalidade	Protótipo	Pipefy	LugarH
Personalização total das informações de documentos necessários no formulário de envio.	X	X	
Executar a aplicação em infraestrutura própria.	X		
Junção dos documentos PDF.	X		
Recursos de inteligência artificial.		X	X
Monitoramento do *status* da aplicação.	X		
Disponibilizar informações centralizadas sobre documentos.	X	X	X

Fonte: Acervo do autor (2023).

Em primeiro no quadro é possível verificar a funcionalidade de personalização total de quais documentos são necessários para um determinado processo. Conforme apresentado durante o desenvolvimento do trabalho, o protótipo desenvolvido permite essa opção. O Pipefy também permite a personalização de quais documentos podem compor um *template* ou formulário. Já o LugarH não permite a personalização visto que seu foco são documentos de admissão digital.

Em segundo a opção de executar a aplicação em infraestrutura própria diz sobre a capacidade de hospedar a aplicação em uma infraestrutura controlada pela organização interessada. Somente o protótipo desenvolvido permite esse tipo de abordagem, visto que isso vai contra o modelo de negócio “SAAS” das outras ferramentas.

A terceira funcionalidade trata de junção dos documentos enviados pelo usuário final, as outras ferramentas avaliadas não permitem realizar esse tipo de processamento, ao menos não de maneira nativa. O que se torna mais uma vantagem do protótipo em relação as outras ferramentas citadas.

O quarto ponto, é a disponibilidade de recursos de inteligência artificial, esses que não estavam no escopo do protótipo, portanto somente as outras ferramentas possuem esse tipo de recurso.

Em quinto lugar, está o suporte para monitoramento das aplicações, e somente o protótipo dentre as ferramentas apresentadas dispõe desse tipo de recurso. Isso está ligado também ao modelo de negócio adotado pelas empresas que fornecem as outras ferramentas.

A última funcionalidade analisada é a de disponibilizar informações sobre os documentos de forma centralizada, essa funcionalidade é possível em todas as ferramentas. Isso facilita para o usuário final obter as informações sobre os documentos, e realizar todo o processo.

4. PROTÓTIPO DE SISTEMA DISTRIBUIDO PARA MONTAGEM DE DOCUMENTOS DIGITAIS

Este capitulo descreverá as etapas do desenvolvimento deste trabalho, abordando tecnologias, rotinas, requisitos, entre outros.

4.1 VISÃO GERAL DO PROTÓTIPO

O protótipo tem como objetivo permitir que os usuários finais, sejam eles quais forem, terem uma maior facilidade no entendimento de quais documentos são necessários para determinado processo que ele está participando. Também irá fornecer de forma centralizada as informações necessárias para montar cada um desses documentos. Além disso irá disponibilizar de maneira segura uma área para realizar a junção dos arquivos PDF envolvidos no processo de montagem de cada documento.

O funcionamento do sistema se dá baseado em alguns conceitos. O primeiro deles é o Plano de documentos, ele se apresenta como um agrupador onde os documentos podem ser cadastrados. Pode-se ter como exemplo: Em um processo de inscrição para bolsas de estudo, existiria um plano de documentos com o nome da bolsa de estudos. Neste plano de documentos teríamos os documentos do processo de inscrição da mesma. Outro conceito é o de documentos do plano, estes que estarão disponíveis em uma lista, que contará com todos os documentos em questão. Estes podem ser cadastrados conforme necessidade, e em cada um poderão ser adicionadas informações indicando como deve ser montado, quais informações deve conter e etc.

O sistema protótipo foi, portanto, divido em duas áreas, a área administrativa e área de execução. A parte administrativa é responsável por permitir que os usuários do tipo administrador tenham acesso a funcionalidades de cadastros dos planos de documentos, dos documentos desses planos e também das informações necessárias em cada um desses documentos. Já a área de execução é montada dinamicamente com base nas informações inseridas na área administrativa. E é onde os usuários finais terão acesso para obter as informações dos documentos necessários, além da opção de juntar estes na plataforma.

4.2 REQUISITOS

De acordo com a visão geral do sistema, existem algumas funcionalidades e características que são necessárias para que a aplicação cumpra adequadamente os seus

objetivos. Desse modo a seguir serão tratados dos requisitos do sistema, que estão divididos em requisitos funcionais e não funcionais.

4.2.1 Requisitos Funcionais

Os requisitos funcionais definem as funcionalidades do sistema. Definindo funções, rotinas e recursos que o sistema deve disponibilizar. Eles podem ser vistos no Quadro 5.

Quadro 5 – Requisitos Funcionais

Número	Nome	Descrição
RF01	Criar Planos de Documento	Tela para realizar o cadastro de planos de documento.
RF02	Consultar Planos de documento	Tela que permite a visualização dos planos de documentos cadastrados.
RF03	Adicionar documentos em um plano	Opção para realizar a adição de novos documentos em um plano de documentos.
RF04	Remover documentos de um plano	Opção para realizar a remoção de documentos de um plano.
RF05	Editar documentos de um plano	Opção para realizar a edição de documentos de um plano.
RF06	Consultar documentos de um plano	Tela que permite a visualização dos documentos relacionados a um plano.
RF07	Adicionar exemplo de documento	Opção para adicionar um arquivo contendo um exemplo do documento a ser gerado.
RF08	Tela para montagem de documentos	Tela que permite ao usuário visualizar as informações dos documentos do plano e adicionar os documentos que ele deseja juntar para formar o documento de resultado.
RF09	Junção de documentos	Opção para realizar a junção dos arquivos enviados pelo usuário na tela de montagem.
RF10	*Download* dos documentos	Opção para permitir o usuário baixar os arquivos de resultado dos documentos que ele enviou.
RF11	Fila de processamentos assíncronos	O sistema deve possuir uma fila de processamento, para que os documentos sejam processados de forma única e ordenada.

Fonte: Acervo do autor (2023).

4.2.2 Requisitos Não Funcionais

Os requisitos não funcionais definem as características gerais da aplicação. Ou seja, itens que devem se aplicar a todas as funcionalidades do sistema, desse modo se aplicando a todo o sistema. Eles podem ser vistos no Quadro 6.

Quadro 6 – Requisitos Não Funcionais

Número	Nome	Descrição
RNF01	Linguagem de programação *backend*.	A linguagem para realizar o desenvolvimento dos serviços *backend* deve ser Java na versão 17.
RNF02	Banco de dados	O banco de dados relacional, utilizado para armazenar as informações administrativas e conteúdo binário dos documentos deve ser postgres.
RNF03	Arquitetura distribuída	A arquitetura ou topologia do sistema deve ser constituída por vários serviços separados, utilizando o conceito de microsserviços.
RNF04	Framework *backend*	O *framework* utilizado para desenvolvimento dos serviços *backend* deve ser *Spring Boot*.
RNF05	Biblioteca *frontend*	O biblioteca utilizada para desenvolvimento do serviço *frontend* deve ser VueJs.
RNF06	Monitoramento de estatísticas	Os serviços envolvidos no processamento dos arquivos devem possuir monitoramento de estatísticas. Como por exemplo: Quantidade de requisições, tempo de resposta, etc.
RNF07	Monitoramento de Logs	Os serviços envolvidos no processamento dos arquivos devem possuir monitoramento de logs remoto. Ou seja as aplicações devem enviar os logs de execução para um servidor de logs.

Fonte: Acervo do autor (2023).

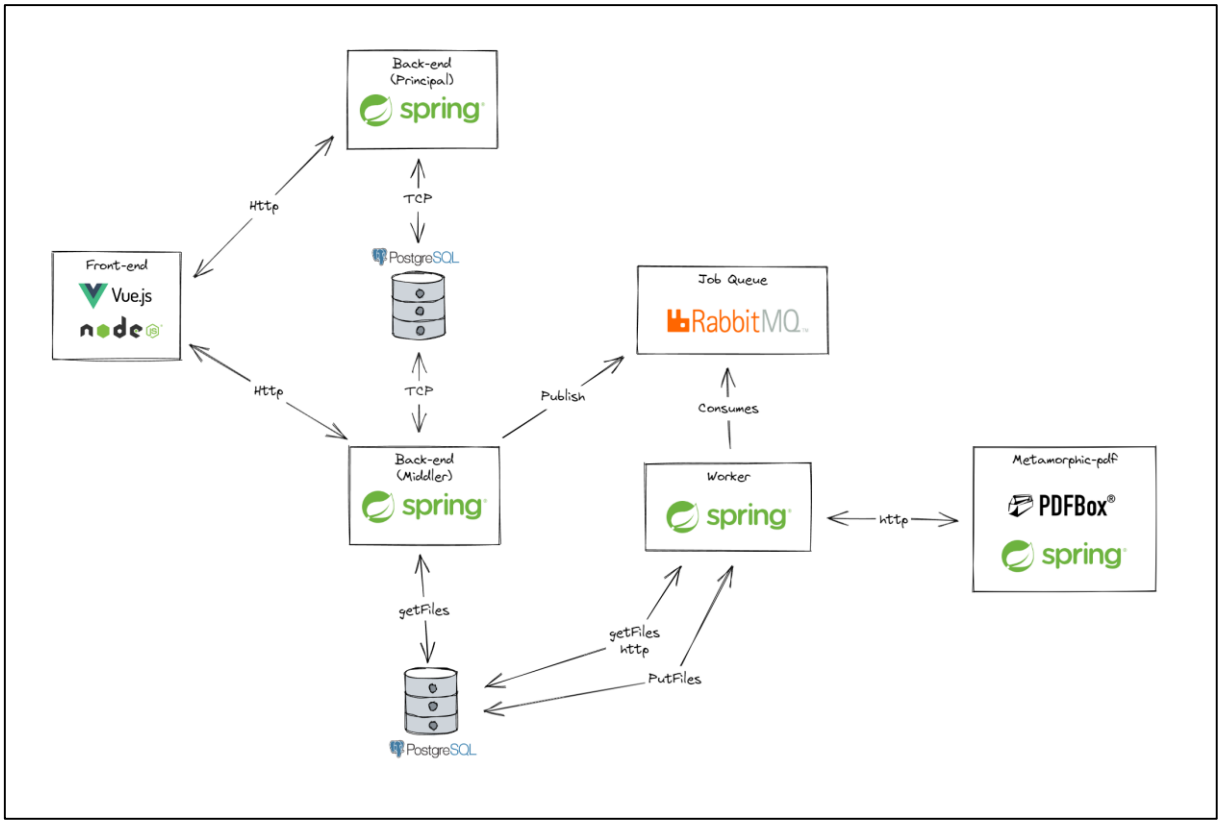
4.3 TOPOLOGIA

A topologia pretende servir como uma forma descrever a estrutura e a disposição dos elementos utilizados na elaboração do protótipo, objetivando um melhor entendimento sobre as estruturas utilizadas e seus objetivos singulares. Conforme imagem da Figura 4, o protótipo prevê ao todo 8 serviços, que são responsáveis por permitir o funcionamento da aplicação.

A estrutura operacional é composta por: 2 servidores de banco de dados, onde um deles é responsável por armazenar as informações da área administrativa, ou seja, dos planos de documentos e dos documentos em sí. O outro é responsável por armazenar as informações da área de execução, que é o conteúdo binários dos arquivos que os usuários enviam para o serviço, além também de armazenar os binários dos arquivos de resultados das junções efetuadas pela aplicação. 1 Servidor NodeJs com a parte *frontend* da aplicação, esse serviço de *frontend* é quem consome e manipula as informações do *backend* administrativo para montar as telas de execução aos usuários finais. Também permite realizar o cadastro das informações de planos de documentos e documentos. Existem ainda 4 serviços Java desenvolvidos com o *framework Spring Boot*, onde o 1° deles é o serviço responsável por lidar com o *backend* da área administrativa. O 2° é o *Middler* (intermediário), que é responsável por receber os documentos enviados pelo usuário, armazenar esses no banco de dados, montar e publicar a estrutura que deve ser encaminhada para a fila de processamentos assíncronos. O 3° serviço Java é o *Worker* (Trabalhador), ele é responsável por observar a fila de processamentos e executar os itens que nela existem, além disso durante o seu processamento ele realiza a comunicação com o 4° serviço Java que é responsável por realizar as junções dos documentos do usuário utilizando a

biblioteca Java conhecida como PDFBox. Por último existe o já mencionado serviço de fila de processamentos, que é basicamente um serviço utilizando a tecnologia do RabbitMq configurado para disponibilizar uma estrutura de fila.

Figura 4 – Topologia da aplicação

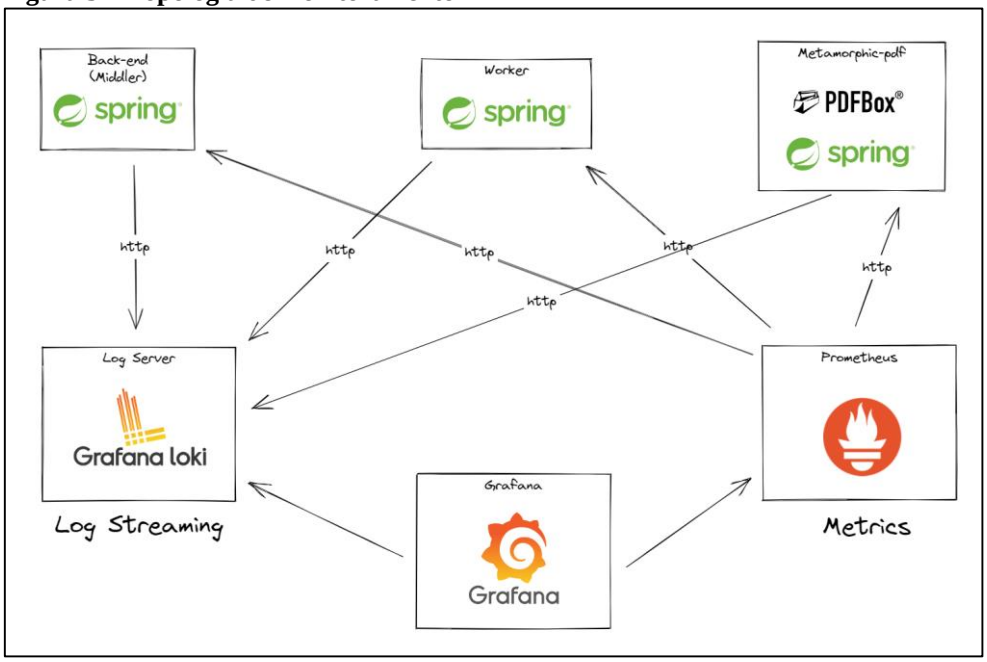


Fonte: Acervo do autor (2023).

4.3.1. Monitoramento

Foram adicionados ao projeto serviços para monitoramento das partes operacionais mais críticas. Que são os 3 serviços Java responsáveis pelo processamento dos arquivos. Cada um dos serviços realiza *streaming* de logs das operações executadas para o serviço Grafana Loki, que armazena essas informações para posteriores análises. Outro serviço importante é o Prometheus, este é responsável executar requisições HTTP para cada aplicação afim de obter informações sobre o seu *status*. No centro inferior temos o Grafana que é responsável por disponibilizar a interface WEB para análise dessas informações de monitoramento. Essa estrutura pode ser vista na Figura 5.

Figura 5 – Topologia de monitoramento



Fonte: Acervo do autor (2023).

4.4 MODELO DE DADOS

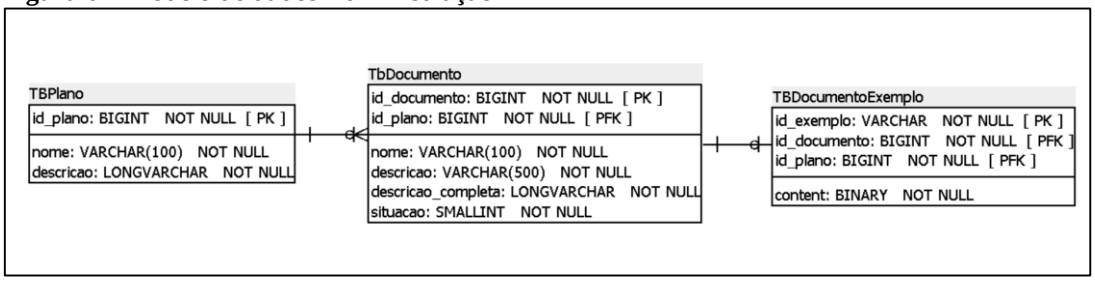
O modelo de dados do protótipo está também divido em área administrativa e área de execução. Sendo que o modelo da parte de administração possui 3 tabelas e o da área de execução é mais simples contando com apenas 2 tabelas.

Na Figura 6 temos o modelo da área administrativa, onde a tabela “TBPlano” é a responsável por armazenar as informações sobre os planos de documentos que podem ser cadastrados no sistema.

Outra tabela é a “TBDocumento”, esta é responsável por armazenar as informações dos documentos de um plano, ela conta com uma chave primária “composta” que é constituída por duas colunas: Id do documento e Id do plano de documento relacionado. Além disso ela conta com outras informações como a situação do documento, que pode ser utilizada no sistema para uma funcionalidade futura de desativar documentos, não necessitando excluir os mesmos.

A última tabela da área administrativa é a “TBDocumentoExemplo” que é responsável por armazenar o conteúdo binário de um documento de exemplo. A criação desta tabela separada para apenas armazenar o documento de exemplo, foi pensada para facilitar alterações posteriores na estratégia de armazenamento do conteúdo dos arquivos. Assim ela pode ser removida do banco de dados com mais facilidade.

Figura 6 – Modelo de dados Administração



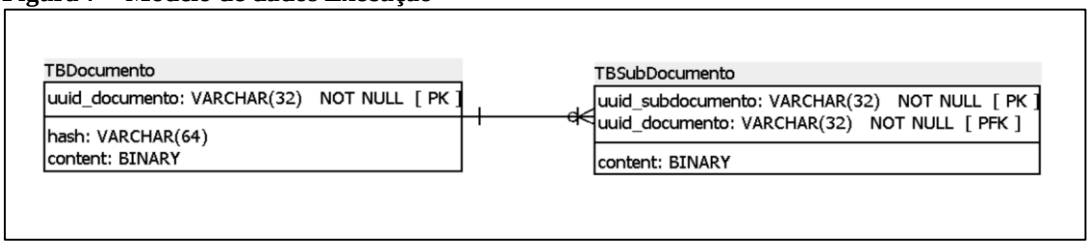
Fonte: Acervo do autor (2023).

A separação do banco de dados de informações administrativas e do conteúdo dos documentos foi pensada como uma forma de possibilitar uma manutenção mais simples das áreas do sistema. Além de permitir uma maior flexibilidade para mudanças de tecnologias, como já citado anteriormente. Ela também permite, por exemplo, que durante algum momento de pico no uso dos recursos de junção de documentos a área administrativa praticamente não seja afetada, já que são duas coisas separadas.

Na Figura 7, temos, o modelo de dados da área de execução da junção de documentos. Ele é composto pela tabela “TBDocumento”, que é responsável por armazenar informações sobre os documentos finais, ou seja, os documentos que serão gerados a partir da junção de vários outros documentos. Ela conta com uma coluna de UUID servindo como chave primaria, outra coluna de *hash*, que é utilizada para armazenar o *hash* resultante da concatenação dos tamanhos dos sub documentos daquele registro. Estratégia esta que foi utilizada para evitar o processamento da junção repetida dos mesmos documentos várias vezes. Assim poupando recursos e tempo de processamento. Essa tabela ainda é responsável por armazenar na coluna “content”, o conteúdo binário do arquivo final do processo de junção.

A outra tabela desse modelo é a “TBSubDocumento” que armazena o conteúdo de cada um dos sub documentos de um documento. Ou seja, dos vários documentos que serão unidos em um só documento, relacionando estes ao registro de documento final.

Figura 7 – Modelo de dados Execução

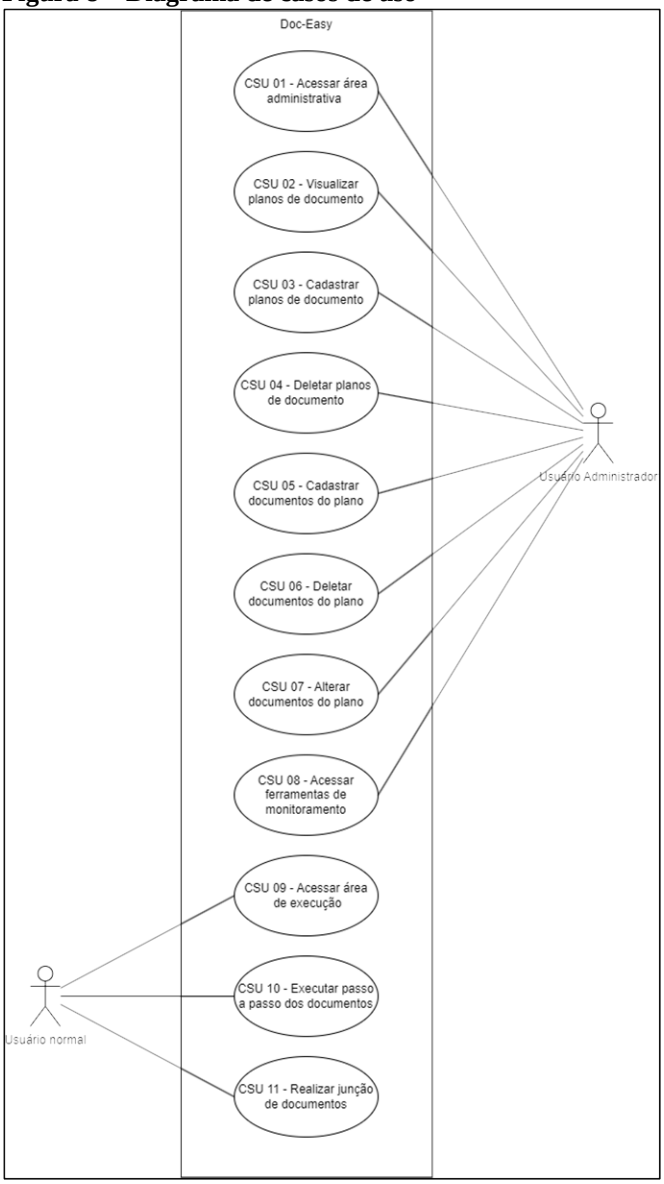


Fonte: Acervo do autor (2023).

4.5 CASOS DE USO

O diagrama de casos de uso, tem objetivo de exemplificar a forma como os agentes, sejam sistemas externos ou usuários, interagem com a aplicação. Mostrando as funcionalidades que são direcionadas para cada tipo de agente. O diagrama de casos de uso, pode ser visto na Figura 8.

Figura 8 – Diagrama de casos de uso



Fonte: Acervo do autor (2023).

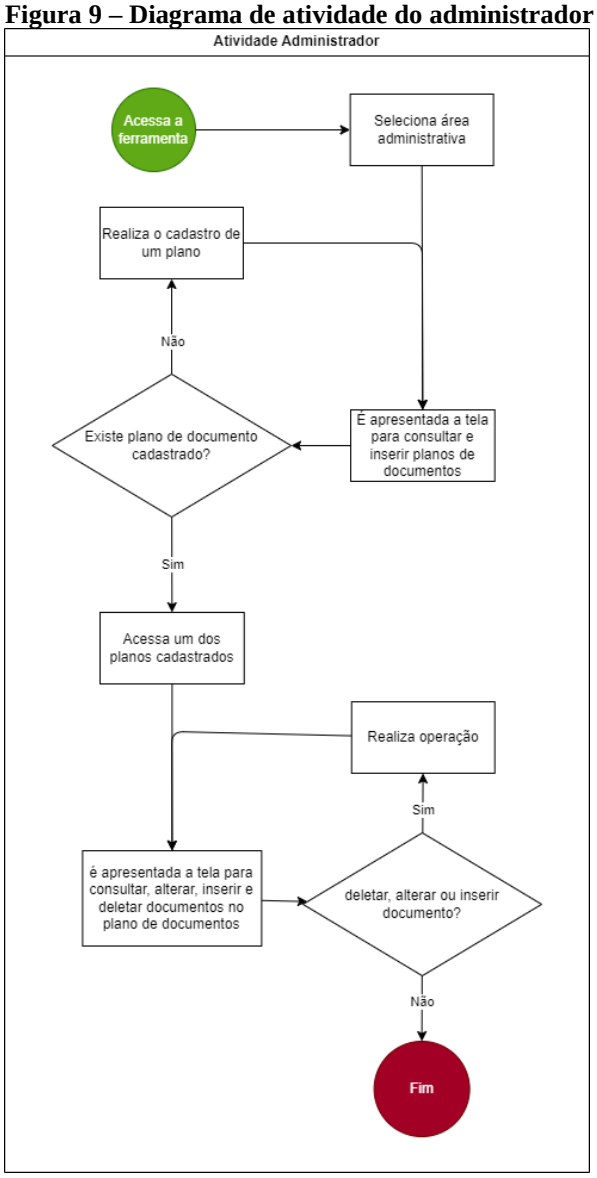
4.6 PROTÓTIPOS

A presente seção objetiva apresentar as interfaces do sistema, com intenção de proporcionar um melhor entendimento do protótipo, bem como explicar quem utilizará as funcionalidades através de diagramas de atividade específicos para cada perfil previsto de usuário que o sistema pode apresentar.

4.6.1 Administração e monitoramento do sistema

O fluxo de um usuário que deseja administrar o sistema, está presente na Figura 9, onde é possível observar com detalhes o seus passos e opções dentro da aplicação.

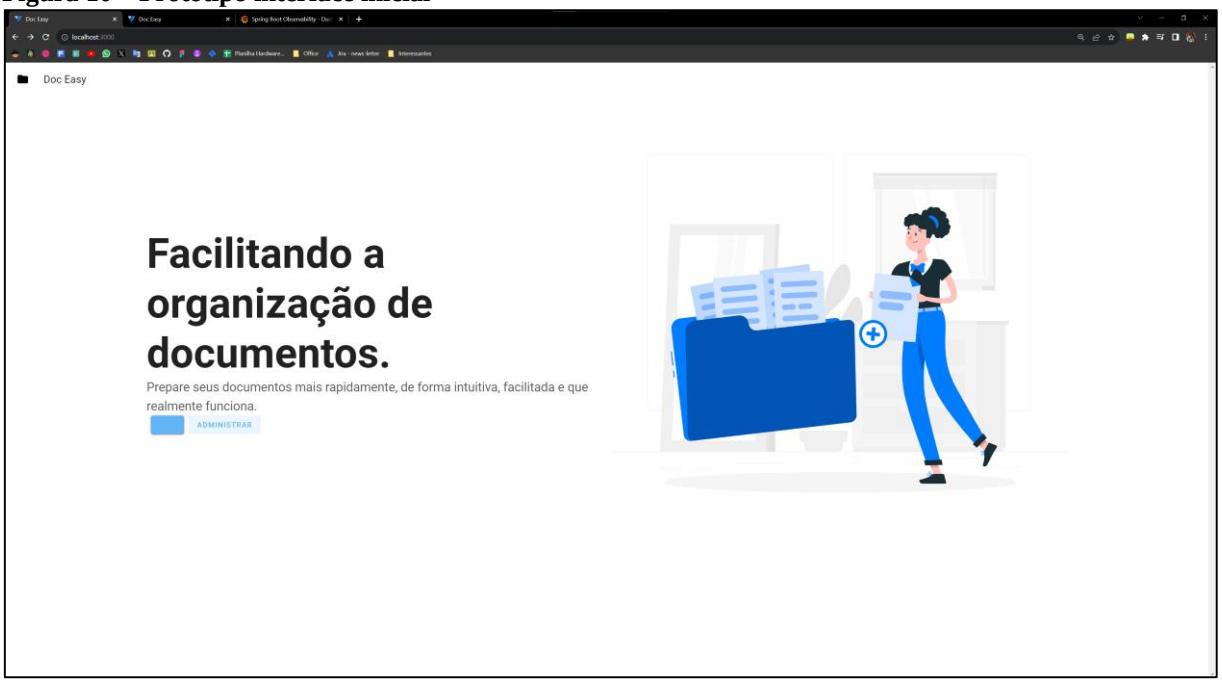
Figura 9 – Diagrama de atividade do administrador



Fonte: Acervo do autor (2023).

A interface que disponibiliza a opção para seleção da área administrativa pode ser vista na Figura 10. Ela apresenta a tela inicial que o sistema irá exibir para um usuário que abriu a aplicação.

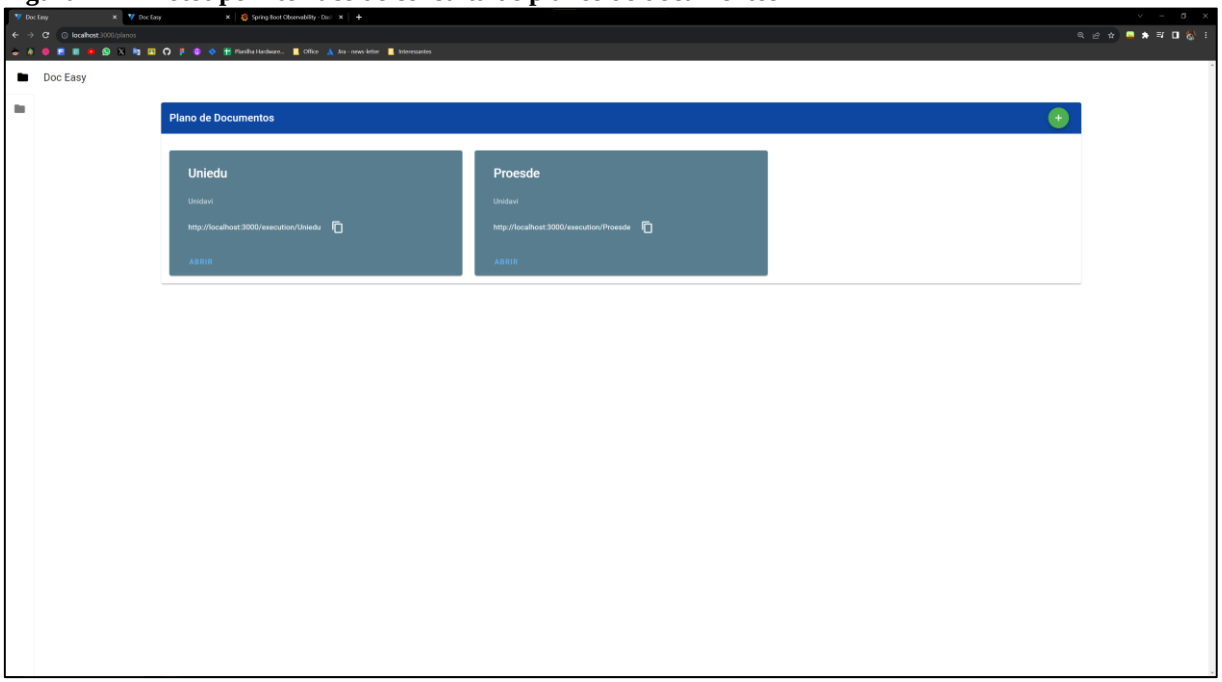
Figura 10 – Protótipo interface inicial



Fonte: Acervo do autor (2023).

Após o usuário administrador clicar em “administrar” o sistema irá exibir conforme Figura 11, a consulta dos planos de documentos já cadastrados no sistema. Os quais o usuário administrador poderá abrir para editar ou então se desejar, criar novos planos de documentos. Além de visualizar também a URL que os usuários finais devem acessar para iniciar a execução da montagem dos documentos daquele plano de documentos.

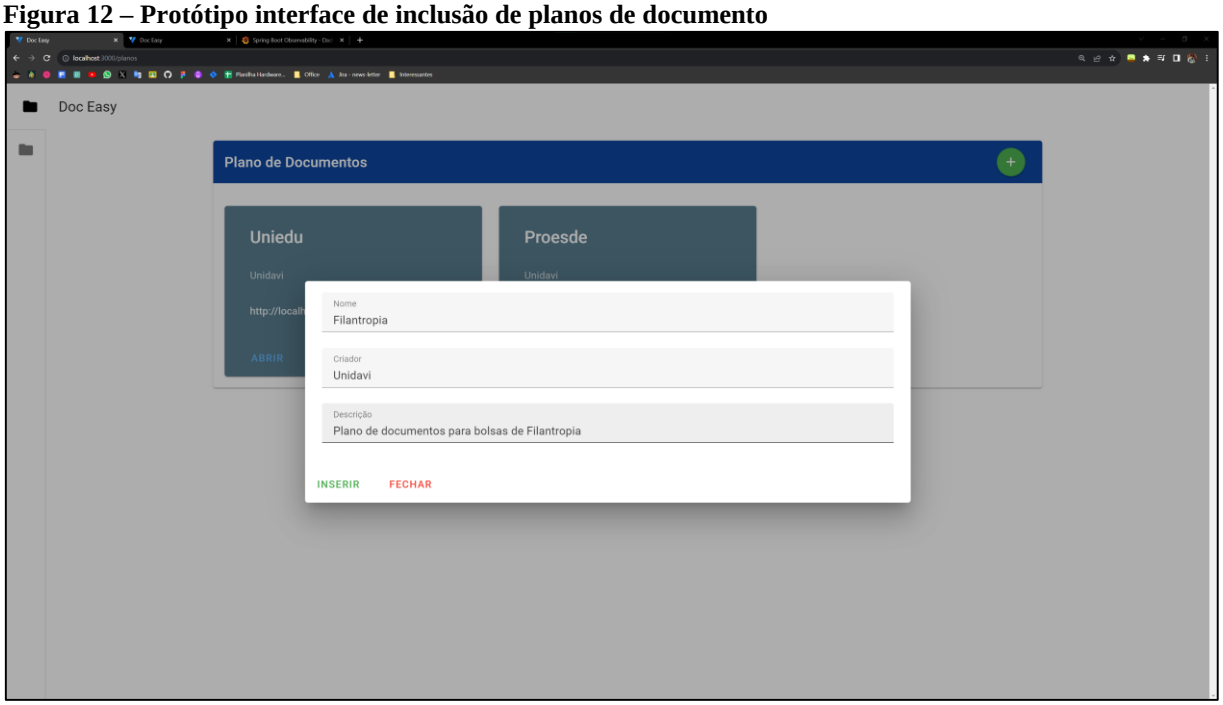
Figura 11 – Protótipo interface de consulta de planos de documentos



Fonte: Acervo do autor (2023).

A tela para criação de um novo plano de documentos pode ser vista na Figura 12. Onde é possível informar o Nome, Criador e Descrição para o plano de documentos.

Figura 12 – Protótipo interface de inclusão de planos de documento

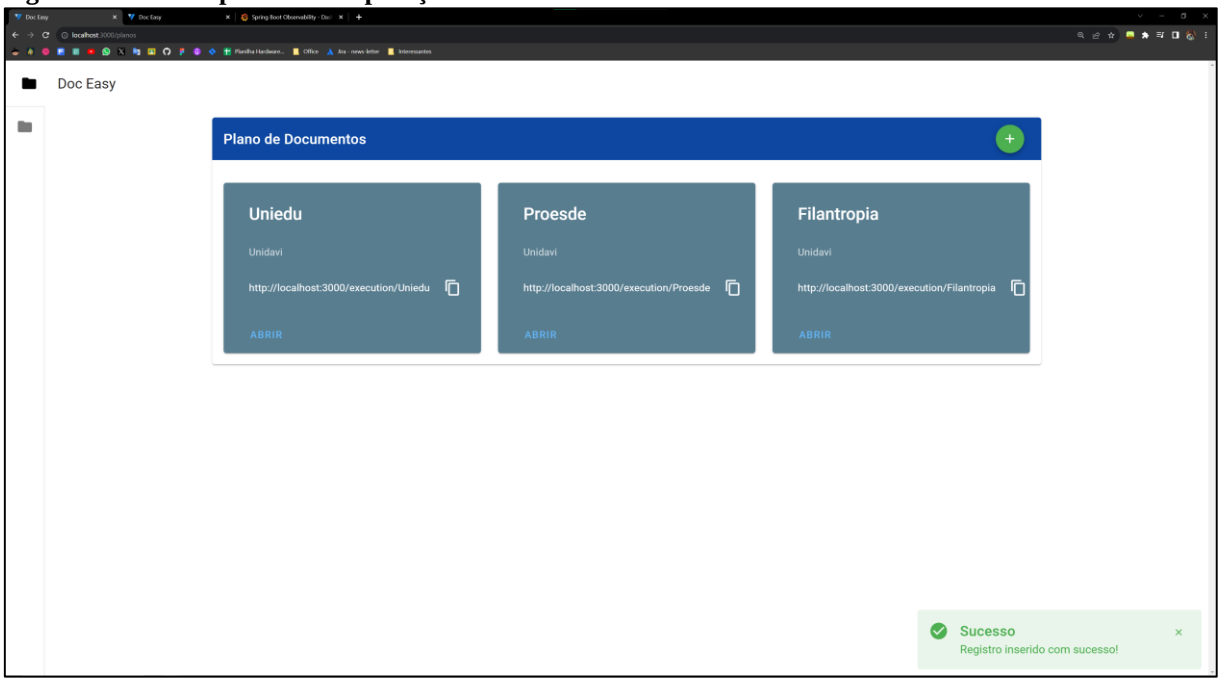


Fonte: Acervo do autor (2023).

Após a confirmação da inclusão pelo usuário, ao realizar a operação o sistema irá emitir um aviso indicando que a operação foi realizada com sucesso caso isso realmente aconteça.

Caso contrário irá emitir um aviso de erro. Informando que a operação não foi realizada com sucesso. O que pode ser visto na Figura 13 e 14.

Figura 13 – Protótipo aviso de operação realizada



Fonte: Acervo do autor (2023).

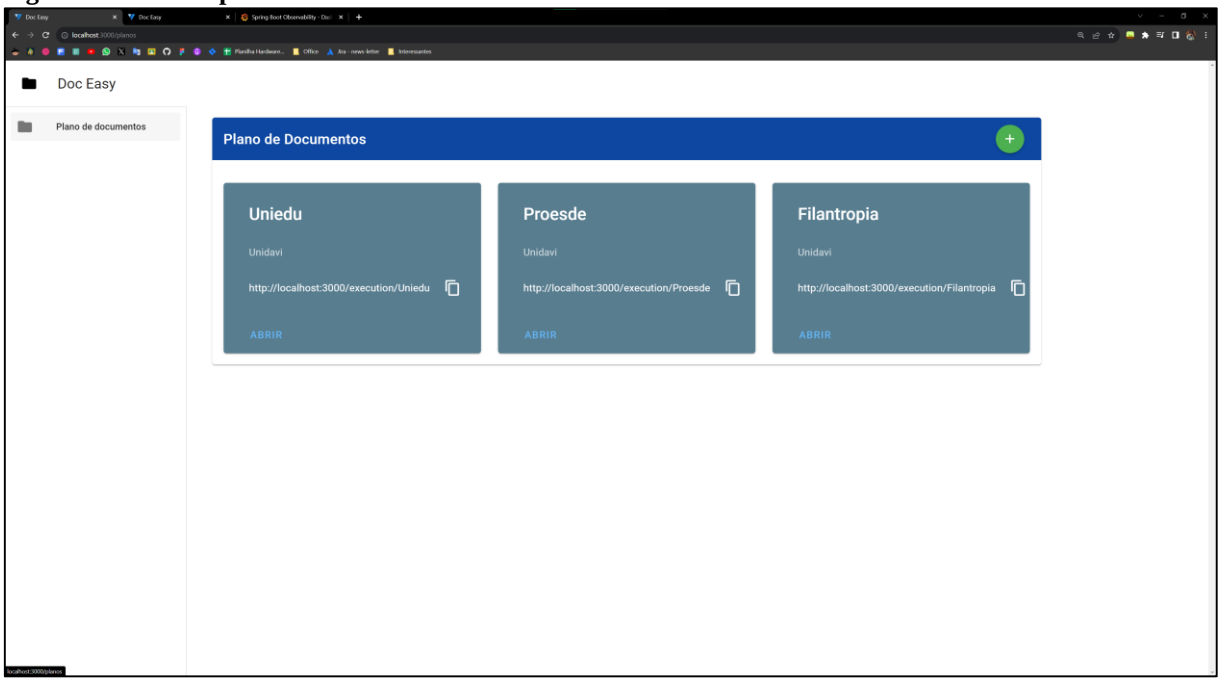
Figura 14 – Protótipo interface de aviso de falha ao realizar operação

Fonte: Acervo do autor (2023).

Além das opções já apresentadas na consulta de planos de documentos, e na outra área do sistema. Na Figura 15, é possível observar que o usuário administrador também irá dispor

de uma aba lateral para acessar o painel de consulta dos planos de documento. Esse recurso possui o objetivo de facilitar a navegação do usuário no sistema.

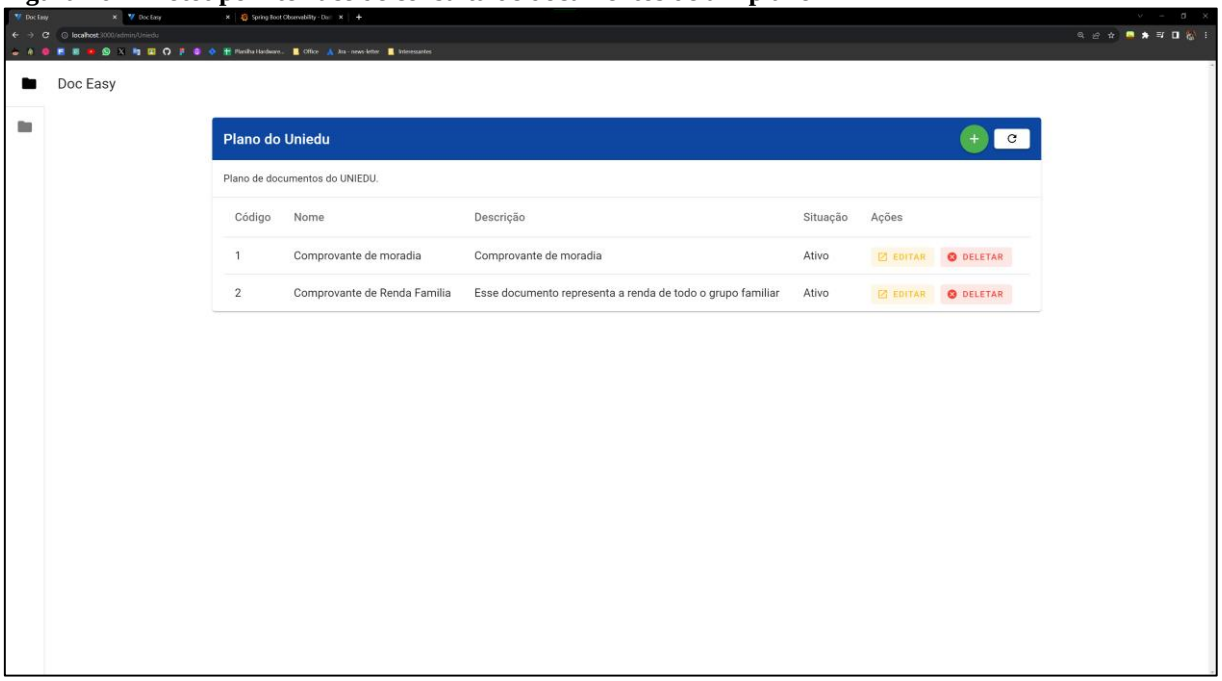
Figura 15 – Protótipo interface de aba lateral



Fonte: Acervo do autor (2023).

Ao abrir um plano de documento, conforme a Figura 16, o usuário terá disponível a consulta de documentos do plano de documentos. Ela dispõe de recursos para adicionar, editar já existentes, remover e atualizar os registros da consulta.

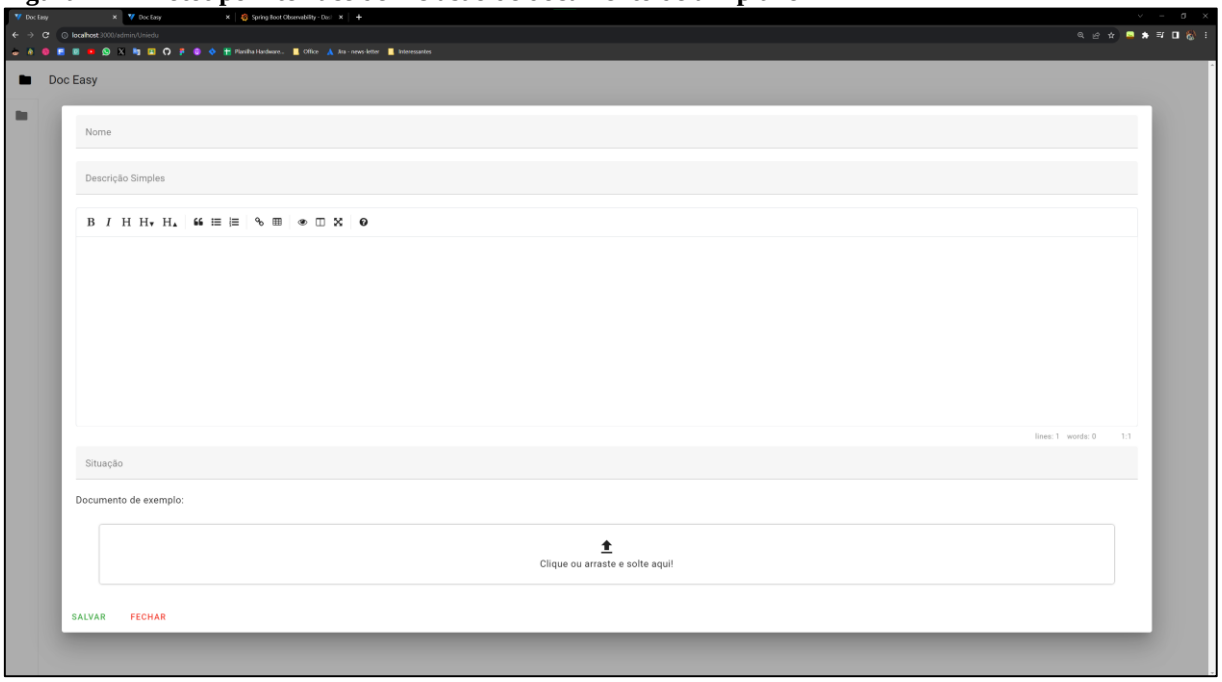
Figura 16 – Protótipo interface de consulta de documentos de um plano



Fonte: Acervo do autor (2023).

A interface de adição de documentos pode ser vista na Figura 17. Ela dispõe de campos para adicionar Nome, Descrição Simples, Descrição Completa, Situação e Documento de exemplo. O campo de descrição completa é um editor que permite ao usuário adicionar Títulos, Itálico, Negrito, Além de listas e tabelas. O campo de documento de exemplo permite ao usuário selecionar um arquivo PDF que corresponda a um exemplo daquele determinado documento que está sendo cadastrado.

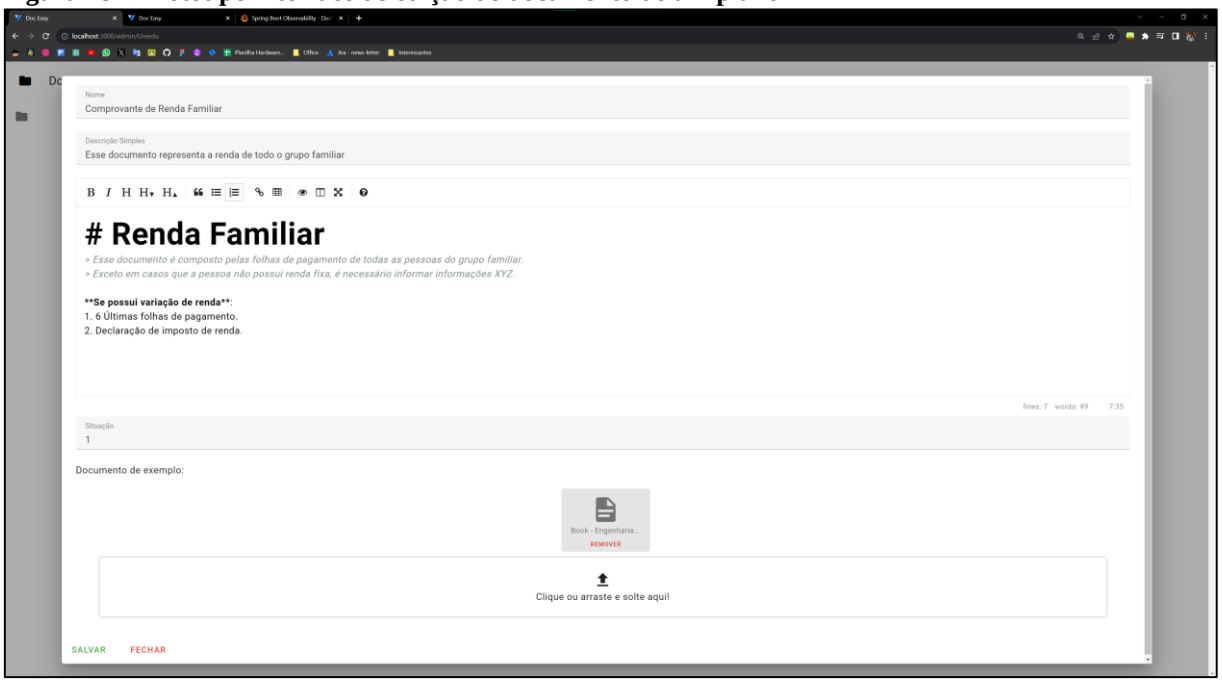
Figura 17 – Protótipo interface de inclusão de documento de um plano



Fonte: Acervo do autor (2023).

A interface de edição de um documento é igual a interface de criação. Porém ela já contará com os dados do documento. Isso pode ser observado na Figura 18. A opção de deletar um documento somente realiza a deleção do mesmo e exibe o aviso de sucesso na operação assim como na adição e alteração de planos ou documentos.

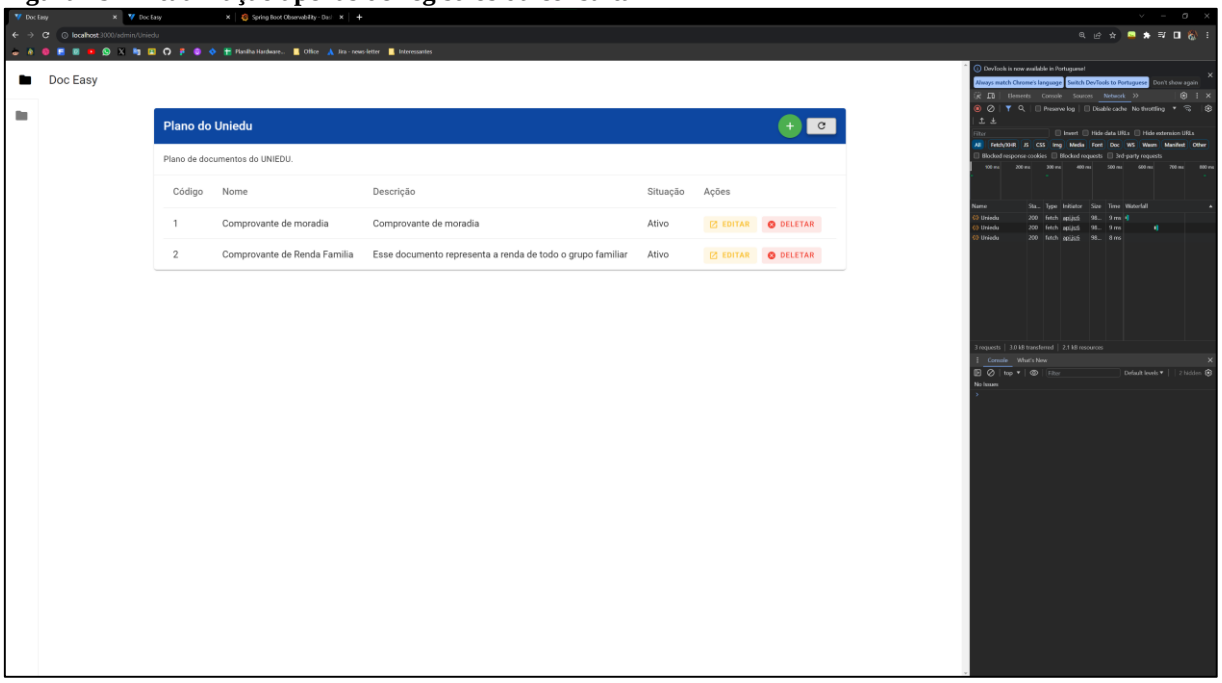
Figura 18 – Protótipo interface de edição de documento de um plano



Fonte: Acervo do autor (2023).

A consulta de documentos, ainda conta com a opção de atualizar os registros da tela, esse recurso foi desenvolvido de maneira a melhorar a experiência do usuário, não realizando o carregamento de toda a página, e sim apenas dos registros da consulta. As requisições realizadas podem ser observadas no *DevTools* na Figura 19.

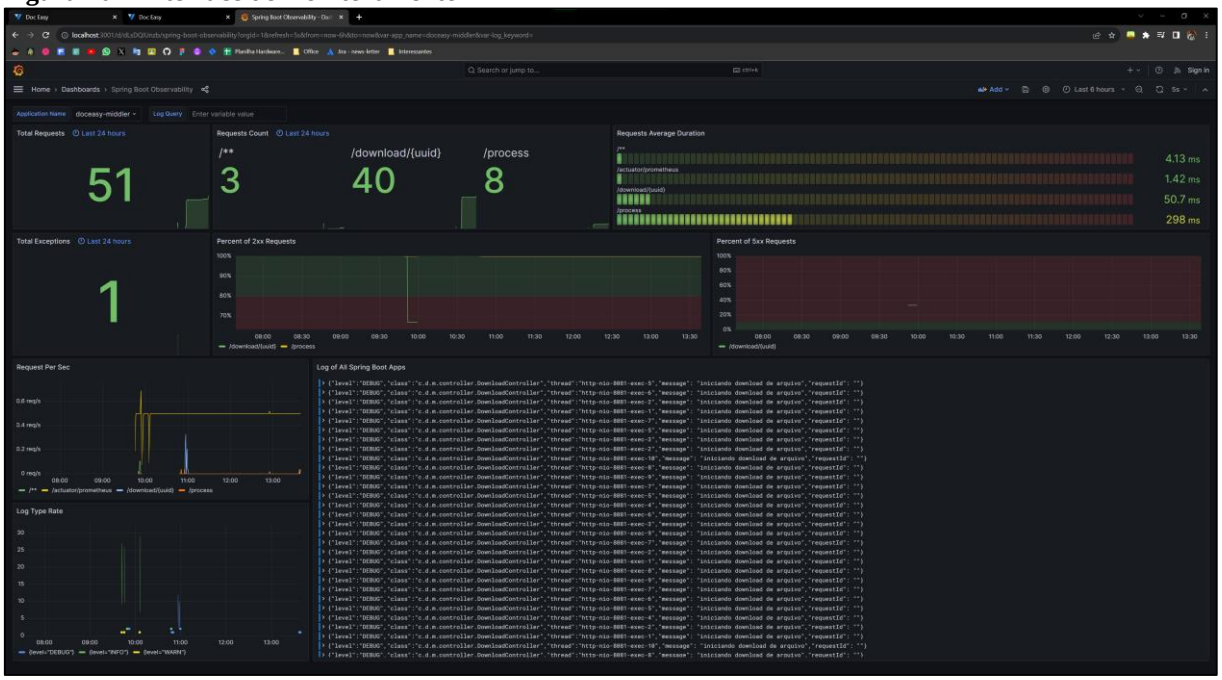
Figura 19 – Atualização apenas de registros da consulta



Fonte: Acervo do autor (2023).

O usuário administrador ainda pode acessar a área de monitoramento da aplicação, ela pode ser vista na Figura 20, a mesma é disponibilizada utilizando a ferramenta Grafana. Essa ferramenta permite que sejam criados *dashboards* com dados disponibilizados pelas aplicações. A interface disponibiliza informações como quantidade total de requisições que as aplicações monitoradas receberam, quais os *end-points* da aplicação foram requisitados, tempo de resposta por *end-point*, número total de exceções registradas pela aplicação, porcentagem de requisições com sucesso e com falha, número de requisições por segundo, indicador de tipos de logs, além de registros dos logs das aplicações. O que permite ao administrador ter várias informações para que seja possível definir melhorias no sistema e notar quando algo foge dos padrões.

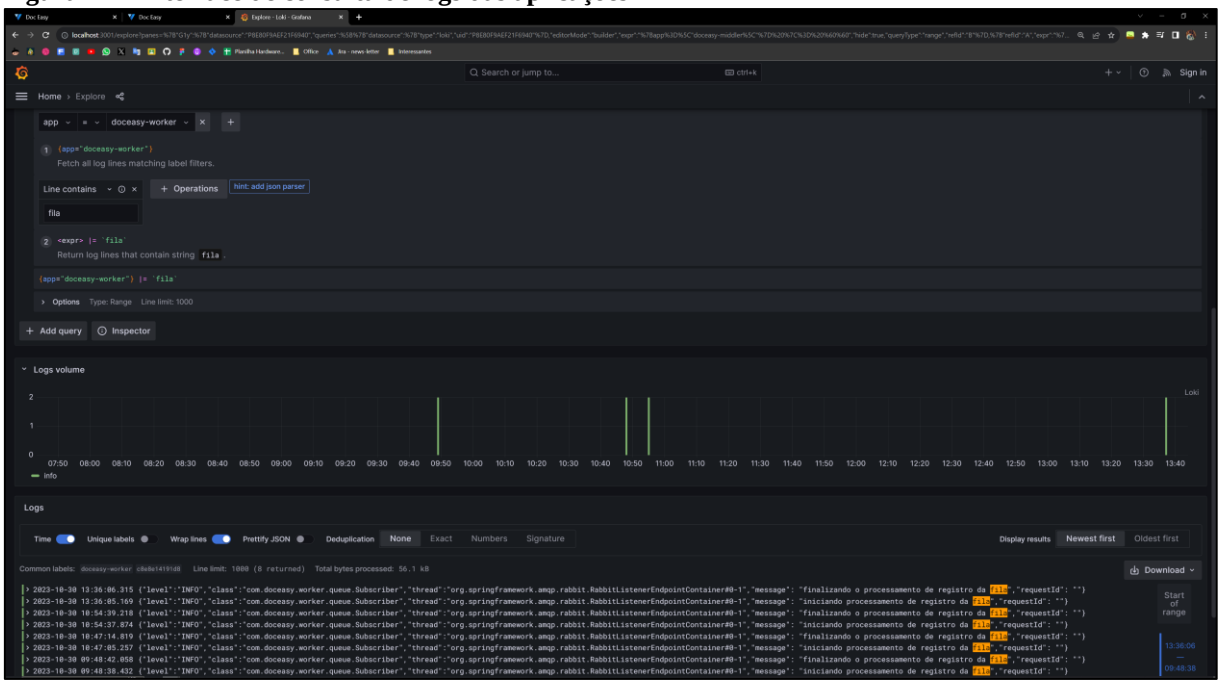
Figura 20 – Interface de monitoramento



Fonte: Acervo do autor (2023).

Além do painel geral, é possível notar na Figura 21, a área de exploração dos logs, onde é possível realizar consultas nas informações disponíveis nos logs das aplicações.

Figura 21 – Interface de consulta de logs das aplicações

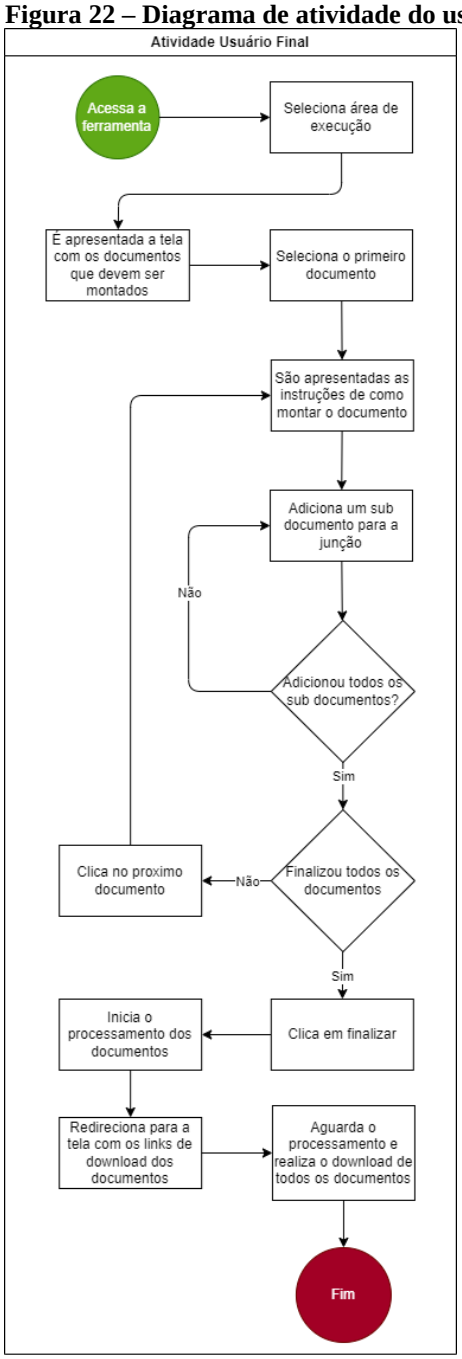


Fonte: Acervo do autor (2023).

4.6.2 Área de execução

A área de execução é onde o usuário irá acessar as informações dos documentos e realizar a montagem dos mesmos de acordo com o plano que ele acessou. Desse modo o fluxo de um usuário final do sistema pode ser observado com detalhes na Figura 22.

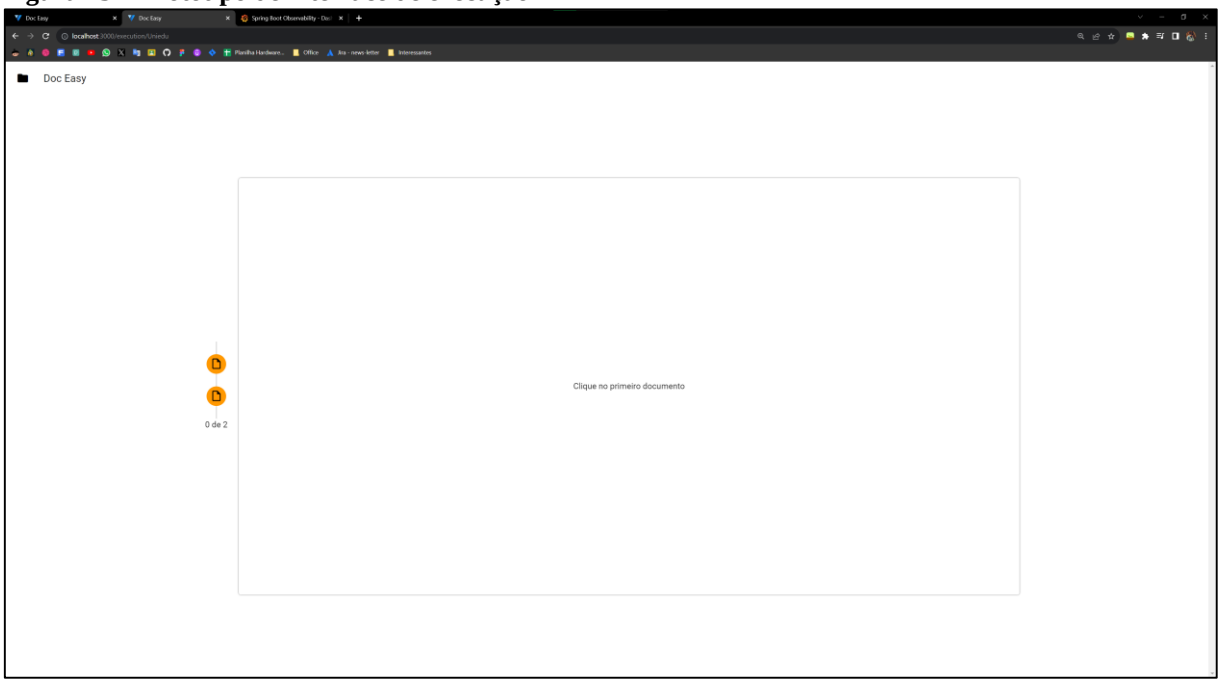
Figura 22 – Diagrama de atividade do usuário final



Fonte: Acervo do autor (2023).

A interface inicialmente não virá com nenhum documento selecionado, como pode ser observado na Figura 23, desse modo o usuário deve selecionar um documento para visualizar suas informações. Os documentos estão disponíveis na linha lateral em cor laranja.

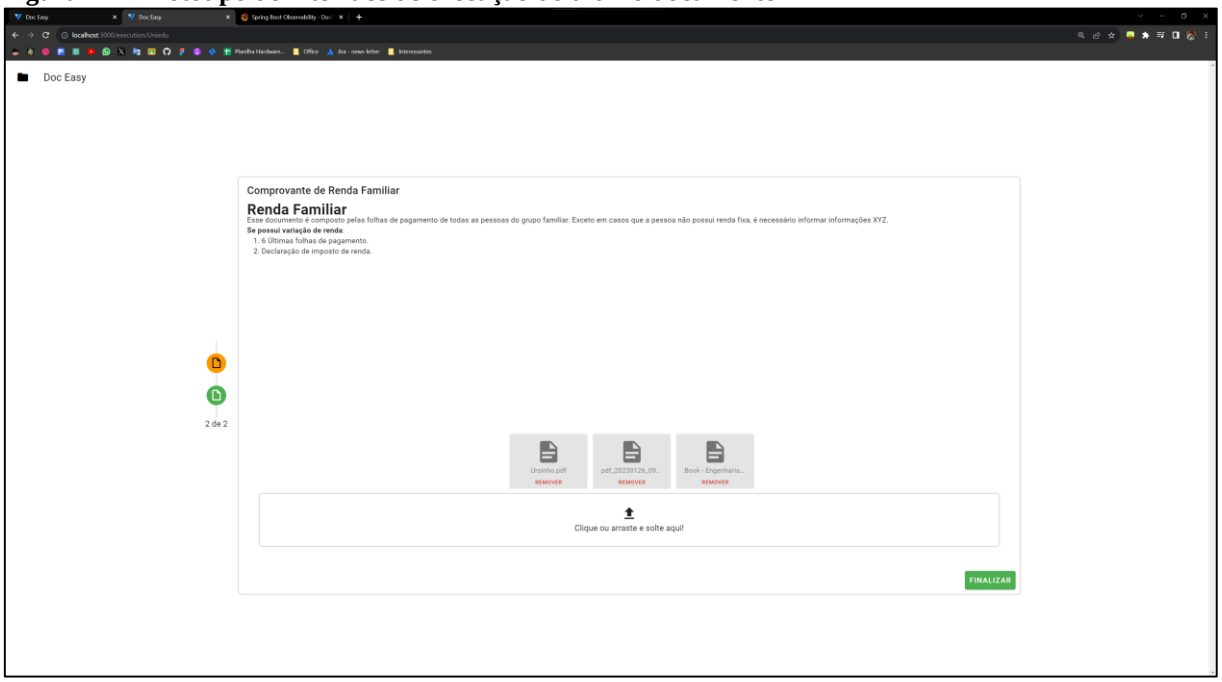
Figura 23 – Protótipo de interface de execução



Fonte: Acervo do autor (2023).

Conforme a Figura 24, ao selecionar um documento são apresentadas as informações sobre o mesmo, de acordo com o configurado pelo administrador. Desse modo, o usuário pode seguir as orientações e realizar a adição dos “pedaços” do documento final. No último documento será possível realizar a “finalização” do processo. Isso irá disparar o processamento de junção dos mesmos.

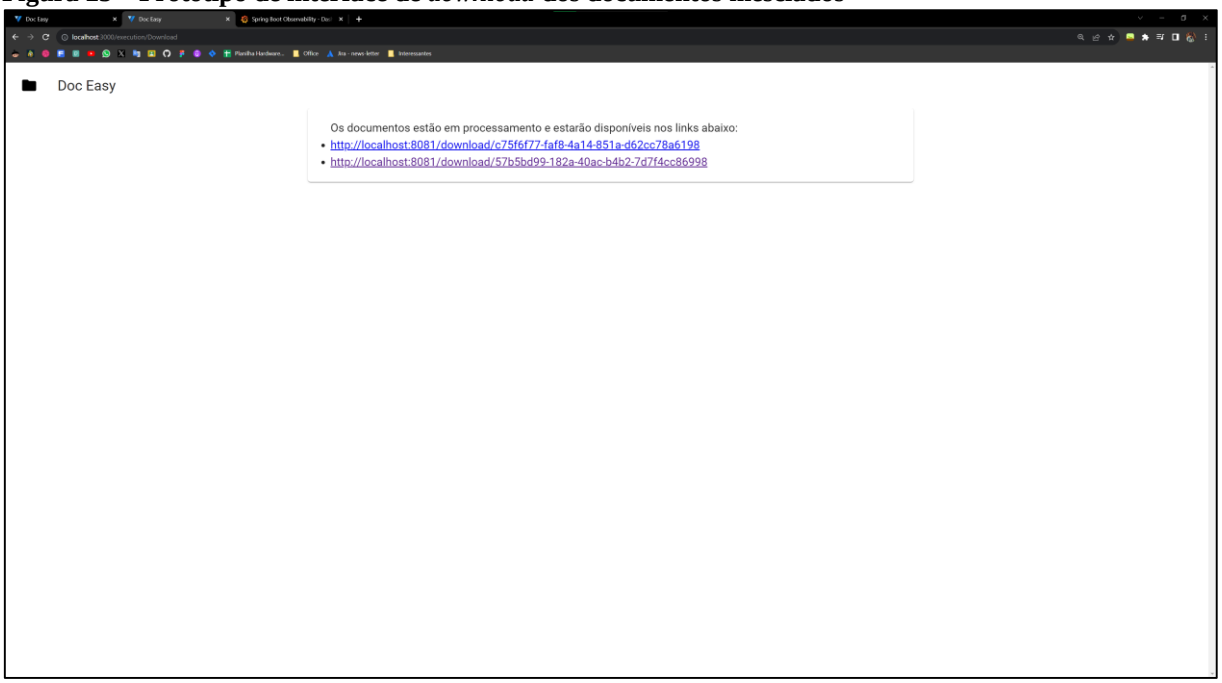
Figura 24 – Protótipo de interface de execução do último documento



Fonte: Acervo do autor (2023).

Ao realizar a finalização, conforme é possível visualizar na Figura 25, será apresentada uma lista com os *links* para *download* de cada um dos documentos processados. Ao clicar nos *links* será aberto o documento, onde é possível realizar o seu *download*.

Figura 25 – Protótipo de interface de *download* dos documentos mesclados



Fonte: Acervo do autor (2023).

4.7 IMPLEMENTAÇÃO

Nesta sessão expõe-se detalhes de implementação de todos os serviços que juntos realizam as operações do sistema. Apresentando diagramas de classes, pacotes e sequência. Envolvendo os passos e estrutura de classes que esses serviços implementam e realizam.

4.7.1 Visão geral

A visão geral da implementação busca explicar conceitos que foram utilizados no desenvolvimento dos serviços. Sendo possível entender melhor a arquitetura e funcionamento dos mesmos.

4.7.2 Classes e pacotes

Os serviços desenvolvidos são compostos por vários pacotes de classes, estes são usados para organização das classes dos projetos, além de ser uma forma de separar as camadas de responsabilidade do código em diferentes contextos e pastas. Isso facilita o entendimento e manutenção do código. Sendo assim os serviços no geral são compostos por 5 pacotes principais. São eles: *Controller*, DTO, *Entity*, *Service* e *Repository*.

Outros pacotes podem ser criados de acordo com o contexto da aplicação para facilitar a organização e direcionar os arquivos para o domínio de negócio da aplicação. O que facilita o entendimento do funcionamento da mesma. No entanto a estrutura supracitada, é a estrutura de pacotes mais comum para realizar a criação de aplicações do modelo *CRUD*.

4.7.2.1 Pacote *Controller*

O pacote *Controller* contém as classes de controladores, que são as classes responsáveis por definir a interface REST de acesso aos recursos do serviço. Ou seja, são nas classes de controladores que teremos as definições de quais recursos e como eles estão disponíveis para o mundo externo. Nesse caso essas classes definem os *end-points* que a aplicação disponibiliza, além de definir o método HTTP que será utilizado em cada um dos *end-points*.

Na Figura 26, é possível verificar um exemplo de um controlador REST desenvolvido com o *framework Spring Boot*.

Após a criação de uma classe é necessário utilizar a *annotation* de "*@RestController*". Isso irá permitir a ferramenta entender qual o tipo dessa classe. Além disso com a "*@RequestMapping*" é possível definir qual será o *path* para acessar os *end-points* dessa classe. Ao criar um método é necessário anotá-lo com um dos verbos HTTP.

Desse modo ao acessar o endereço do serviço no *path* "/api/v1/hello" o mesmo irá responder uma *string* com o conteúdo "Hello World!". Dada a explicação, é possível notar a facilidade para criação de uma API utilizando essa ferramenta.

Figura 26 – Exemplo de controlador

```java
package dev.spring.teste.controller;

import org.springframework.web.bind.annotation.GetMapping;

@RestController
@RequestMapping("/api/v1/")
public class ExampleController {

    @GetMapping("/hello")
    public String test() {
        return "Hello World!";
    }

}

```

Fonte: Acervo do autor (2023).

4.7.2.2 Pacote DTO

Pacote *DTO*, este é responsável apenas pela transferência de dados entre as camadas, funciona como uma forma de padronizar como os serviços e controladores devem se comunicar. Existe um DTO para cada contexto, ou seja, pode existir um DTO de carro, que abrigaria informações como ano, cor, modelo, etc. Essa camada, não contem regras de negócio, contém apenas as informações sobre o contexto que ela está inserida.

4.7.2.3 Pacote *Entity*

Esse pacote é semelhante ao *DTO*, porém não se restringe apenas a transferência de informações. Ele também pode conter outras regras, como validações e manipulações nas estruturas de dados.

Na Figura 27, pode-se observar um exemplo de uma entidade criada nos padrões do *framework Spring Boot*.

Utilizando a *annotation* "*@Entity*" é definido o tipo da classe. Assim o *Spring* pode entender que se trata de uma entidade do sistema. Com "*@Table*" é possível definir o nome da tabela no banco de dados. Após isso são necessários atributos com os respectivos campos da tabela. Conforme as *annotations* presentes nos atributos, é possível definir regras de validação, baseadas nas regras do banco de dados.

Figura 27 – Exemplo de entidade

```java
package com.doceasy.backend.entity;

import java.util.UUID;

@Data
@Entity
@Table(name = "tbdocumento")
public class Document {

    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID uuid;

    private Long idPlano;

    @Column(length = 100, nullable = false)
    private String nome;

    @Column(length = 100, nullable = false)
    private String descricao;

    private String descricaoCompleta;

    private Integer situacao;
```

Fonte: Acervo do autor (2023).

4.7.2.4 Pacote *Service*

Pacote *Service*, representa a camada que é utilizada pelos controladores para realizar o processamento de fato, as classes desse pacote utilizam as classes especializadas em determinado contexto requerido. Em um cenário de inclusão de um registro, por exemplo, essa camada é quem se comunica com as classes que lidam com o banco de dados. Além disso ela pode conter regras de negócio e chamadas para validações.

4.7.2.5 Pacote *Repository*

Pacote *Repository* esse pacote é responsável por realizar a comunicação com os recursos e classes ligadas ao banco de dados. No caso do *framework Spring Boot* com a biblioteca *Data-*

JPA que foi utilizada, é necessário apenas criar interfaces de comunicação, informando qual o objeto de entidade que aquele *repository* irá utilizar e o tipo da sua chave primária no banco de dados. Todo o resto da comunicação com o banco é abstraída com os métodos padrões e estrutura da biblioteca. Para facilitar a visualização na Figura 28, está um exemplo de um *repository* criado no desenvolvimento do projeto.

Figura 28 – Exemplo de repositório

```java
package com.doceasy.backend.repository;

import java.util.List;

@Repository
public interface DocumentRepository extends JpaRepository<Document, UUID>{

    List<Document> findByIdPlano(Long id);

}
```

Fonte: Acervo do autor (2023).

4.7.3 Serviço *backend* administrativo

O serviço de *backend* administrativo realiza as operações para salvar as informações no banco de dados administrativo. Portanto esse é o serviço responsável por conter a lógica que permite as operações *DML* de inserir, alterar e deletar registros nas tabelas de Documento, Exemplo de Documento e Planos de documento. Essas tabelas foram detalhadas na Figura 6 com o diagrama de entidade relacionamento.

Conforme a Figura 29 abaixo, a aplicação para o serviço administrativo é composta por 5 pacotes e 17 classes. Os pacotes seguem a estrutura padrão, as classes também seguem o padrão definido para os pacotes, porém com suas especializações e particularidades.

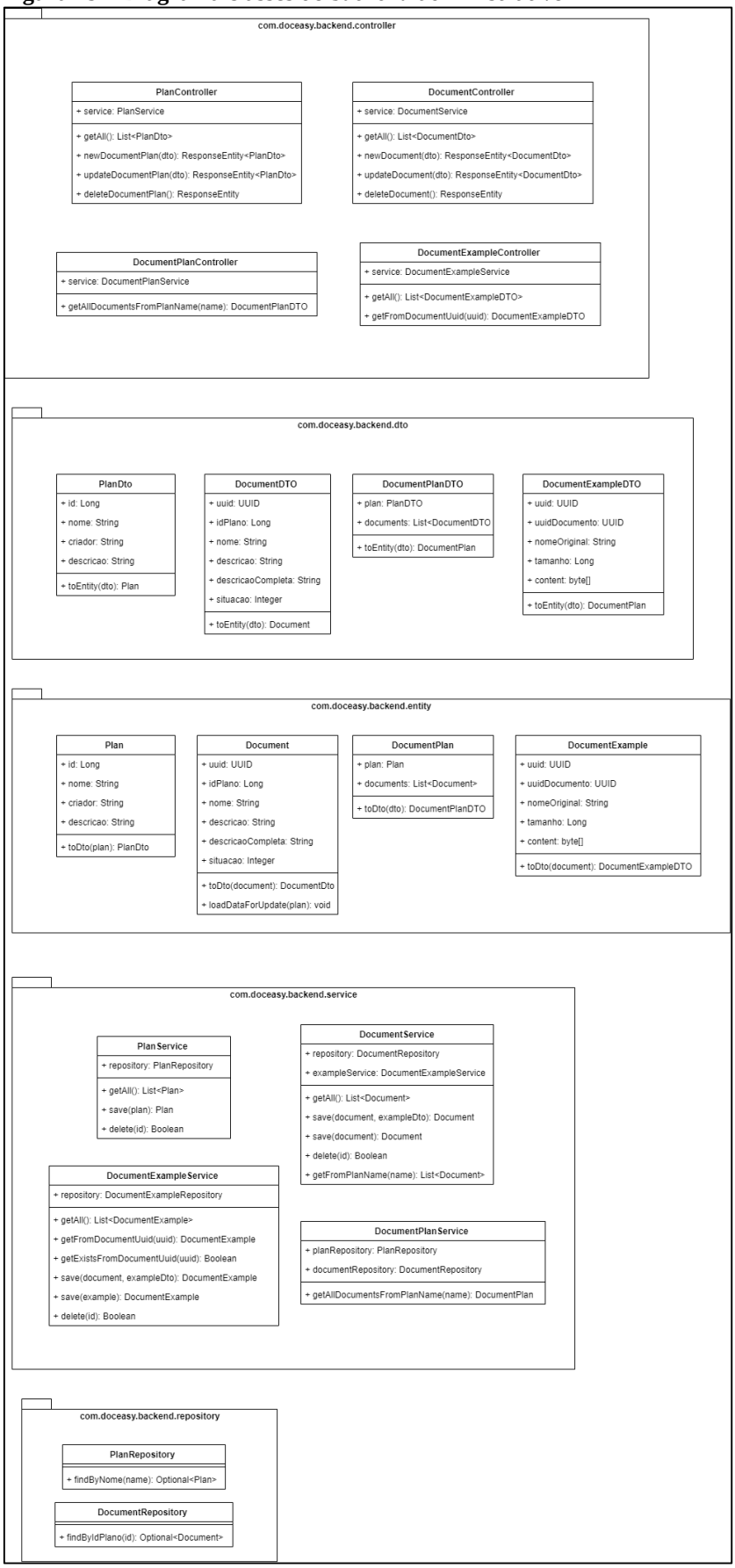
Existem 4 controladores para definição dos recursos de acordo com o contexto. O “PlanController” é responsável por disponibilizar os *end-points* para manipular os registros da tabela de Planos de documentos. Portanto ele possui opções para criar, deletar, atualizar e recuperar planos de documentos. O mesmo se aplica para o controlador de documentos. O “DocumentPlanController” possui apenas um *end-point* para retornar todos os documentos de um plano específico. O último controlador é o de exemplos de documentos que possui métodos para listar os documentos de exemplo.

O pacote de DTO possui classes para as entidades do sistema, assim como o pacote *entity*. Essas classes apenas possuem objetivos diferentes, mas sua composição é igual, possuem

apenas os atributos que existem nas tabelas relacionadas, afim de permitir operações no banco de dados e transferência de informações.

No pacote *Service* tem-se a comunicação com as classes do pacote *repository* que apenas realiza a ponte com os recursos da biblioteca *Spring Data JPA*.

Figura 29 – Diagrama classes do *backend* administrativo



Fonte: Acervo do autor (2023).

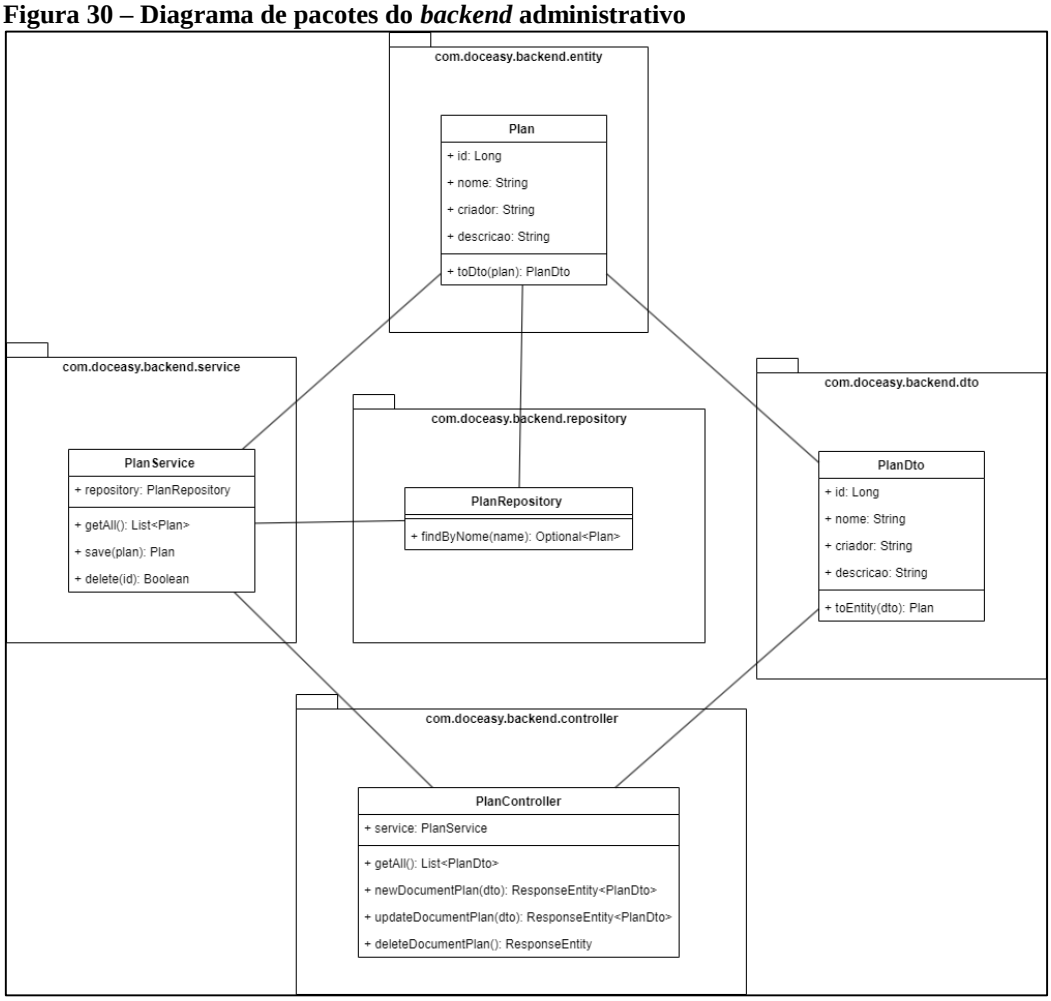
4.7.3.1 Relação entre os pacotes

Com intuito de elucidar melhor a relação entre os pacotes da aplicação e sua divisão, na Figura 30 é demonstrado quais classes estão associadas a quais, dado o contexto de plano de documentos. Neste caso, o fluxo da aplicação sempre inicia no *controller*, como pode-se observar o “PlanController” utiliza a classe “PlanDto” e “PlanService” para realizar as suas operações. Já “PlanService” utiliza tanto a entidade “Plan” como o repositório “PlanRepository” para realizar as operações no banco de dados.

Portanto pode-se se entender que o processo que é iniciado no controlador, utiliza uma classe DTO para transferir as informações recebidas para as classes de *Service*, que por sua vez instanciam entidades que são classes do pacote *entity*. E por último realizam as operações no banco de dados através das classes de repositório *repository*.

Assim gerando a interligação dos recursos, tentando manter as responsabilidades separadas por pacotes e classes.

Figura 30 – Diagrama de pacotes do *backend* administrativo



Fonte: Acervo do autor (2023).

4.7.3.2 Sequência do processamento de inclusão

Com objetivo de explicar a sequência de passos da aplicação, e demonstrar melhor como as classes se relacionam, na Figura 32, é apresentado o diagrama de sequência da inclusão de um plano de documento.

Com o serviço iniciado, algum cliente irá realizar uma requisição utilizando o verbo HTTP *POST* para o endereço da aplicação na rota "/plan/new". É necessário o envio das informações que deseja constar no plano. Podem ser passadas todas as informações disponíveis no modelo de dados de planos de documentos. Exceto o ID que é criado pelo banco de dados. A estrutura JSON de exemplo pode ser vista na Figura 31.

Figura 31 – Estrutura JSON para inclusão de plano

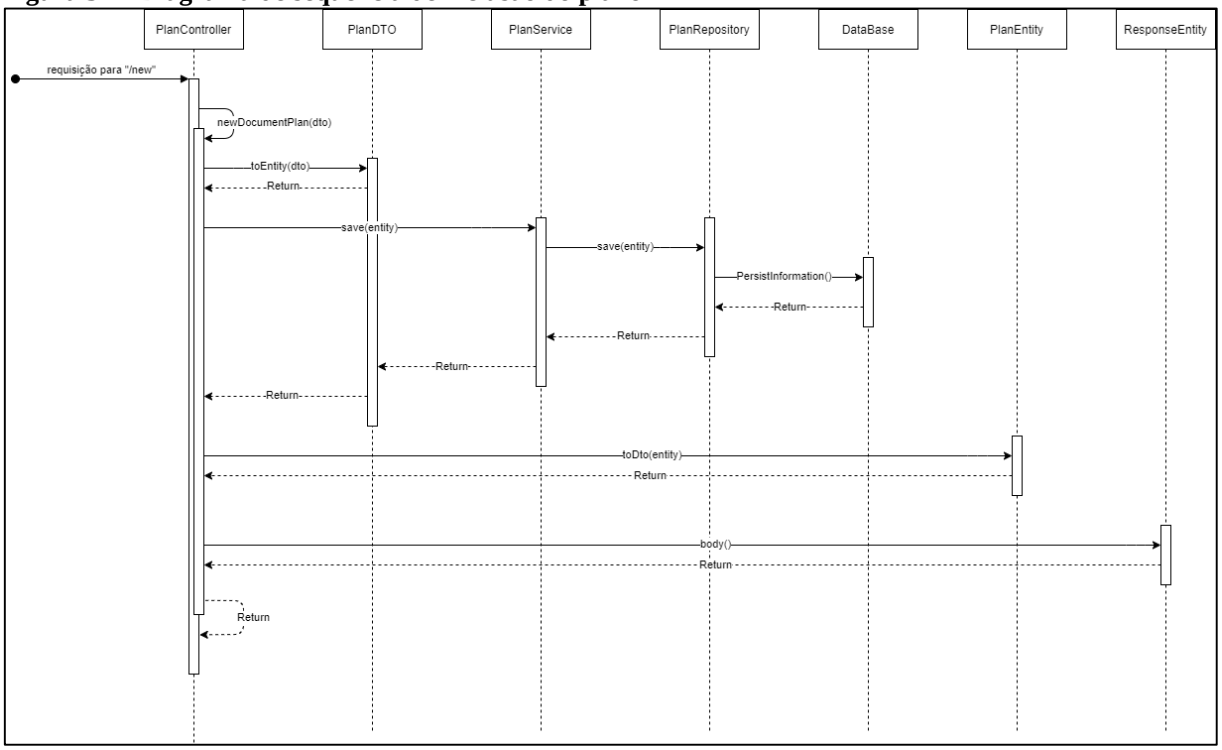
```
{
  "nome":"Uniedu",
  "criador":"Unidavi",
  "descricao":"Plano de documentos do Uniedu"
}
```

Fonte: Acervo do autor (2023).

Ao realizar a requisição conforme descrito, iniciará o processo mostrado na Figura 32. Onde o "PlanController" irá executar o método "newDocumentPlan" que está configurado para receber as requisições vindas da rota "/plan/new". O *Spring framework* realiza a montagem da classe "PlanDto" de acordo com os dados passados na requisição, então através do método "toEntity" do "PlanDto" monta-se o objeto de entidade que será enviado para a classe "PlanService" no método "save" que utilizará o "PlanRepository" para persistir a informação no banco de dados e retornar o objeto de entidade com todas as informações inseridas. Neste momento a entidade conta também com o ID gerado pelo banco de dados. Então essa entidade é convertida para DTO e então é retornada em formato JSON para o cliente que enviou a requisição.

Todas as operações *DML* do banco de dados possuem um fluxo semelhante a esse. Portanto isso se aplica para documentos, planos, exemplos de documentos e relacionamento entre documento e um plano.

Figura 32 – Diagrama de sequência de inclusão do plano



Fonte: Acervo do autor (2023).

4.7.4 Microsserviço Intermediário

Conforme já abordado na topologia da aplicação existe uma fila de processamentos que é gerenciada pelo serviço de fila do RabbitMq. Porém para que a fila funcione adequadamente é necessário que existam "*publishers*" e "*subscribers*" para essa fila. Ou seja deve existir algum serviço para realizar a publicação de conteúdos nessa fila e outros serviços para realizar a leitura e execução desses itens da fila, transformando eles em processos do sistema.

O serviço responsável por publicar esses itens na fila é o "*backend* intermediário" que justamente realiza o intermédio entre a aplicação visual na máquina do usuário e a publicação na fila de processamentos assíncronos. Porém não é somente necessário que esse serviço realize a publicação dos itens em fila, é necessário também salvar o conteúdo dos arquivos enviados pelo usuário no banco de dados de execução.

4.7.4.1 Classes e pacotes

O esquema de classes e pacotes do serviço intermediário respeita a estrutura necessária para realização de comandos *DML* no banco de dados como o serviço de *backend* principal.

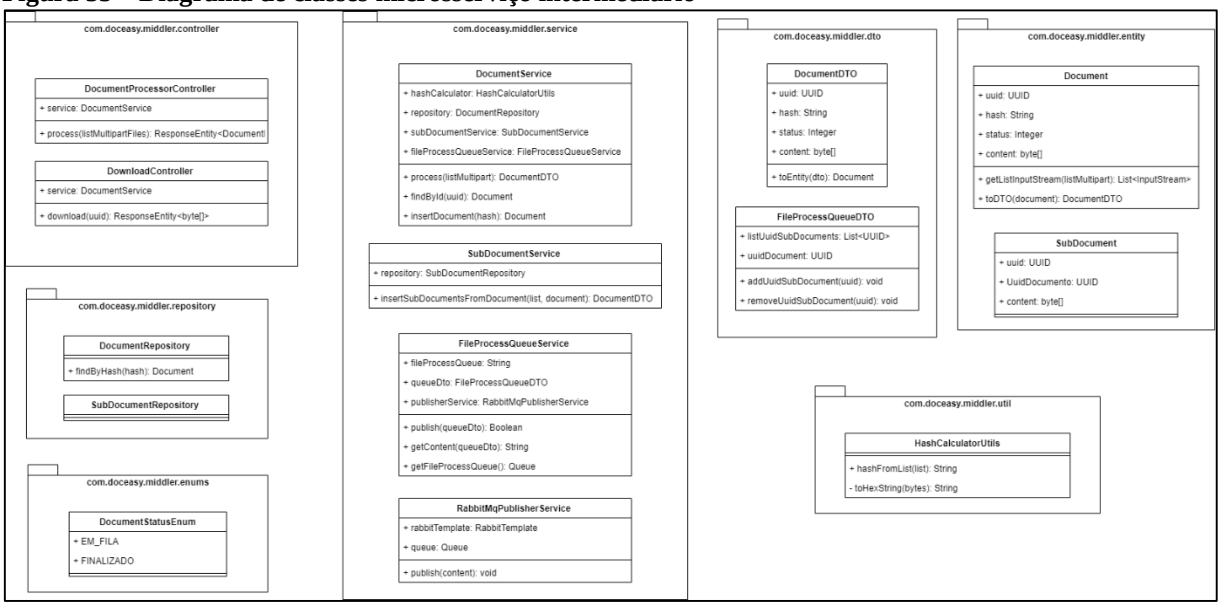
Porém nele existe algumas diferenças para implementar o serviço de publicação de itens em fila.

Como uma forma de facilitar a publicação de registros na fila, foi criado um conjunto de classes para realizar o processamento mais complexo da publicação. As classes envolvidas nesse processo são: “FileProcessQueueService”, “FileProcessQueueDto” e “RabbitMqPublisherService”.

A classe “FileProcessQueueService” é uma implementação que permite a publicação de um objeto da classe “FileProcessQueueDto” em uma fila de processamentos assíncronos. Esse objeto DTO possui duas informações que serão colocadas como um registro na fila. Elas são: O UUID do documento de resultado e uma lista de UUID dos sub documentos que estão relacionados. Assim é possível que o serviço que estiver observando a fila consiga saber quais documentos ele deve buscar e juntar para gerar o documento final, além de saber também onde ele deve armazenar o conteúdo desse documento de resultado.

Esse processo de publicação é possível pois a classe de serviço utiliza “RabbitMqPublisherService” que é a classe que utiliza as bibliotecas do RabbitMq para realizar a comunicação de fato com o servidor de fila. O diagrama de classes desse serviço pode ser observado na Figura 33.

Figura 33 – Diagrama de classes microsserviço intermediário



Fonte: Acervo do autor (2023).

4.7.4.2 Publicação em fila

A publicação dos registros na fila de processamentos foi implementada utilizando a biblioteca "*Spring AMQP*". Para utilizar a mesma é necessário adicionar o conteúdo presente na Figura 34, ao arquivo "pom.xml" da aplicação *Spring*.

Figura 34 – Dependência *Spring AMQP*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

Fonte: Acervo do autor (2023).

A adição desse recurso permite então que a aplicação consuma e publique registros na fila de processamentos. Porém é necessário informar no arquivo de configurações do projeto o endereço do servidor de fila. Isso pode ser visto na Figura 35, onde são listadas as informações do *RabbitMq* no arquivo de propriedades.

Figura 35 – Endereços do RabbitMq

```
## Rabbit
spring.rabbitmq.host=localhost
spring.rabbitmq.port=5672
spring.rabbitmq.username=guest
spring.rabbitmq.password=guest
mq.queues.files-to-process=files-to-process
```

Fonte: Acervo do autor (2023).

Após essas etapas e dado o contexto do serviço, que é o de realizar a publicação de registros para processamento posterior. É possível iniciar a implementação da camada de comunicação com o mecanismo de fila. Na Figura 36, é possível observar a implementação padrão de uma classe para realizar a publicação de conteúdos em fila. É necessário que ela tenha dois atributos. Um para armazenar o "RabbitTemplate" e outro para armazenar uma instância da Fila. A classe concreta da fila de processamento de arquivos, nesse caso a "FileProcessQueueService" é quem irá instanciar a fila e utilizar essa classe de comunicação, já o *template* o próprio *Spring* irá se encarregar.

Figura 36 – Implementação do Publisher

```java
package com.doceasy.middler.service;

import org.springframework.amqp.core.Queue;

@Data
@RequiredArgsConstructor
@Service
public class RabbitMqPublisherService {

    private final RabbitTemplate rabbitTemplate;
    private Queue queue;

    public void publish(String content) {
        rabbitTemplate.convertAndSend(queue.getName(), content);
    }

}
```

Fonte: Acervo do autor (2023).

A implementação da classe “FileProcessQueueService” pode ser observada na Figura 37. Onde é possível notar os métodos para converter o objeto em *string* para a publicação, assim como também a instanciação do objeto da fila de processamentos. Note que o valor do atributo “fileProcessQueue” é buscado do arquivo de configurações do projeto, ou seja, é valor que foi definido anteriormente no processo de configuração.

Figura 37 – Implementação de FileProcessQueueService

```java
package com.doceasy.middler.service;

import org.springframework.amqp.core.Queue;

@Data
@Service
public class FileProcessQueueService {

    @Value("${mq.queues.files-to-process}")
    private String fileProcessQueue;

    private FileProcessQueueDTO queueDto;

    @Autowired
    private RabbitMqPublisherService publisherService;

    public Boolean publish(FileProcessQueueDTO queueDto) throws JsonProcessingException {
        String content = this.getContent(queueDto);

        publisherService.setQueue(this.getFileProcessQueue());
        publisherService.publish(content);

        return true;
    }

    private String getContent(FileProcessQueueDTO queueDto) throws JsonProcessingException {
        ObjectMapper mapper = new ObjectMapper();
        String content = mapper.writeValueAsString(queueDto);

        return content;
    }

    /**
     * Retorna o objeto da fila.
     * @return
     */
    public Queue getFileProcessQueue() {
        return new Queue(fileProcessQueue, true);
    }

}
```

Fonte: Acervo do autor (2023).

4.7.4.3 Fluxo da aplicação

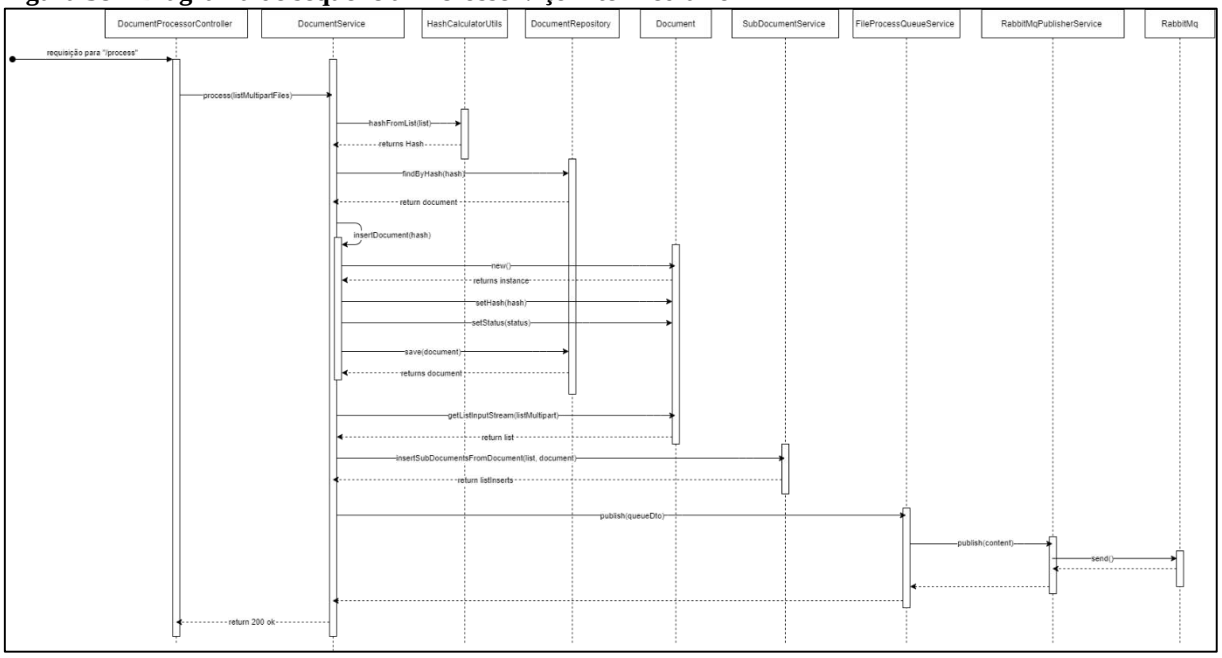
A sequência de operações realizadas pelo serviço intermediário para publicar os itens na fila pode ser vista na Figura 38.

O processamento começa no controlador que recebe a requisição enviada pelo usuário para realizar o processamento dos arquivos. Após receber os arquivos e parâmetros adequadamente o controlador utiliza o serviço de documentos para continuar com o processamento, ele por sua vez, primeiro realiza o cálculo do *hash* SHA256 dos arquivos enviados através da classe “HashCalculatorUtils”, após calcular o *hash* é consultado no banco de dados através da classe de repositório de documentos, se a sequência atual de documentos já foi enviada antes.

Isso evita que o usuário confirme duas vezes a mesma tela com os mesmos documentos e seja então processada a junção deles duas ou mais vezes desnecessariamente. Caso não existir um documento com o mesmo *hash* é então criado um registro para salvar o documento de resultado do processo de junção. Também são salvos e criados registros para cada um dos sub documentos envolvidos no processamento.

Após isso é utilizada a classe “FileProcessQueueService”, onde são enviados os objetos de transferência de informações com a lista de documentos que devem ser juntados, nessa classe esse objeto é convertido para uma representação JSON dessa estrutura, e então através do serviço de publicação na fila esse JSON é inserido na fila de processamentos.

Figura 38 – Diagrama de sequência microsserviço intermediário



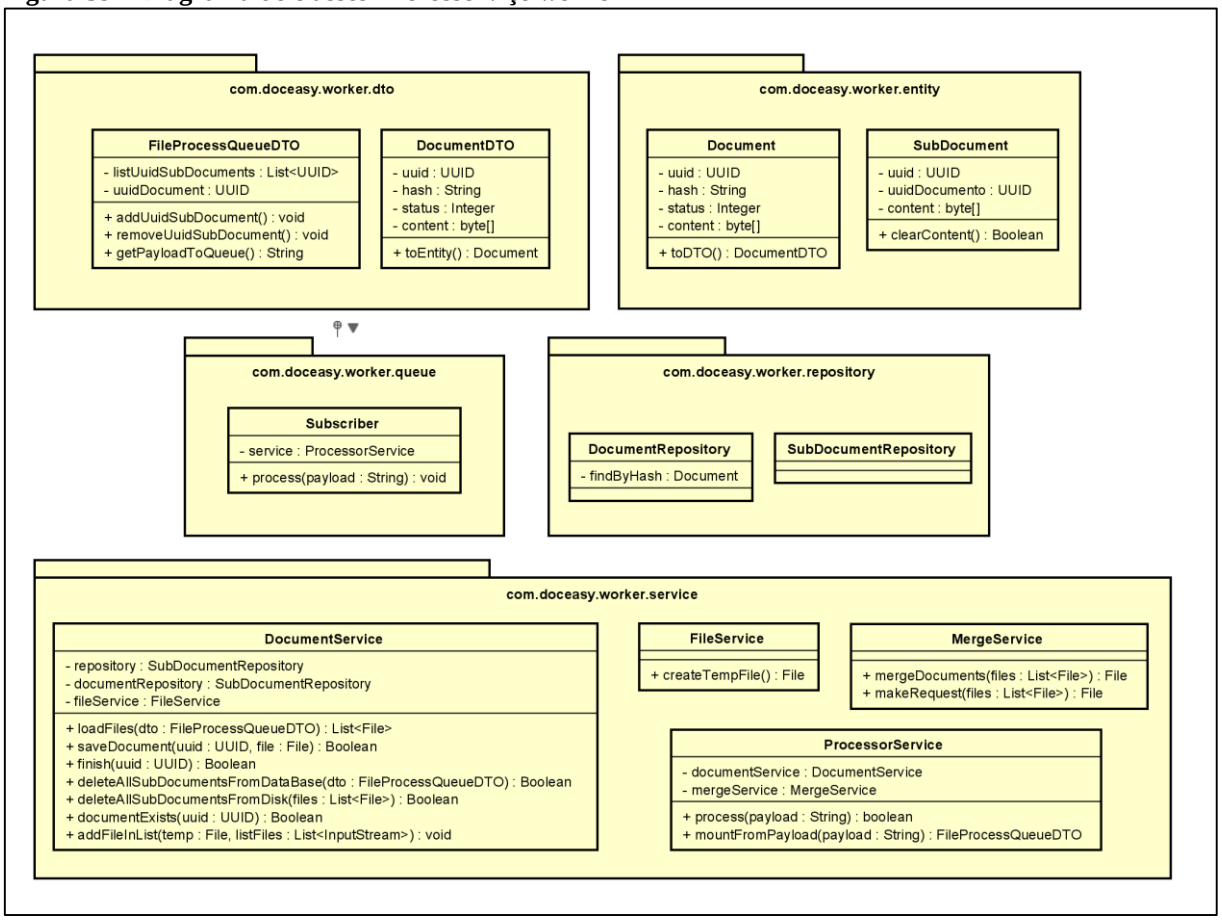
Fonte: Acervo do autor (2023).

4.7.5 Microsserviço Worker

A fila de processamentos assíncronos necessita de um serviço para realizar a leitura dos registros da mesma e então realizar o processamento que aquele registro tem como objetivo. Esse tipo de serviço é conhecido como "*subcriber*", o papel dele é iniciar o processamento assim que um registro for adicionado na fila, executando os registros um por um.

Esse serviço é composto também pela estrutura padrão de classes para manipulação dos dados no banco de dados. A sua diferença está nas classes para comunicar-se com a fila e nas chamadas para o serviço responsável por efetivamente juntar os documentos. Essas classes estão contidas nos pacotes "com.doceasy.worker.queue" e "com.doceasy.worker.service" que podem ser vistos na Figura 39.

Figura 39 – Diagrama de classes microsserviço worker



Fonte: Acervo do autor (2023).

4.7.5.1 Inscrição na Fila de processamento

A leitura dos registros da fila é realizada através da inscrição do serviço em uma fila do *RabbitMQ*. A sua implementação assim como já abordado na publicação de registros é realizada utilizando a biblioteca de AMQP do *Spring* e o arquivo de configuração do projeto. Porém neste caso é somente necessário anotar um método da classe que irá ouvir a fila com o "*@RabbitListener*" informando qual fila ele deve escutar. Um exemplo pode ser visto na Figura 40.

Figura 40 – Implementação de ouvinte

```java
package com.doceasy.worker.queue;

import org.slf4j.Logger;

@Service
public class Subscriber {

    private Logger log = LoggerFactory.getLogger(Subscriber.class);

    @Autowired
    private ProcessorService service;

    /**
     * Recebe as informações do registro da fila de processamento e
     * encaminha para o serviço adequado.
     * @param payload
     * @throws InterruptedException
     */
    @RabbitListener(queues = "${mq.queues.files-to-process}")
    public void process(@Payload String payload) throws InterruptedException {
        log.info("iniciando processamento de registro da fila");
        service.process(payload);
        log.info("finalizando o processamento de registro da fila");
    }

}
```

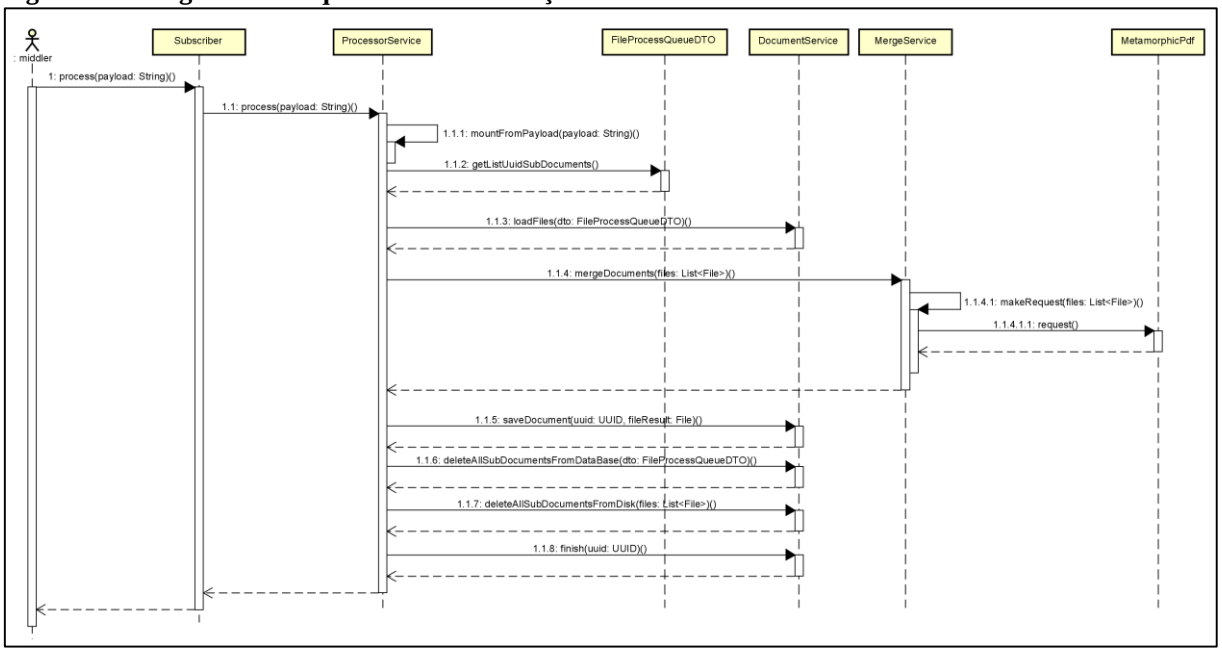
Fonte: Acervo do autor (2023).

4.7.5.2 Fluxo da aplicação

Após a publicação de um registro em fila a aplicação executa o processamento para realizar a junção dos documentos. Esse processo pode ser visto através da Figura 41. Onde o serviço recebe as informações da fila através da classe "Subscriber", essas informações são passadas para o "ProcessorService" que realiza a montagem da estrutura JSON vinda da fila para objetos em memória. Após isso é necessário carregar os arquivos que devem ser juntados em memória para que eles possam ser enviados ao serviço de junção. Através do método "mergeDocuments" da classe "MergeService" os arquivos são enviados para o outro serviço e são juntados em apenas um documento. Esse conteúdo binário do arquivo é salvo no registro

do documento de resultado criado anteriormente pelo serviço intermediário. Após isso são removidos os conteúdos dos arquivos do disco e finalizado o processamento.

Figura 41 – Diagrama de sequência microsserviço worker

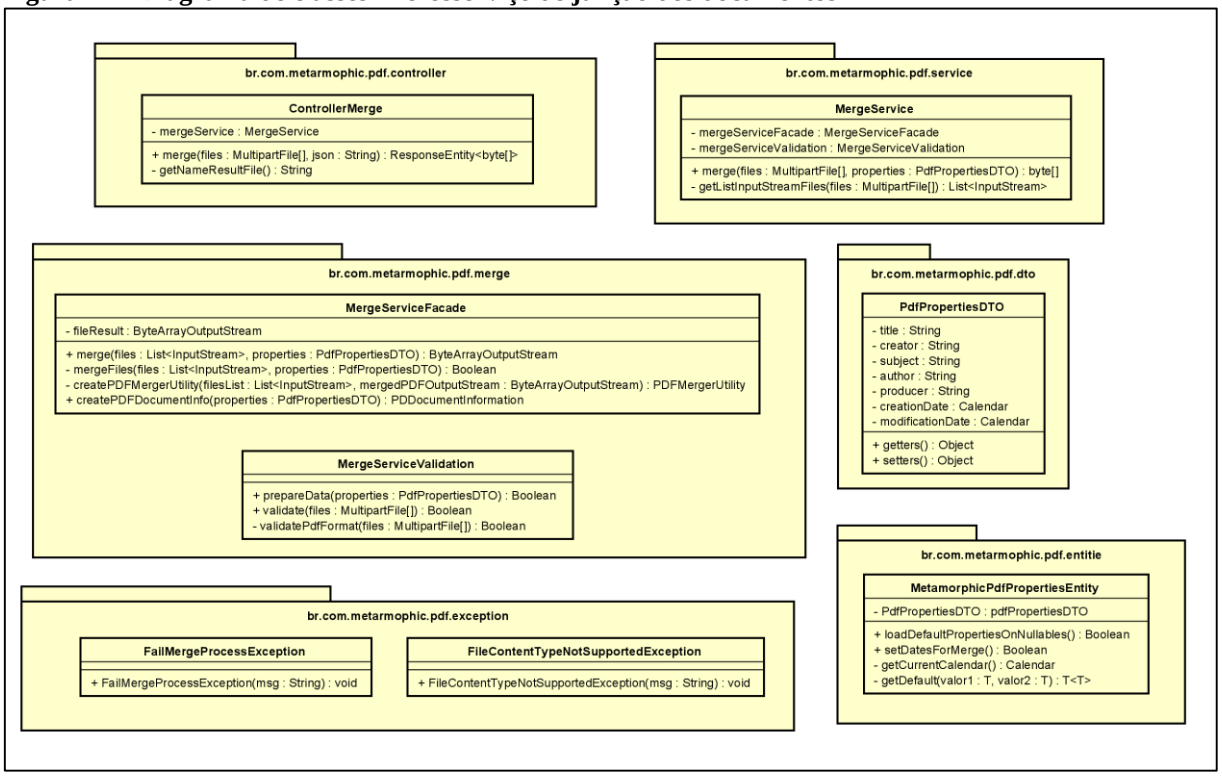


Fonte: Acervo do autor (2023).

4.7.6 Microsserviço para junção de documentos

O serviço responsável por realizar as junções dos documentos possui uma estrutura de pacotes e classes semelhante as utilizadas pelos outros serviços. Ela pode ser vista na Figura 42. Esse serviço é composto por 6 pacotes e 8 classes. A diferença está principalmente no pacote “br.com.metamorphic.pdf.merge”, pois nele estão contidas as classes de fachada para utilização dos recursos da biblioteca PDFBox, que foi utilizada para realizar a junção dos arquivos PDF. Essas classes são consumidas pela classe “MergeService” que é utilizada pelo controlador para realizar o processamento. Assim a camada de serviço não fica acoplada aos recursos da biblioteca diretamente, pois existem classes no meio para facilitar a manutenção futura.

Figura 42 – Diagrama de classes microsserviço de junção dos documentos



Fonte: Acervo do autor (2023).

4.7.6.1 Utilização do PDFBox

Para realizar a junção dos documentos foi implementado um algoritmo utilizando PDFBox, que é uma biblioteca Java especializada em manipulação de documentos PDF. Um dos recursos disponibilizados pela biblioteca, é exatamente a junção de documentos PDF. Dentro estrutura de classes do sistema, foi realizada a criação do pacote “Merge” para conter essa camada de junção dos documentos separada do restante da lógica de domínio da aplicação.

Como uma forma de criar uma interface para o restante da aplicação foi realizada a criação da classe “MergeServiceFacade” que atua como uma fachada para que o restante da aplicação possa utilizar os recursos do PDFBox através dessa classe e não diretamente da implementação das classes do PDFBox. O código de implementação da mesma consiste em instanciar algumas classes do PDFBox que permitem realizar a junção dos documentos. Ele foi elaborado utilizando a documentação do PDFBox. Isso pode ser observado na Figura 43.

Figura 43 – Implementação da junção

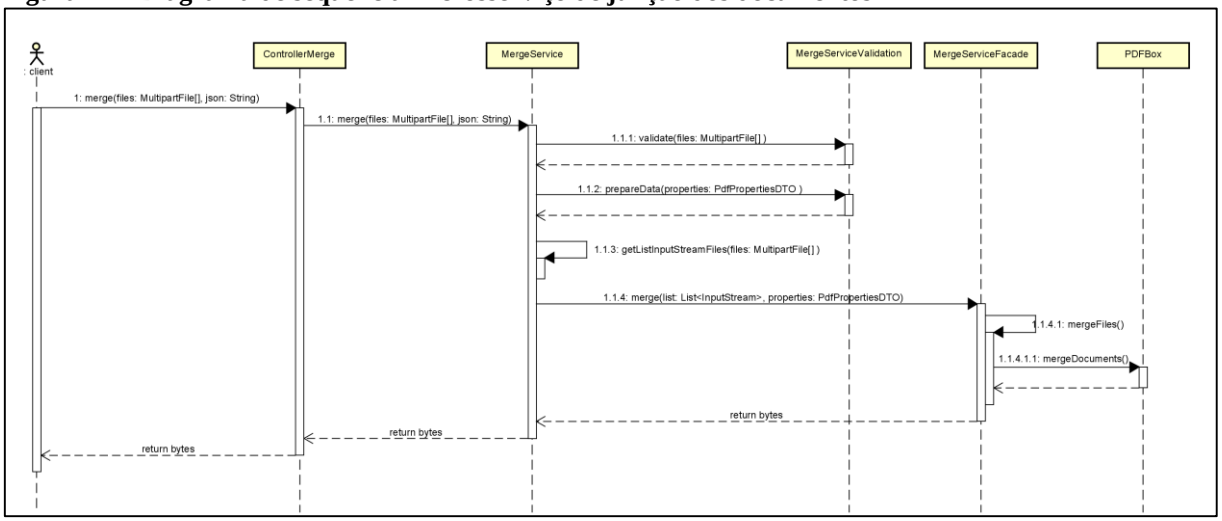
```
/**
 * Método principal para realizar o merge de dois documentos.
 * @param file1
 * @param file2
 * @return Boolean - Indica se o processamento ocorreu com sucesso.
 */
private Boolean mergeFiles(List<InputStream> files, PdfPropertiesDTO properties) {
    //TODO Implementar um esquema de logs das operações executadas.
    log.debug("Instanciando recursos para juncao.");

    try (ByteArrayOutputStream mergedPDFOutputStream = new ByteArrayOutputStream())
    {
        PDFMergerUtility pdfMerger = this.createPDFMergerUtility(files, mergedPDFOutputStream);
        PDDocumentInformation pdfDocumentInfo = this.createPDFDocumentInfo(properties);
        pdfMerger.setDestinationDocumentInformation(pdfDocumentInfo);

        log.debug("Juntando " + files.size() + " arquivos");
        pdfMerger.mergeDocuments(MemoryUsageSetting.setupMixed(MEMORY_QUANTITY));

        this.setFileResult(mergedPDFOutputStream);
    }
    catch (Exception e)
    {
        this.setFileResult(null);
        log.debug("Problema ao realizar o merge");
        log.debug(e.getMessage());
        if (log.isDebugEnabled()) {
            e.printStackTrace();
        }
        return false;
    }
    finally
    {
        files.forEach(IOUtils::closeQuietly);
    }

    return true;
}
```

Fonte: Acervo do autor (2023).

4.7.6.2 Fluxo da aplicação

O processo de junção dos documentos inicia na classe “ControllerMerge” que posteriormente passa os documentos que devem ser unidos para o “MergeService” que através do “MergeServiceFacade” realiza a utilização dos recursos do PDFBox para unir os documentos. Após a junção dos mesmos ele retorna o conteúdo binário do arquivo resultante desse processo para quem realizou a requisição da junção. Esse processo pode ser visto na Figura 44.

Figura 44 – Diagrama de sequência microsserviço de junção dos documentos



Fonte: Acervo do autor (2023).

4.7.7 Streaming de logs

Os serviços *backend*, envolvidos no processo de junção dos documentos possuem uma funcionalidade voltada a monitoramento das aplicações que é o *streaming* de logs. Para que essas aplicações consigam realizar isso é necessário adicionar algumas bibliotecas no projeto de cada serviço e realizar uma configuração.

4.7.7.1 Bibliotecas

Para permitir o funcionamento correto do *streaming* de logs são necessárias duas bibliotecas principais. O LogBack que é a biblioteca que permite implementar mecanismos personalizados de log para a aplicação, essa biblioteca já está contida no pacote WEB do *Spring framework*. Este que vem por padrão em projetos WEB que utilizam o Framework. A segunda biblioteca é uma implementação de LogBack para o servidor de logs Loki, a configuração da Figura 45, deve ser adicionada ao arquivo pom.xml do projeto, para que os recursos da biblioteca possam ser utilizados.

Figura 45 – Repositório Maven loki appender

```
<dependency>
        <groupId>com.github.loki4j</groupId>
        <artifactId>loki-logback-appender</artifactId>
        <version>1.4.1</version>
</dependency>
```

Fonte: Acervo do autor (2023).

4.7.7.2 Configuração

Para que a aplicação utilize o *appender* para enviar os logs para o servidor Loki é necessário adicionar na pasta de recursos do projeto um arquivo “logback-spring.xml”. Esse arquivo irá conter informações sobre o servidor Loki e também sobre os *appenders* de log. Os *appenders* são basicamente os meios de coleta dos logs. Conforme a Figura 46, é possível configurar nesse arquivo um *appender* para o console da aplicação.

Figura 46 – logback appender

```
<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
  <encoder>
    <pattern>
      %d{HH:mm:ss.SSS} %-5level %logger{36} %X{X-Request-ID} - %msg%n
    </pattern>
  </encoder>
</appender>
```

Fonte: Acervo do autor (2023).

Desse mesmo modo é possível configurar um *appender* que irá enviar as informações para o servidor Loki. Isso pode ser visto na Figura 47. Na *tag appender* é possível definir a classe que será responsável pelo *appender*, o valor “com.github.loki4j.logback.Loki4jAppender” é de uma classe da biblioteca LogBack para Loki. Ainda é possível definir na *tag* “HTTP” e “URL” o endereço do servidor de logs remoto. Assim os logs serão enviados para esse endereço para que sejam armazenados e posteriormente visualizados.

Figura 47 – logback loki appender

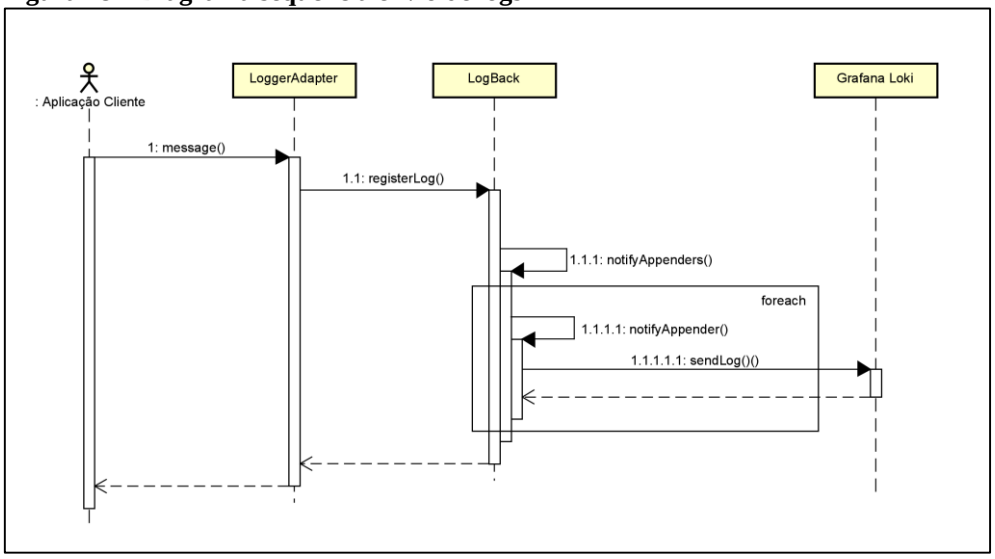
```
<appender name="LOKI" class="com.github.loki4j.logback.Loki4jAppender">
  <!-- (1) -->
  <http>
    <url>http://localhost:3100/loki/api/v1/push</url>
  </http>
  <format>
    <!-- (2) -->
    <label>
      <pattern>app=${name},host=${HOSTNAME},level=%level</pattern>
      <!-- (3) -->
      <readMarkers>true</readMarkers>
    </label>
    <message>
      <!-- (4) -->
      <pattern>
                    {"level":"%level","class":"%logger{36}","thread":"%thread","message": "%message","requestId": "%X{X-Request-ID}"}
      </pattern>
    </message>
  </format>
</appender>
```

Fonte: Acervo do autor (2023).

4.7.7.3 Fluxo do envio de logs

A Figura 48, demonstra o fluxo que os logs percorrem para chegar ao servidor Loki. Ao realizar log de alguma informação, existe uma camada de adaptadores de log fornecida pelo LogBack, esses adaptadores encaminham os logs para as classes do LogBack, onde os *appenders* configurados em "logback-spring.xml" *serão* notificados. Ao realizar essa notificação é instanciada a classe da configuração e então internamente nela é realizado o envio das informações de logs para o servidor Loki remoto.

Figura 48 – Diagrama sequência envio de logs



Fonte: Acervo do autor (2023).

4.7.8 Monitoramento de Estatísticas

Os serviços que implementam o *streaming* de logs também implementam mecanismos para permitir monitorar informações das aplicações, como quantidades de requisições que foram recebidas em um determinado *end-point*, tempo médio das requisições entre outras informações. Esse monitoramento é possível quando se implementa algumas bibliotecas para realizar a disponibilização dessas informações via requisições HTTP.

4.7.8.1 Bibliotecas

O funcionamento das estatísticas é baseado na busca de informações da aplicação via *end-points* HTTP. Para disponibilizar informações sobre a aplicação de maneira padronizada e sem necessidade de implementar passo a passo, pode-se utilizar as bibliotecas da Figura 49, ao adicionar elas ao projeto através do arquivo "pom.xml", é possível configurar para que sejam disponibilizadas informações sobre a aplicação via HTTP.

O *Spring actuator* permite a disponibilização de informações via requisições HTTP, e quando utilizado em conjunto com o "micrometer-registry-prometheus" é possível disponibilizar as informações no padrão que o Prometheus espera.

Figura 49 – Dependências monitoramento

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
        <scope>runtime</scope>
</dependency>
```

Fonte: Acervo do autor (2023).

Após adicionar as dependências, no arquivo de configurações do projeto, conforme a Figura 50, é possível definir quais *end-points* e informações estarão disponíveis. Ao colocar "*" a aplicação entende que tudo deve ficar exposto para coletas.

Figura 50 – configuração do *spring actuator*

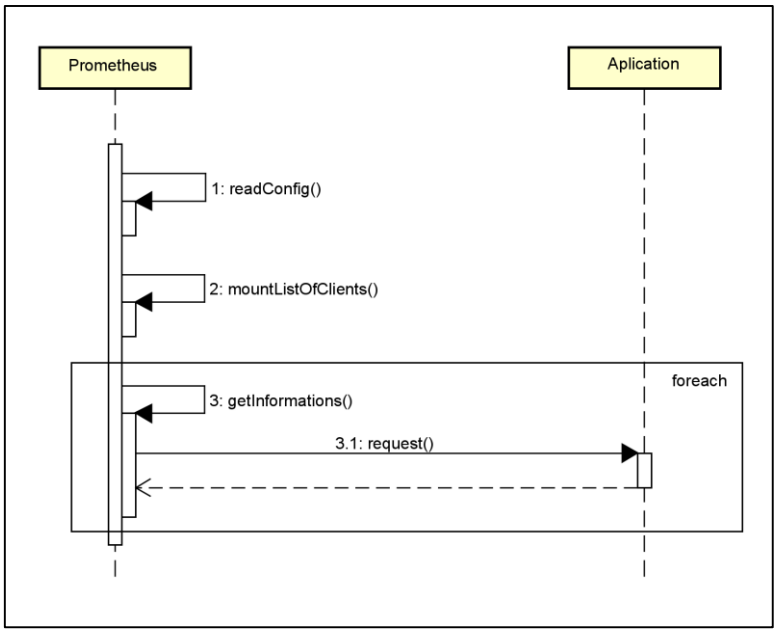
```
## Monitoring
management:
  endpoints:
    web:
      exposure:
        include: '*'
```

Fonte: Acervo do autor (2023).

4.7.8.2 Sequência da coleta de informações

O serviço do Prometheus realiza coletas de tempos em tempos em cima de *end-points* que ele foi configurado para buscar. Na Figura 51, é possível visualizar como o processo funciona. Ao iniciar o serviço do Prometheus ele lê o arquivo de configurações e monta a lista de *end-points* que ele deve coletar. Desse modo ele entra em loop, realizando essas coletas no intervalo de tempo definido, realizando as coletas e armazenando as informações.

Figura 51 – Diagrama sequência coleta de métricas



Fonte: Acervo do autor (2023).

4.7.9 Componentes reativos

A parte visual da aplicação, foi desenvolvida utilizando a biblioteca JavaScript conhecida como VueJs, que permite a criação de componentes reativos. Esses reagem as alterações nas informações relevantes da tela, como: número de registros, valores informados pelo usuário, etc. Permitindo atualização dos componentes da tela em tempo real, e sem a necessidade de recarregar todo o conteúdo da página.

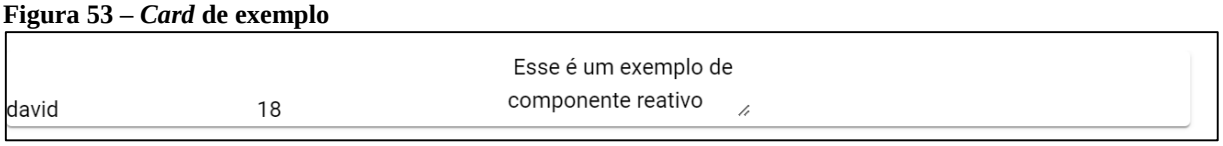
O código fonte de um componente de *card* utilizando VueJs, pode ser visto na Figura 52. Na *tag* "*template*" é definido o HTML do componente. E na *tag* "*script*" é definido o comportamento e dados do componente. Além disso o resultado do código da Figura 52, pode ser visto na Figura 53. Os componentes se chamam reativos pois ao modificar o valor de uma das variáveis utilizadas o componente é automaticamente reconstruído. Além disso também podem ser definir estilos CSS nos componentes.

Figura 52 – Implementação exemplo de componente

```
<template>

    <div>
        <v-card width="50vw">
            <input type="text" :value="this.name">
            <input type="number" :value="this.age">
            <textarea> {{ this.content }}</textarea>
        </v-card>
    </div>

</template>

<script>

export default {
    data() {
        return {
            name: 'david',
            age: '18',
            content: 'Esse é um exemplo de componente reativo'
        }
    }
}

</script>
```

Fonte: Acervo do autor (2023).

Figura 53 – *Card* de exemplo



Fonte: Acervo do autor (2023).

5. CONCLUSÃO

O desenvolvimento do projeto possibilitou a visualização de uma aplicação WEB para centralizar informações sobre documentos e ainda permitir a junção dos mesmos. Também contando com recursos de monitoramento e segurança, que podem garantir o não vazamento de informações dos documentos dos usuários. Isso graças a possibilidade de hospedagem da aplicação em uma infraestrutura controlada pela empresa ou entidade interessada.

Através das áreas do sistema é possível que um usuário administrador consiga além de monitorar a execução e funcionamento da aplicação, também definir os documentos e as informações para montagem dos mesmos. O que facilita para o usuário final encontrar as informações de forma centralizada na plataforma, e ainda através das funcionalidades de junção montar os documentos que são compostos por vários outros. Gerando uma melhora significativa em todo o processo de montagem de documentos.

Para fazer com que o projeto cumprisse seus objetivos específicos foram necessários levantamentos e estudos em formas de como através da tecnologia seria possível resolver tais problemas e fornecer uma aplicação distribuída e segura.

Seguindo o fluxo de desenvolvimento e pensando no primeiro objetivo, foi realizado com sucesso o levantamento dos requisitos funcionais e não funcionais que a aplicação deveria possuir.

Alcançar o primeiro objetivo específico, permitiu entender melhor o escopo da aplicação bem como as tecnologias que seriam necessárias para o desenvolvimento. Sendo assim através da construção da literatura e diagramas de topologia da aplicação foi possível alcançar com sucesso a definição de tecnologias para o desenvolvimento.

O terceiro objetivo específico visava efetivamente realizar o desenvolvimento e utilização dessas tecnologias para montar uma aplicação distribuída e monitorável, seguindo os requisitos e objetivos do sistema. Este foi realizado com sucesso, e pode ser visto através da seção de protótipos da aplicação, bem como diagramas e demonstrações de código presentes na elaboração do trabalho.

O quarto objetivo foi alcançado com sucesso pois através da arquitetura do sistema e das tecnologias utilizadas, é possível facilmente executar a aplicação em uma infraestrutura própria da entidade interessada no projeto. O que garante que os dados armazenados no banco de dados não serão utilizados para outros fins desconhecidos pelo usuário da aplicação.

Diante do exposto é possível concluir que tanto os objetivos específicos como o objetivo geral de prototipar uma aplicação WEB para facilitar o processo de montagem de documentos digitais foram atingidos com sucesso ao desenvolvimento do trabalho.

5.1 TRABALHOS FUTUROS

Mesmo atingindo todos os objetivos propostos, existem muitas melhorias que podem ser implementadas no projeto com o decorrer do tempo. Portanto para a continuidade desde trabalho existem passos para evoluir o sistema.

A primeiro seria implementação de uma estrutura de agendamentos ou tempo de vida para os arquivos dentro da plataforma. Pois atualmente o arquivo final gerado é mantido permanentemente, mesmo após a realização do *download* pelo usuário.

Em segundo lugar o armazenamento de arquivos foi desenvolvido pensado para ser realizado em um banco de dados. Porém existem maneiras e ferramentas especializadas em armazenamentos de arquivos. Um exemplo é a ferramenta MINIO S3, que se trata de um servidor para armazenamento de objetos. Essa alteração no armazenamento pode trazer mais robustez e performance para o sistema como um todo. Relacionado a isso ainda existe a ausência do recurso de visualizar o documento de exemplo, que seria importante ser adicionado em uma versão futura.

A terceira melhoria está relacionada com a implementação de um tutorial de usabilidade para novos usuários no sistema. Existem também ferramentas como o Driver.js que permitem a criação de *tours* de apresentação de recursos de interface para sites. Um usuário não muito familiarizados com tecnologia podem se beneficiar dessa funcionalidade.

Como quarta melhoria seria a implementação de fato do sistema de *login* com tipos de usuários. O que permitiria de fato a criação de administradores e usuários finais. Com limitações de acessos e etc. Iria acrescentar segurança a aplicação além dar a capacidade de personalizar melhor a visualização de acordo com o perfil do usuário.

A última recomendação seria a implementação de um servidor de cache para as informações dos documentos. Atualmente elas sempre são buscadas do banco de dados administrativo, porém, o uso em larga escala, pode acabar gerando gargalos de performance no banco de dados. Um servidor de cache facilitaria o funcionamento sobre alta demanda.

REFERÊNCIAS

ALVES, William P. **Banco de Dados.** São Paulo: Editora Saraiva, 2014. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788536518961/. Acesso em: 21 abr. 2023.

ALVES, William P. **HTML & CSS:** aprenda como construir páginas web. São Paulo: Editora Saraiva, 2021. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786558110187/. Acesso em: 21 abr. 2023.

AWS. **O que são microsserviços?** 2023. Disponível em: <https://aws.amazon.com/pt/microservices/>. Acesso em: 19 de mai. de 2023.

AWS. **O que é uma API?** 2023. Disponível em: <https://aws.amazon.com/pt/what-is/restful-api/>. Acesso em: 20 de mai. de 2023.

CGI.BR. **Proporção de empresas brasileiras que venderam pela Internet cresce durante a pandemia e chega a 73%, revela pesquisa do CGI.br.** 2022. Disponível em: <https://www.cgi.br/noticia/releases/proporcao-de-empresas-brasileiras-que-venderam-pela-internet-cresce-durante-a-pandemia-e-chega-a-73-revela-pesquisa-do-cgi-br/#:~:text=A%20propor%C3%A7%C3%A3o%20de%20empresas%20que,setor%20de%20informa%C3%A7%C3%A3o%20e%20comunica%C3%A7%C3%A3o.>. Acesso em: 18 de jun. de 2023.

DOCKER. **O que é container?** 2023. Disponível em: <https://www.docker.com/resources/what-container/ >. Acesso em: 21 de abril de 2023.

ECLIPSE FOUNDATION. **IDEs de área de trabalho.** 2023. Disponível em: <https://www.eclipse.org/ide/>. Acesso em: 21 de mai. de 2023.

ECLIPSE FOUNDATION. **Sobre a Fundação Eclipse.** 2023. Disponível em: <https://www.eclipse.org/org/>. Acesso em: 21 de mai. de 2023.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software - Produtos - Vol.1.** Rio de Janeiro: Grupo GEN, 2019. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788521636724/. Acesso em: 31 mar. 2023.

FLANAGAN, David. **JavaScript:** o guia definitivo. Porto Alegre: Grupo A, 2013. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788565837484/. Acesso em: 21 abr. 2023.

FOROUZAN, Behrouz A. **Protocolo TCP/IP.** Porto Alegre: Grupo A, 2010. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788563308689/. Acesso em: 20 mai. 2023.

FOWLER, Martin. **UML essencial.** Porto Alegre: Grupo A, 2011. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788560031382/. Acesso em: 14 abr. 2023.

GRAFANA. **Painel de qualquer coisa. Observe tudo.** 2023. Disponível em: <https://grafana.com/grafana/>. Acesso em: 23 de out. de 2023.

GRAFANA LABS. **Grafana Loki.** 2023. Disponível em: <https://grafana.com/oss/loki/>. Acesso em: 30 de out. de 2023.

GIT. **Sobre.** 2023. Disponível em: <https://git-scm.com/>. Acesso em: 21 de abril de 2023.

HEMRAJANI, Anil. **Desenvolvimento ágil em Java com Spring, Hibernate e Eclipse.** São Paulo: Pearson Prentice Hall, 2007.

HORSTMANN, Cay. **Conceitos de Computação com Java.** Porto Alegre: Grupo A, 2009. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788577804078/. Acesso em: 10 mar. 2023.

INCAU, Caio. **Vue.js construa aplicações incríveis.** São Paulo: Casa do código, 2019.

LUGARH. **Seu RH digital com admissão rápida e segura.** 2023. Disponível em: <https://lugarh.com.br/>. Acesso em: 2 de nov. de 2023.

LUGARH. **EMPLOYER ORGANIZAÇÃO DE RECURSOS HUMANOS.** 2023. Disponível em: <https://azuremarketplace.microsoft.com/pt-br/marketplace/apps/employer-tecnologia-1453055.lugarh_?tab=overview>. Acesso em: 2 de nov. de 2023.

MAVEN. **O que é Maven?** 2023. Disponível em: <https://maven.apache.org/what-is-maven.html>. Acesso em: 7 de mai. de 2023.

MDN. **Uma visão geral do HTTP.** 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>. Acesso em: 20 de mai. de 2023.

MEDEIROS, Ernani Sales de. **Desenvolvendo software com UML 2.0:** definitivo. São Paulo: Pearson Makron Books, 2004.

MILETTO, Evandro M.; BERTAGNOLLI, Silvia C. **Desenvolvimento de software II:** introdução ao desenvolvimento web com HTML, CSS, javascript e PHP. (Tekne). Porto Alegre : Grupo A, 2014. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582601969/. Acesso em: 07 abr. 2023.

OLIVEIRA, Cláudio Luís V.; ZANETTI, Humberto Augusto P. **JAVASCRIPT DESCOMPLICADO - PROGRAMAÇÃO PARA WEB, IOT E DISPOSITIVOS MÓVEIS.** São Paulo: Editora Saraiva, 2020. Disponível em: <https://integrada.minhabiblioteca.com.br/#/books/9788536533100/>. Acesso em: 21 abr. 2023.

PDFBOX. **Uma biblioteca Java PDF?** 2023. Disponível em: <https://pdfbox.apache.org/>. Acesso em: 19 de mai. de 2023.

PIPEFY. **Conheça a Tecnologia No-Code.** 2023. Disponível em: <https://www.pipefy.com/pt-br/introducao-no-code/>. Acesso em: 2 de nov. de 2023.

PIPEFY. **Como o Pipefy funciona?** 2023. Disponível em: <https://www.pipefy.com/pt-br/como-pipefy-funciona/>. Acesso em: 2 de nov. de 2023.

POSTGRESQL. **Sobre.** 2023. Disponível em: <https://www.postgresql.org/about/ >. Acesso em: 21 de abril de 2023.

PROMETHEUS. **Visão Geral.** 2023. Disponível em: <https://prometheus.io/docs/introduction/overview/>. Acesso em: 30 de out. de 2023.

RABBITMQ. **RabbitMQ é o corretor de mensagens de código aberto mais amplamente implementado.** 2023. Disponível em: <https://www.rabbitmq.com/#features>. Acesso em: 23 de out. de 2023.

RED HAT. **O que é API REST?** 2023. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api>. Acesso em: 20 de mai. de 2023.

ROMERO, DANIEL. **Containers com Docker.** São Paulo: Casa do Código, 2015.

SPRING IO. **Over View.** 2023. Disponível em: <https://docs.spring.io/spring-framework/docs/current/reference/html/index.html >. Acesso em: 21 de abril de 2023.

SPRING IO. **Sprint Boot.** 2023. Disponível em: <https://spring.io/projects/spring-boot>. Acesso em: 21 de abril de 2023.

SOMMERVILLE, Ian. **Engenharia de Software.** São Paulo: Pearson Prentice Hall, 2011.

WINDER, Russel; GRAHAM, Roberts. **Desenvolvendo Software em Java**. Rio de Janeiro: Grupo GEN, 2009. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9/. Acesso em: 10 mar. 2023.