INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

CAMPUS MONTEIRO

CST EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

VICTOR FERNANDO VIANA DE MACÊDO

PEDRO LUCAS GONZAGA PRATA

JORDIELSON EMANUEL CALDEIRA DA SILVA

TRABALHO DE CONCLUSÃO DE CURSO

Imagem Pipeline: Sistema de pipeline para processamento de imagens

MONTEIRO

2023

VICTOR FERNANDO VIANA DE MACËDO
PEDRO LUCAS GONZAGA PRATA
JORDIELSON EMANUEL CALDEIRA DA SILVA

TRABALHO DE CONCLUSÃO DE CURSO

Imagem Pipeline: Sistema de pipeline para processamento de imagens

Trabalho de Conclusão de Curso (TCC) apresentado ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Campus Monteiro, formatado na Modalidade Projeto de Implementação, como pré-requisito para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas, sob orientação do Prof. M.Sc. Renata França de Pontes.

MONTEIRO
2023

Dados Internacionais de Catalogação na Publicação – CIP
Bibliotecária responsável Porcina Formiga dos Santos Salgado campus IFPB Monteiro.
CRB15/204

M141i Macedo, Victor Fernando Viana de; Prata, Pedro Lucas Gonzaga; Silva, Jordielson Emanuel Caldeira da.

Imagem Pipeline: sistema de pipeline para processamento de imagens / Victor Fernando Viana de Macedo; Pedro Lucas Gonzaga Prata; Jordielson Emanuel Caldeira da Silva – Monteiro-PB. 2023.

107fls. : il.

TCC (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - IFPB campus, Monteiro.

Orientadora: Profª. Msc. Renata França de Pontes

1. Imagem - Processamento 2. Software - Desenvolvimento 3. Pipeline - Sistema I. Título.

CDU 004.932:004.453

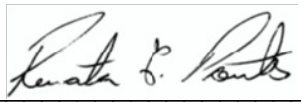
VICTOR FERNANDO VIANA DE MACËDO
PEDRO LUCAS GONZAGA PRATA
JORDIELSON EMANUEL CALDEIRA DA SILVA

TRABALHO DE CONCLUSÃO DE CURSO

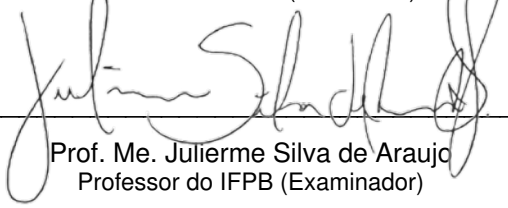
Imagem Pipeline: Sistema de pipeline para processamento de imagens

Trabalho de Conclusão de Curso (TCC) apresentado ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Campus Monteiro, formatado na Modalidade Projeto de Implementação, como pré-requisito para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas, sob orientação da Prof. M.Sc. Renata França de Pontes.

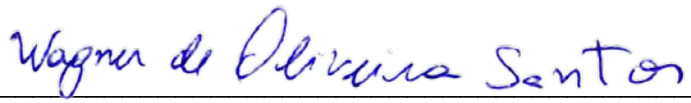
BANCA EXAMINADORA



Prof. M.Sc. Renata França de Pontes.
Professor do IFPB (Orientador)

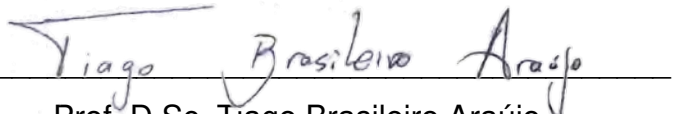


Prof. Me. Julierme Silva de Araujo
Professor do IFPB (Examinador)



Prof. Esp. Wagner de Oliveira Santos
Professor do IFPB (Examinador)

Visto e permitida a impressão.
Monteiro-PB, 08 de Agosto de 2023.



Prof. D.Sc. Tiago Brasileiro Araújo
Coordenador do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

AGRADECIMENTOS

Eu, Victor, agradeço aos meus pais por todo suporte e incentivo que me deram. A minha namorada que esteve sempre ao meu lado me incentivando. Aos amigos que fiz ao longo do curso que foram fundamentais para meu desenvolvimento profissional, assim como os professores e a instituição de ensino IFPB. Por fim, agradeço à professora Renata por toda sua dedicação na orientação do nosso Trabalho de Conclusão de Curso.

Eu, Jordielson, agradeço a Deus, Uno e Trino. Aos meus pais que são meus pilares por estarem comigo em todos os momentos, sendo uma fonte de inspiração, amparo, apoio e encorajamento, a vocês meu eterno e profundo agradecimento. A meus familiares por todo o incentivo prestado. Aos meus colegas de curso que são verdadeiros amigos, ajudando-me no meu crescimento profissional. Aos professores e demais membros do corpo docente, que contribuíram para a minha formação acadêmica, transmitindo conhecimentos e despertando minha paixão pelo desenvolvimento de sistemas. E um especial agradecimento à professora Renata Pontes, por sua orientação, paciência e expertise ao longo de todo esse processo, sua orientação foi fundamental para o desenvolvimento deste trabalho.

Eu, Pedro, gostaria de dedicar este momento de agradecimento para expressar minha imensa gratidão a todos que estiveram ao meu lado durante este processo de aprendizado. Em primeiro lugar, agradeço a Deus por me guiar e me proporcionar força e sabedoria e determinação para enfrentar os desafios ao longo desse caminho. A minha família, por seu amor, apoio constante e por acreditar em mim, sou eternamente grato. Aos meus colegas, e agora amigos, que sempre estiveram presentes, me incentivaram e compartilharam momentos de descontração, vocês foram uma fonte de motivação indispensável. Quero expressar minha profunda gratidão aos meus professores, em especial ao Professor João Paulo pelo seu comprometimento e atenção e entusiasmo por cada estudante, igualmente especial à excelentíssima Professora Renata Pontes, por sua dedicação, orientação e cuidado. Sua perícia e comprometimento foram fundamentais para o sucesso deste trabalho. A todos vocês, meu mais sincero agradecimento por fazerem parte desta conquista e por terem sido parte importante dessa jornada.

RESUMO

Este trabalho surge da necessidade de um desenvolvedor de sistemas, especialista em inteligência artificial, no gerenciamento de diversos serviços, primordialmente de processamento de imagens, distribuídos em várias APIs. Uma das dificuldades relatadas foi a construção de *pipeline* automatizado que é um fluxo de processamento de vários serviços computacionais autônomos, executados paralelo ou sequencialmente, com intuito de resolver um problema computacional, como a detecção de placas de trânsito. A solução proposta foi o desenvolvimento da plataforma Imagem Pipeline, que fornece uma *interface* para a integração com APIs de terceiros e a flexibilidade na criação de *pipelines* personalizados, permitindo a configuração de fluxos dos processos automatizados a serem aplicados às imagens. Assim, essa abordagem traz benefícios significativos ao usuário, incluindo o aumento da produtividade, eficiência, redução de custos e tempo de desenvolvimento visto que ele se mantém em sua especialidade. Esse *software* e sua implementação serão apresentados neste trabalho de conclusão de curso.

Palavras-chave: desenvolvimento de sistemas. *pipeline*. processamento de imagem. integração de API. gerenciamento de serviços

ABSTRACT

This work arises from the need of a systems developer, specialized in artificial intelligence, in the management of various services, primarily image processing, distributed across multiple APIs. One of the reported difficulties was the construction of an automated pipeline, which is a flow of processing multiple autonomous computational services, executed in parallel or sequentially, with the aim of solving a complex computational problem, such as traffic sign detection. The proposed solution was the development of the Imagem Pipeline platform, which provides an interface for integration with third-party APIs and flexibility in creating custom pipelines, allowing the configuration of automated process flows to be applied to images. Thus, this approach brings significant benefits to the user, including increased productivity, efficiency, cost reduction, and development time, as they can focus on their expertise. This software and its implementation will be presented in this Term paper.

Keywords: systems development. pipeline. image processing. API integration. service management

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ADS	Análise e Desenvolvimento de Sistemas
API	Application Programming *Interface*
BD	Banco de dados
BDD	Behavior Driven Development
CPS	Cenário Principal de Sucesso
CSS	Cascading Style Sheets
FPS	*Frames* por segundo
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IFPB	Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
IP	Internet Protocol
JPA	Java Persistence API
JSON	JavaScript Object Notation
JWT	JSON *Web* Token
MVC	*Model* View *Controller*
NPM	Node Package Manager
PA	Parâmetro de Aceitação
PDI	Processamento Digital de Imagens
PT-BR	Português do Brasil
RF	Requisito Funcional
RN	Regra de Negócio
RNF	Requisito Não Funcional
SGBD	Sistema de Gerenciamento de Banco de Dados
SL	Sessão de Levantamento
UC	Use Case
UML	*Unified Modeling Language*
URL	Uniform Resource Locator
UX	*User* Experience
XPath	XML Path Language

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTO E PROBLEMATIZAÇÃO (DOMÍNIO DO PROBLEMA)

Nos dias atuais é bastante comum APIs para processamento de imagens, seja para fotos, seja para vídeos. Com o uso de diversos aplicativos que trabalham com imagens, o processamento das mesmas é essencial. O professor José Filho, cliente do projeto em questão, está desenvolvendo APIs com serviços capaz de processar imagens no formato padrão (PNG, JPEG, JPG), videos (MP4, WMV) e câmeras de segurança (Protocolo RTSP) de forma que estas aplicações sejam capazes de interagir entre si, gerando um fluxo de processos conhecidos como *pipeline*, que tem como objetivo atingir um resultado específico como, por exemplo, a detecção de placas de trânsito seguido de uma mudança na *color*ação da imagem para tons de preto e branco.

Houve uma necessidade por parte do cliente de uma *interface* para manipular e gerenciar as APIs para que fosse possível, além de integrá-las pela *interface*, criar um *pipeline* para salvá-las e aplicá-las, futuramente, em outras mídias.

A falta de recursos, como, por exemplo, documentação das APIs, impossibilitou o desenvolvimento de uma *interface* padrão. Sendo assim, foi necessário buscar uma solução, de forma que a integração com as APIs funcionasse independentemente de sua estrutura.

1.2 JUSTIFICATIVA

A implementação da *interface* foi feita para garantir ao cliente uma integração de suas APIs em tempo de execução no formato *low-code* que é uma ferramenta de *model*agem visual com finalidades de criar lógicas nas aplicações utilizando pouco ou nenhum código, precisando apenas do mínimo de conhecimento em programação (BOBOYOROVA, CHAROS. 2023). Essa solução para o problema, citada na seção 1.1, garante que a aplicação possa ser colocada em produção antes da finalização de todas as APIs necessárias, já que esta será cadastrada na aplicação dinamicamente, sem programação. Além disso, foi desenvolvida a funcionalidade de criação de *pipelines* para possibilitar a interação entre vários processos de APIs variadas. Com isso, é possível aplicar técnicas de PDI a imagens, vídeos, e sequencias de imagens provida de câmeras de

segurança, em tempo de execução, e com baixo nível de acoplamento entre os serviços integrados.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Desenvolver uma plataforma *online* para manipular técnicas de PDI através de *pipelines* aplicadas a imagens, vídeos e sequencias de imagens provida de câmeras de segurança, em tempo real, utilizando APIs para o processamento de dados e execução dos serviços dispostos em ordem de aplicação pelo *pipeline* nas mídias citadas. Além disso, possibilitar a integração de API em tempo de execução.

1.3.2 Objetivos Específicos

Gerenciamento de Câmeras: Possibilitar o gerenciamento de câmeras de uso geral que possam ser cadastradas e conectadas ao sistema, permitindo, não só o monitoramento direto de vídeo individual de cada câmera, mas também a aplicação de múltiplas *pipelines* para o processamento de serviços oferecidos pela plataforma, utilizando de todos os dados oferecidos pelos aparelhos de câmera.

Gerenciamento de Pipelines: Possibilitar a criação e gerenciamento de *pipelines* que agrupam, ordenam e possibilitam a configuração de parâmetros dos serviços adicionados que serão aplicados em imagens, vídeos e/ou sequencias de imagens provida de câmeras de segurança.

Gerenciamento de Serviços: Possibilitar o cadastro e gerenciamento de serviços oferecidos pela API externa, contendo a definição dos parâmetros esperados pela API externa que os utilizará para o processamento de dados e solução do serviço.

Aplicação de Serviços: Possibilitar a aplicação de serviços em imagens, vídeos e sequencias de imagens provida de câmeras de segurança. Dado o

cadastro prévio de um ou mais serviços, estes podem ser adicionados e ordenados por um ou mais *pipelines*. *Pipelines* estes que, após a configuração de parâmetros dos serviços, podem ser aplicados a uma ou mais imagens, vídeos ou câmeras que, após esses procedimentos, apresentarão como resultado o conteúdo sobre influência dos efeitos definidos pelos parâmetros de cada serviço adicionado ao *pipeline* aplicado.

2 FUNDAMENTAÇÃO TEÓRICA

Com base na literatura, este capítulo aborda o contexto do *software*, os elementos teóricos e as tecnologias que suportam este trabalho.

2.1 CONTEXTUALIZAÇÃO DO PÚBLICO-ALVO E DAS APLICAÇÕES DO PRODUTO DE SOFTWARE

Um profissional da área de tecnologia da informação está desenvolvendo APIs que provê vários serviços como processamento de imagens, extração de dados e envio de mensagens. Considerando os diversos *endpoints* das APIs, a flexibilização dos parâmetros que permitem a escolha da análise aplicada e a visualização do resultado, percebe-se uma carência de um *software* de gerenciamento dessas APIs. No entanto, a principal dificuldade é a disposição dos serviços prestados pelas APIs em uma sequência de etapas, constituindo, assim, um *pipeline*. Assim, foi desenvolvido uma plataforma que permite a automatização desse trabalho manual de gerenciamento dos *pipelines*, reduzindo o tempo e custo na criação do *pipeline* e ainda resolvendo o problema de escalabilidade dos serviços. O público-alvo deste *software* é alguém com conhecimento em computação que possui serviços de processamento de imagens e está comprometido com a extração de dados de imagens por meio de fluxos e processos automatizados.

2.2 CONCEITOS E TRABALHOS RELACIONADOS AO DOMÍNIO DO PROBLEMA DE SOFTWARE

Processamento de imagem é um processo aplicado em imagem com o objetivo de melhorá-la para a visualização humana ou de prepará-la para aplicações computacionais automatizadas (Gonzalez; Woods, 2000).

De acordo com Ramamoorthy(Pipeline Architecture, 1977, p.2):

> *Pipeline* refere-se a uma segmentação de um processo computacional em vários subprocessos que são executados por unidades autônomas dedicadas a processos sucessivos os quais podem ser executados em um modo sobreposto análogo a uma linha de mon*tag*em industrial. Sendo uma forma de incorporar o paralelismo ou simultaneidade em um sistema computacional.

As principais plataformas conhecidas e estudadas para o processamento de imagem com a arquitetura de *pipeline* são o Viso.Ai e o Clarifai.

2.2.1 Viso.Ai

Viso é uma plataforma de inteligência artificial que permite o processamento de vídeo em tempo real para análise e tomada decisão em áreas como segurança, trânsito e varejo. Segundo a Intel, a Viso é uma solução que permite às empresas de todos os tamanhos aproveitar a tecnologia de visão computacional para automatizar processos de negócios, melhorar a segurança e gerar *insights* valiosos" (Intel, 2021).

De acordo com a Viso.Ai (Viso.ai, 2021, p. 2):

> "A Viso.ai oferece uma ampla variedade de recursos de processamento de imagem, incluindo reconhecimento de objetos, rastreamento de objetos, detecção de faces, análise de expressões faciais, reconhecimento de texto, análise de cores e muito mais"

A plataforma oferece uma Demo para apresentação do produto, mas permanece como um produto por assinatura para acesso completo das funcionalidades.

2.2.2 Clarifai

Clarifai é uma plataforma de inteligência artificial que fornece serviços para empresas que possibilitam a elas identificar os elementos contidos nas imagens

e vídeos. Através de sua plataforma é possível personalizar como e quais elementos serão reconhecidos (Clarifai, 2023).

Apesar dessas plataformas processarem algum formato de imagem, não fornecem recursos de suporte para desenvolvedores aplicarem seus serviços de processo de imagem externos na plataforma. Essas plataformas foram implementadas com a intenção de alcançar um público alvo diferente do Imagem Pipeline, além de serem pagas.

2.3 TECNOLOGIAS UTILIZADAS

React:

React é uma biblioteca de JavaScript para desenvolvimento de *interface* de usuário que isola códigos em componentes o que o torna uma biblioteca declarativa, eficiente e flexível (React, 2023).

O React foi escolhido para fazer a *interface* de usuário por seu nível de aprendizado rápido e eficiente, e por suas bibliotecas que oferecem funções que atendem a necessidade do cliente.

React-Bootstrap:

O React-Bootstrap é uma biblioteca para a utilização do Bootstrap no React. O Bootstrap, por sua vez, é um *framework* que fornece componentes Javascript e HTML com CSS pré-definidos, com design responsivo e profissional (REACT BOOTSTRAP, 2023).

A sua aplicação foi necessária para agilizar o tempo de desenvolvimento da *interface* de usuário, pela responsividade e facilidade de personalização que o Bootstrap oferece.

React Flow:

React Flow é uma biblioteca Javascript que fornece recursos para a criação de *interface*s de usuário de fluxo de trabalho visual como diagramas e fluxogramas (React flow, 2023). A biblioteca foi utilizada para possibilitar a criação e manipulação de *pipelines* de imagens.

NPM:

NPM (Node Package Manager) é o Gerenciador de pacotes para a plataforma Node.Js que permite a instalação e a gerência de pacotes de *softwares* reutilizáveis de Javascript (NPMJS, 2023).

O NPM foi utilizado para a instalação de todas as bibliotecas Javascript utilizadas no sistema.

Axios:

Axios é uma biblioteca que permite a integração do sistema com APIs externas através de chamadas HTTP assíncronas e utiliza promessas para lidar com essas chamadas (AXIOS, 2023).

Essa tecnologia fornece uma API mais simples e intuitiva de realizar chamadas HTTP, o que facilitou no aprendizado e implementação das APIs no sistema Imagem Pipeline.

JUnit:

"O JUnit é um *framework* de código aberto inicialmente desenvolvido por Kent Beck e Erick Gamma. O JUnit permite a criação de testes automatizados utilizando Java." (DEV MEDIA, 2021). É uma ferramenta popular entre desenvolvedores de *software*, pois ajuda a garantir que o código esteja funcionando corretamente e sem erros.

Através dos testes, a equipe de desenvolvimento pode verificar se o código está funcionando como deveria e se atende às expectativas. Além disso, o JUnit ajuda a identificar erros e falhas no código antes que o produto final seja entregue. Isso economiza tempo e recursos, já que é mais fácil e barato corrigir problemas no início do desenvolvimento do que após o lançamento.

O JUnit foi escolhido, pois além do citado, esta tecnologia permite não só a criação de testes automatizados como também pode ser usada em conjunto com outras tecnologias que possibilitam a criação de testes funcionais e de integração.

Selenium:

De acordo com o site oficial do Selenium, "Selenium é um projeto que abrange uma variedade de ferramentas e bibliotecas que permitem e suportam a automação de navegadores da *web*." (Selenium, 2023).

O Selenium permite a execução de testes automatizados em diferentes navegadores e plataformas. Ele simula as ações do usuário em um navegador, permitindo o teste de um aplicativo *web* de forma mais eficiente e com menos esforço.

O Selenium é uma ferramenta de código aberto que suporta diversas linguagens de programação, entre elas a linguagem JAVA. Ele permite a escrita de *scripts* de teste que podem ser executados automaticamente em diferentes navegadores. Além disso, o Selenium pode ser integrado com outras ferramentas de automação de testes, como o JUnit e Cucumber que também foram usados no desenvolvimento de testes do Imagem Pipeline.

Cucumber:

"Cucumber é uma ferramenta que suporta BDD, que é o acrônimo para Behavior Driven Development, ou seja, Desenvolvimento Orientado por Comportamento." (COODESH, 2021).

Cucumber facilita a comunicação entre equipes de desenvolvimento e negócios. Ela permite escrever testes em uma linguagem simples e legível, chamada Gherkin. Com o Cucumber, é possível criar cenários de testes que representam as funcionalidades de um *software* de forma clara e objetiva. Esses cenários podem ser facilmente entendidos por todas as partes envolvidas no processo de desenvolvimento, incluindo desenvolvedores, testadores e *stakeholders*.

O Cucumber foi escolhido especialmente por proporcionar uma leitura mais simples dos casos de testes, facilitando a organização dos mesmos e criando um ambiente altamente ajustável e possibilitando uma variedade imensa de casos de testes, além de gerar relatórios detalhados sobre o comportamento do sistema, incluindo informações sobre quais testes foram executados com sucesso e quais falharam. Isso ajuda os desenvolvedores a identificar rapidamente quaisquer problemas ou erros em seu código e a corrigi-los antes que eles se tornem problemas mais sérios.

Gherkin:

Gherkin é uma linguagem de especificação utilizada em desenvolvimento de *software* visando criar uma documentação automatizada e estruturada de cenários de teste. Ela cria cenários de teste escritos em formato de histórias, essas histórias descrevem o comportamento esperado do sistema em diferentes cenários. Segundo o site oficial do Cucumber, "Gherkin é um conjunto de regras gramaticais que torna o texto simples estruturado o suficiente para o Cucumber entender." (CUCUMBER, 2023).

Gherkin é usado em conjunto com o Cucumber, que a interpreta e executa os testes, gerando relatórios detalhados sobre o resultado dos testes. Esta linguagem foi o principal atrativo para que o Cucumber fosse utilizado, pois sua implementação é simples, fácil de ler e entender, pois usa palavras-chave como Given, When e Then , com suporte ao idioma Português Brasileiro (Dado, Quando e Então) além de outras variações, para descrever o comportamento esperado do sistema em diferentes cenários. Possibilitando uma criação intuitiva de etapas sequenciais para a criação e validação de cada cenário de teste funcional do sistema.

Java:

Sendo uma poderosa plataforma de *software*, a linguagem de programação Java é utilizada para desenvolver aplicações para uma ampla variedade de ambientes, podendo ser caracterizada por ser: orientada a objeto, simples, distribuída, multith*read*ed, dinâmica, robusta, segura, portátil, alta performance e arquitetura neutra (Oracle, 2023). Java foi escolhida pois não apenas é uma plataforma confiável, garantindo a segurança necessária a aplicação, mas também possui uma série de APIs e de *frameworks* com diversos módulos que solucionam os problemas encontrados nesse *software*. Por fim, Java possui uma enorme comunidade muito ativa a qual serve de auxílio durante o desenvolvimento.

Spring *framework*:

O Spring *framework* é uma plataforma Java de código aberto que dispõe de uma infraestrutura para o desenvolvimento simples e rápido de aplicativos Java

robustos e sustentáveis (JOHNSON, 2005). O Spring foi escolhido devido a módulos e projetos de seu ecossistema como o Spring Boot, para reduzir o tempo de desenvolvimento de aplicativos Java complexos da *Web*, o Spring Security, para a autenticação e controle de acesso à API, e o Spring Data JPA, para a realização do mapeamento objeto relacional.

REST:

REST é um estilo arquitetural que define um conjunto de restrições à interação entre sistemas computacionais *Web*, fornecendo, assim, padrões que facilitam essa comunicação (Fielding, 2000). A utilização do estilo arquitetural REST foi primeiramente devido a restrição de implementação cliente e servidor, separando as preocupações com a *interface* do usuário das regras de negócio e gerenciamento dos dados, permitindo maior escalabilidade dos serviços e melhor experiência do usuário. Isso faz com que cada componente evolua independentemente com interação flexível. O segundo motivo foi não ter necessidade do servidor de manter o estado da sessão de cada cliente.

Git:

Git é uma ferramenta de controle de versão que armazena os dados como uma série de *snapshots* de um pequeno sistema de arquivos. Além disso, todas as operações no Git geram um *checksum* garantindo que qualquer modificação em um arquivo ou diretório seja identificada pelo Git (CHACON; STRAUB, 2014). A decisão pelo Git foi justamente pelo mecanismo de integridade e pela forma de armazenamento do histórico, sendo muito difícil perder dados ou não conseguir restaurar uma versão anterior do sistema.

PostgreSQL:

PostgreSQL é um sistema de gerenciamento de banco de dados relacional de objeto que dispõe de vários recursos modernos como consultas complexas, visões atualizáveis, integridade transacional, entre outros (PostgreSQL, 2023). Esse SGBD (Sistema de Gerenciamento de Banco de Dados) foi escolhido por suas características como, por exemplo, facilidade de acesso, garantia da integridade, proteção dos dados, compatibilidade com várias plataformas, suporte para as principais linguagens de programação, etc.

Astah:

Astah é uma ferramenta de *model*agem de sistemas que oferece suporte para diagramas UML (*Unified Modeling Language*) e geração automática de código-fonte a partir dos diagramas (*Astah*, 2021). O *Astah* foi utilizado na *model*agem de diagramas UML ajudando a equipe, na fase de análise, a especificar e organizar os requisitos e o fluxo do sistema por meio dos diagramas de caso de uso, de atividade e de estado. Durante a especificação do projeto, auxiliou na compreensão das funcionalidades, das estruturas e dos comportamentos do sistema com os diagramas de classes, de sequência, de atividades, de componentes e de implantação.

Node.js:

Node.js é um ambiente de desenvolvimento de código aberto que executa código JavaScript no lado do servidor, orientado a evento e possui uma arquitetura de E/S não bloqueante, isto é, permite a execução do restante do código sem a necessidade de aguardar o término de uma operação não JavaScript. Isso o torna muito eficiente para aplicativos em tempo real (POWERS, 2012). A escolha do Node.js foi pelo seu ótimo desempenho com aplicações em tempo real, o que é necessário para obtenção das imagens das câmeras.

TypeScript:

TypeScript é uma linguagem de programação que se baseia e estende JavaScript com a adição de diversas capacidades como os tipos estáticos, garantindo mais segurança e robustez no projeto (TypeScript, 2023). A decisão por TypeScript foi para ajudar na prevenção de muitos *bugs* durante o desenvolvimento, facilitar a manutenção do código e, ainda, economizar tempo, visto não ter necessidade de fazer a validação dos tipos de variáveis.

Python:

Python é uma linguagem de programação orientada a objetos com ti*page*m dinâmica. Ele oferece suporte para módulos e pacotes, além de possuir uma ampla biblioteca padrão (LUTZ, 2013). Python é essencial para a análise de dados e aprendizado de máquina, pois possui várias bibliotecas analíticas desenvolvidas

por sua vasta comunidade, as quais oferecem grande suporte. Por esse motivo Python, em conjunto com Flask e OpenCV, foi selecionada na construção de serviços, simples e exemplares de processamento de imagem, com o objetivo de visualizar e testar o funcionamento do processamento de mídia na *interface*.

Flask:

Flask é um *framework* de aplicação *Web* escrito em Python sem a imposição de nenhuma dependência ou estrutura de projeto (GRINBERG, 2014). A decisão de Flask foi por ser um leve *microframework*, então não só é rápido e fácil de aprender como fornece a opção de quais bibliotecas e ferramentas utilizar no projeto. O Flask foi utilizado para a criação de uma API de processamento de imagens que se assemelha a uma API externa que será utilizada.

Docker:

Docker é uma plataforma de código aberto para o desenvolvimento, execução e automatização da implantação de aplicativos, destinado a separar aplicativos do ambiente de desenvolvimento utilizando containers portáteis e autossuficientes (Docker, 2023). A escolha do Docker foi devido, principalmente, a essas características de isolamento e portabilidade para que toda a configuração e dependência do projeto sejam armazenados em um container que pode ser compartilhado com a garantia que a execução ocorrerá da mesma maneira em todos os ambientes de desenvolvimento e de produção.

OpenCV:

OpenCV é uma biblioteca com estrutura modular de código aberto que possui uma ampla variedade de algoritmos de visão computacional (OpenCV, 2023). A API do OpenCV disponibiliza várias ferramentas e algoritmos para o processamento de imagem e vídeo, as quais foram fundamentais para o rápido e simples desenvolvimento de serviços de processamento de imagem como segmentação de objetos para a detecção de placas.

3 RESULTADOS OBTIDOS

Esta seção apresenta os processos de desenvolvimento de *software*, incluindo todas as atividades e análises realizadas para o levantamento de informações e as etapas de desenvolvimento e teste do sistema.

3.1 ENGENHARIA DE REQUISITOS

Esta seção detalha todas as atividades realizadas que conduziram ao entendimento dos requisitos do sistema.

A engenharia de requisitos são tarefas e técnicas que resultam no entendimento das necessidades do cliente e dos requisitos do sistema. Os requisitos indicam as características operacionais do *software* e especificam as restrições que o sistema deve atender (PRESSMAN, 2016).

ID	OBJETIVOS	TÉCNICA SELECIONADA	FORMATO DE REGISTRO DA SESSÃO
SL 1	Identificar as necessidades e expectativas iniciais do cliente	Entrevista	Chamada por Vídeo
SL 2	Compreender como será aplicada o processamento	Questionário	Aplicativo de mensagem instantânea
SL 3	Entender o acesso da API que serão usadas para o processamento de imagens dentro de um *pipeline*	Oficina	Chamada por Vídeo
SL 4	Entender até que ponto a configuração da câmera poderá ser feita	Questionário	Aplicativo de mensagem instantânea
SL 5	Fazer uma experimentação da aplicação para ter uma base de conhecimento do que será feito no Imagem Pipeline	Experimentação de *software*	https://viso.ai/*features*/

Quadro 1 - Levantamento de requisitos

SL 1 - Nesta sessão de levantamento de requisitos a equipe de desenvolvimento planejou, executou e registrou uma reunião de entrevista por videoconferência com o cliente para conhecer o domínio do problema, identificar suas necessidades e vislumbrar a solução através de suas expectativas.

SL 2 - A segunda sessão de levantamento de requisitos foi feita por meio de um questionário, via aplicativo de mensagem instantânea, com o intuito de obter informações satisfatórias para compreensão sobre como funcionaria a aplicação de processos, uma vez que foi levantado na entrevista inicial a informação de que o processamento dos serviços seria realizado mediante APIs externas, de forma que a solução seria responsável apenas pela comunicação entre elas.

SL 3 - Para aprofundar as informações coletadas via questionário na segunda sessão de levantamento de requisitos, a equipe solicitou ao cliente uma reunião por videoconferência em forma de oficina, pois o cliente era o único detentor de informações sobre as APIs externas que seriam utilizadas para o processamento de serviços. Estas informações foram importantes para o entendimento sobre o acesso, comunicação e funcionamento destas APIs.

SL 4 - Na quarta sessão também foi elaborado um questionário via aplicativo de mensagem instantânea para entender quais configurações precisam ser levadas para as APIs que irão processá-la e quais configurações precisam ser feitas para manipular uma câmera de segurança na plataforma.

SL 5 - A quinta e última sessão de levantamento de requisitos foi feita mediante uma experimentação de *software* semelhante, o VISO.Ai. Esta experimentação possibilitou à equipe analisar aspectos de usabilidade, identidade visual e funcionalidade de uma câmera de segurança em conjunto com um *pipeline* de processos aplicada a esta câmera.

3.1.1 Requisitos Funcionais

Os requisitos funcionais são declarações dos serviços que o sistema deve possuir para cumprir as funcionalidades básicas do *software*. Eles explicam o que o sistema deve ou não deve fazer em respostas a diferentes entradas e situações (SOMMERVILLE, 2011).

Com base nas Informações coletadas nas sessões de levantamento de requisitos, ficaram definidos os seguintes requisitos funcionais:

MÓDULO DE FUNCIONALIDADES	**#**	**REQUISITO FUNCIONAL**	**RASTREABILIDADE (com outros RFs)**	**MUTABILIDADE**	**PRIORIDADE**
Usuário (US)	RF.US 1	Cadastramento de usuário	-	Baixa	Alta
Usuário (US)	RF.US 2	Autenticação de usuário	RF.US 1	Baixa	Alta
Câmera (CR)	RF.CR 1	Cadastramento de câmera	RF.US 2	Baixa	Alta
Pipeline (PL)	RF.PL 1	Cadastramento de *pipeline*	RF.US 2, RF.GP 1	Média	Alta
Pipeline (PL)	RF.PL 2	Configuração de serviços e de *pipeline*	RF.US 2,RF.PL 1	Alta	Alta
Pipeline (PL)	RF.PL 3	Ativação e desativação de *pipeline*	RF.US 2, RF.PL 1	Alta	Alta
Pipeline (PL)	RF.PL 4	Visualização do *pipeline*	RF.US 2,RF.PL 1, RF.PL 2, RF.PL 3	Média	Alta
Pipeline (PL)	RF.PL 5	Processamento de imagem, vídeo ou câmera pelo pipeline	RF.US 2,RF.PL 1, RF.PL 2, RF.PL 3	Baixa	Alta
Pipeline (PL)	RF.PL 6	Gerenciamento do histórico de versões do *pipeline*	RF.US 2,RF.PL 1	Baixa	Média
*Model*o Serviço (MS)	RF.MS 1	Gerenciamento de serviços para *pipeline*	RF.US 2	Alta	Alta

Quadro 2 - Requisitos funcionais

RF. US 1	**REQUISITO FUNCIONAL**
	Cadastramento de usuário
DESCRIÇÃO	
Sem um cadastro, não será possível acessar nenhum serviço do sistema uma vez que cada usuário tem privacidade sobre suas informações.. Com o cadastro concluído, o usuário poderá se autenticar e terá acesso a tudo que o sistema fornece.	

REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	O e-mail precisa ser único e validado no padrão de e-mail convencional (exemplo@exemplo.com)
RN02	O e-mail deverá ser utilizado em apenas um cadastro
RN03	Deve ser obrigatório ao usuário uma senha com no mínimo seis dígitos
RN04	O próprio usuário faz o seu cadastro
RN05	Será necessário inserir um e-mail e uma senha para se cadastrar

Quadro 3 - Requisito funcional: Cadastramento de usuário

RF. US 2	**REQUISITO FUNCIONAL**
	Autenticação de usuário
DESCRIÇÃO	
A autenticação será necessária para usar todas as outras funcionalidades do sistema, somente após a autenticação, o usuário poderá utilizar das funcionalidades atribuídas a ele. Na seção de *login* será informado o e-mail e a senha corretamente para se autenticar e poder utilizar a aplicação.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Será preciso informar o e-mail e a senha para se autenticar
RN02	É necessário validar o e-mail e senha informado

Quadro 4 - Requisito funcional: Autenticação de usuário

RF. CR 1	**REQUISITO FUNCIONAL**
	Cadastramento de câmara
DESCRIÇÃO	
O cadastramento de câmera será responsável por cadastrar e salvar as câmeras de um usuário no sistema. Será útil para fazer o usuário fazer o gerenciamento das câmeras e escolher a câmera que será usada pelo *pipeline*. Essa funcionalidade é importante para o sistema obter as imagens das câmeras que serão processadas pelo *pipeline*.	

REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Deverá informar um IP fornecido pela câmera de segurança, assim como suas informações básicas (Resolução e *Color*ação)
RN02	Verificar se a API está recebendo imagem do IP antes de concluir o cadastro

Quadro 5 - Requisito funcional: Cadastramento de câmera

RF. PL 1	**REQUISITO FUNCIONAL**
	Cadastramento de *pipeline*
DESCRIÇÃO	
Fundamental no sistema já que o *pipeline* é o fluxo de todo o processo que o cliente irá ordenar e direcionar, podendo usar diferentes serviços para que tenha o resultado esperado no final de tudo isso. Será necessário usá-la para fazer a configuração dos serviços e pré-visualização de mídia após a execução do *pipeline*.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Um nome o *pipeline* deve ser atribuído, assim como uma câmera já cadastrada pelo usuário.
RN02	Não deve ser possível remover *pipelines* que estão sendo processadas.
RN03	Não poderá haver dois *pipelines* criadas com o mesmo nome

Quadro 6 - Requisito funcional: Cadastramento de *pipeline*

RF. PL 2	**REQUISITO FUNCIONAL**
	Configuração de serviços e de *pipeline*
DESCRIÇÃO	
Será utilizada para adicionar os serviços como de processamento de imagem e a configuração de seus parâmetros (dimensões da imagem, por exemplo) no *pipeline*. Será permitido adicionar um *pipeline* como se fosse um serviço em outro *pipeline*, desde que os processos sejam finitos. Necessário para detalhar as informações para aquele serviço selecionado, tendo em vista o *pipeline* que ele faz parte.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Os serviços adicionados são ordenados com a técnica arrasta e solta

RN02	Alterar ou adicionar informações no processamento de imagem para adaptar a necessidade
RN03	Serviços que fornecem funcionalidade de marcação terão configuração de cor sobre estas marcações

Quadro 7 - Requisito funcional: Configuração de serviços e de *pipeline*

RF. PL 3	**REQUISITO FUNCIONAL**
	Ativamento e desativamento de *pipeline*
DESCRIÇÃO	
É feita a propagação do *pipeline* nos serviços que ela possui, ou seja, as requisições aos serviços. Sem o ativamento do *pipeline* não seria possível executar os serviços de processamento de imagem e assim a principal funcionalidade do sistema que é o gerenciamento da sequência de etapas do *pipeline* não estaria disponível. O desativamento é necessário para pausar a ativação do *pipeline* caso um serviço esteja demorando muito, consumindo muito processamento ou por conveniência do usuário.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Ao ativar estará processando o *pipeline* indicando que está sendo usado.
RN02	Ao desativar será parado o processamento do mesmo fazendo que não seja utilizável naquele momento.
RN03	Será permitido a reativação de um serviço que foi desativado, devendo o processamento continuar da onde parou.
RN04	O usuário poderá ativar mais de um *pipeline* ao mesmo tempo.

Quadro 8 - Requisito funcional: Ativamento e desativamento de *pipeline*

RF. PL 4	**REQUISITO FUNCIONAL**
	Visualização do *pipeline*
DESCRIÇÃO	
Permite ao usuário ver o resultado após a configuração de um *pipeline* possibilitando ao usuário ver o resultado do *pipeline* criado sendo realizado em tempo real. Esse é um exemplo de como seria o resultado final do processamento do *pipeline*.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Só será possível ter uma visualização assim que o *pipeline* estiver com um fluxo aceitável (com *pipeline* e serviços bem selecionados e ligados).

RN02	O usuário poderá visualizar um exemplo do resultado do *pipeline* que estiver configurando.

Quadro 9 - Requisito funcional: Visualização de *pipeline*

RF. PL 5	**REQUISITO FUNCIONAL**
	Processamento de imagem, vídeo ou câmera pelo *pipeline*
DESCRIÇÃO	
Permite ao usuário aplicar o *pipeline*, após a conclusão da configuração, em sua imagem, vídeo ou câmera. Possibilitando ao usuário ver o resultado e realizar o download no caso de processamento de imagem ou vídeo. Esse é o resultado final da funcionalidade principal que o sistema fornece.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Só será possível realizar o processamento do *pipeline* ativo.
RN02	O usuário poderá fazer download da imagem ou vídeo gerado.

Quadro 10 - Requisito funcional: Processamento de imagem, vídeo ou câmera pelo *pipeline*

RF. PL 6	**REQUISITO FUNCIONAL**
	Gerenciamento do histórico de versões do *pipeline*
DESCRIÇÃO	
Lista com todas as versões de um *pipeline* possibilitando ao usuário fazer a restauração de uma versão específica para a sua reconfiguração e realizar o processamento do *pipeline* nesta versão.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Ao salvar um *pipeline* devidamente configurado será criado uma versão específica e adicionado ao histórico
RN02	O histórico de versões mostrará inicial as últimas modificações do *pipeline*
RN03	O usuário poderá nomear cada versão do histórico

Quadro 11 - Requisito funcional: Gerenciamento do histórico de versões do *pipeline*

RF. MS 1	REQUISITO FUNCIONAL
	Gerenciamento de serviços para *pipeline*
DESCRIÇÃO	
Um *model*o de serviço que especifica seus parâmetros pode ser criado. Deve ser utilizado pelo usuário para a criação funcional deste determinado serviço. O objetivo de cada serviço é fazer com que todos os parâmetros requisitados sejam enviados diretamente para a API externa informada, solicitando o processo do serviço.	
REGRAS DE NEGÓCIO ASSOCIADAS (RN)	
RN01	Deverá possuir uma marcação que definirá se o parâmetro será ou não obrigatório
RN02	Não pode ter nome repetido para o *model*o.
RN03	O próprio usuário faz o gerenciamento dos *model*os de serviços.
RN04	Os tipos de parâmetros que poderão ser criados são: parâmetros de texto, número, cor, lista de opções e *boolean*o

Quadro 12 - Requisito funcional: gerenciamento de serviços para *pipeline*

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais expressam restrições de características aos serviços ou funções do sistema para satisfazer uma necessidade do usuário (SOMMERVILLE, 2011).

No quadro a seguir serão apresentados requisitos não funcionais que foram levantados de acordo com a necessidade do cliente tendo em vista o conhecimento técnico da equipe para solucionar o problema proposto.

ATRIBUTO DE QUALIDADE	#	REQUISITO NÃO-FUNCIONAL	RASTREABILIDADE (com RFs)	MUTABILIDADE	PRIORIDADE
Manutenibilidade	RNF 1	Desenvolvimento de back end e front end separados	Todos	Baixa	Alta
Operacionalidade	RNF 2	Arrasta e solta para configurar o pipeline	Todos	Média	Alta

ATRIBUTO DE QUALIDADE	#	REQUISITO NÃO-FUNCIONAL	RASTREABILIDADE (com RFs)	MUTABILIDADE	PRIORIDADE
Apreensibilidade	RNF 3	Material didático para a configuração de *pipelines*	RF.PL 1, RF.PL 2, RF.PL 3, RF.PL 4	Média	Baixa
Interoperabilidade	RNF 4	Comunicação com a API de serviços externos	RF.MS 1	Baixa	Alta
Portabilidade	RNF 5	Acesso por meio de navegador *web*	Todos	Baixa	Alta
Portabilidade	RNF 6	Exibir formato de vídeo compatível com o serviço	RF.PL 4	Média	Alta
Confiabilidade	RNF 7	Informações serão salvas em um SGBD com possibilidade de *backup*	Todos	Média	Baixa
Segurança	RNF 8	Serviço de autenticação para o usuário	Todos	Baixa	Alta

Quadro 13 - Requisitos não funcionais

RNF 01	**REQUISITO NÃO FUNCIONAL**
	Desenvolvimento de *back-end* e *front-end* separados
DESCRIÇÃO	
Desenvolvimento do *back-end* e do *front-end* separadamente para uma melhor organização e para uma melhor divisão de trabalho por parte dos desenvolvedores do sistema.	
PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	O *front-end* deverá ser desenvolvido em React
PA02	O *back-end* deverá ser desenvolvido em Java utilizando o *framework* Spring
PA03	A comunicação entre eles será por padrão REST

PA04	Os dados serão transferidos no formato JSON

Quadro 14 - Requisito não funcional: Desenvolvimento de back-end e front-end separados

RNF 02	**REQUISITO NÃO FUNCIONAL**
	Arrasta e solta para configurar o *pipeline*
DESCRIÇÃO	
A configuração do *pipeline* e seu ordenamento será realizado com a metodologia arrasta e solta, onde o usuário clica e segura uma figura representativa de um Serviço e solta onde desejar num espaço de edição de *pipeline* reservado para essa funcionalidade e conecta um Serviço com outra de forma ágil, prática e intuitiva para o usuário	
PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	Será utilizada a técnica de *Drag-and-drop* para fazer este serviço.
PA02	A biblioteca utilizada será React Flow

Quadro 15 - Requisito não funcional: Arrasta e solta para configurar o *pipeline*

RNF 03	**REQUISITO NÃO FUNCIONAL**
	Material didático para a configuração de *pipelines*.
DESCRIÇÃO	
Devido à complexidade de configuração de um *pipeline*, por haver inúmeras funcionalidades a serem configuradas, torna-se necessário a criação de materiais didáticos, demonstrando como realizar o procedimento de diferentes tipos de configuração.	

PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	Deve ser apresentado GIFs nos guias para um melhor entendimento.
PA02	Deve ser feito um roteiro com os pontos principais que serão demonstrados no guia
PA03	O guia deve ser disponibilizado na própria plataforma.

Quadro 16 - Requisito não funcional: Material didático para configuração de *pipelines*

RNF 04	**REQUISITO NÃO FUNCIONAL**
	Comunicação com a API de serviços externos
DESCRIÇÃO	
Estabelecer uma comunicação com a API de processamento de imagem é fundamental já que o sistema de gerenciamento de *Pipelines* depende exclusivamente dela. É a API que vai fazer todo o trabalho de entregar a imagem pronta para o sistema, abstraindo toda a complexidade.	
PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	API disponibilizada remotamente
PA02	Comunicação com a API no Padrão RESTful
PA03	Os dados serão transferidos no formato JSON

Quadro 17 - Requisito não funcional: Comunicação com a API de serviços externos

RNF 05	**REQUISITO NÃO FUNCIONAL**
	Acesso por meio de navegador *web*

DESCRIÇÃO	
Para acessar o sistema será preciso informar o endereço do mesmo no navegador de internet que será definido futuramente verificando a disponibilidade de domínio.	
PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	Conexão com a internet
PA02	*Interface web* responsiva
PA03	Acessado pelo endereço que foi definida no *front-end*

Quadro 18 - Requisito não funcional: Acesso por meio de navegador web

RNF 06	**REQUISITO NÃO FUNCIONAL**
	Informações serão salvas em uma base de dados com possibilidade de *backup*
DESCRIÇÃO	
Todas as informações que forem fornecidas quando utilizado o sistema serão salvas em um sistema de gerenciamento de banco de dados e terá a possibilidade de fazer um *backup* de dados que poderiam ser perdidos em uma outra ocasião.	
PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	Será utilizado PostgreSQL como sistema de gerenciamento de banco de dados
PA02	Para o *backup* será criado um clone do banco de dados principal
PA03	*Backup* será atualizado a cada alteração com a versão anterior

Quadro 19 - Requisito não funcional: Informações serão salvas em uma base de dados com possibilidade de backup

RNF 07	**REQUISITO NÃO FUNCIONAL**
	Serviço de autenticação para o usuário
DESCRIÇÃO	
Vai existir um serviço que vai se encarregar de garantir a autenticação do sistema, trazendo uma restrição de acesso ao mesmo. Utilizado no momento de tentar acessar as funcionalidades do produto.	
PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	O limite de tempo autenticado no sistema é de 4h
PA02	A autenticação do sistema usará o padrão JWT

Quadro 20 - Requisito não funcional: Serviço de autenticação para o usuário

RNF 08	**REQUISITO NÃO FUNCIONAL**
	Exibir vídeo com formato compatível com o serviço
DESCRIÇÃO	
Visualização do vídeo após o ativamento do *pipeline* no mesmo formato de recebimento do serviço, possibilitando a utilização desse vídeo em outro serviço dentro do *pipeline*.	
PARÂMETROS DE ACEITAÇÃO (PA)	
PA01	Formatos aceitáveis: - (MP4 e WMV)

Quadro 21 - Requisito não funcional: Exibir vídeo com formato compatível com o serviço

3.2 PROJETO COMPORTAMENTAL E ESTRUTURAL

Nesta seção será abordado a estrutura e o comportamento dinâmico do sistema. Com o objetivo de ter uma visão geral do projeto e um melhor entendimento das necessidades do *software*, foram produzidos artefatos na atividade de *model*agem do *software* durante a disciplina de Projeto 1.

3.2.1 Diagramas Comportamentais

*Model*os comportamentais apresentam a reação do sistema, quando está em execução, a um estímulo que pode ser a chegada de novos dados ou a ocorrência de um evento externo ou interno, sempre acarretando algum processamento do sistema (SOMMERVILLE, 2011).

3.2.1.1 Diagrama de caso de uso

Para Martin Fowler (2004), os Casos de Uso são um método de extração dos requisitos funcionais de um sistema, especificando as típicas interações entre o *software* e seus usuários, assim os diagramas de Caso de Uso fornecem uma narrativa da utilização do sistema.

No estudo de caso, a *model*agem do sistema foi realizada em um único diagrama, pois há apenas um grupo de funcionalidades utilizadas por somente um autor, denominado Usuário.

A Figura 1 apresenta todos os casos de uso do sistema e como estão relacionados com o autor.

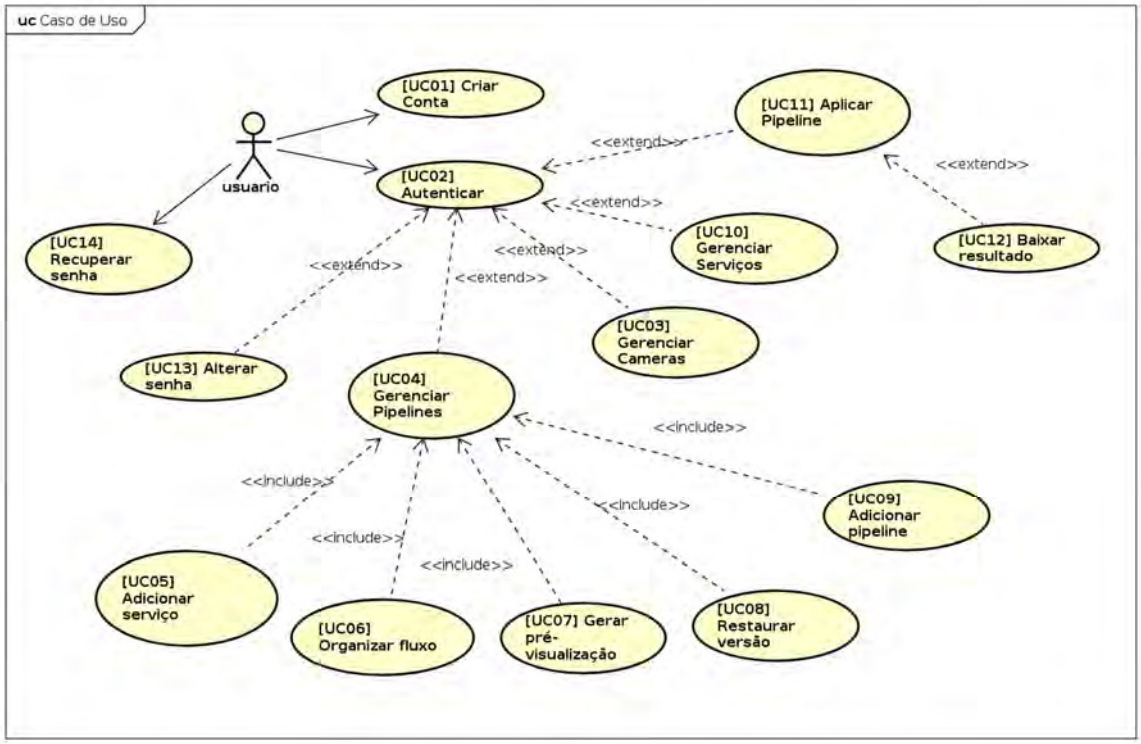


Figura 1 - Diagrama de caso de uso do sistema

Como não há uma maneira padronizada para descrever o conteúdo dos casos de uso, será utilizada uma que melhor descreve os cenários e as condições, visto a complexidade do sistema no gerenciamento de *pipeline* e serviços e por haver um único autor. Portanto, como bem ensina Fowler (2004, p. 105), o detalhamento do caso de uso conterá:

1. Os **requisitos funcionais** atendidos pelo caso de uso;
2. As **pré-condições** é tudo o que o sistema deve garantir antes de iniciar o caso de uso;
3. As **garantias de sucesso** são as condições que se mantêm após os cenários bem sucedidos;
4. O **cenário principal de sucesso** (CPS) é a principal alternativa de sucesso do caso de uso;
5. As **extensões** são cenários alternativos do caso de uso nos quais o autor pode atingir o fim desejado ou não.

- **[UC01] Criar Conta**

  Requisitos Funcionais

- RF.US 1

Pré-Condições

- Nenhum

Garantias de Sucesso

- O sistema deve armazenar as informações do usuário possibilitando a realização de *login*.

Cenário Principal de Sucesso

1. O usuário acessa a página de cadastramento de usuário no sistema;
2. O usuário preenche os campos com as informações necessárias;
3. O sistema realiza o cadastramento;
4. O sistema redireciona o usuário para a tela de autenticação.

Extensões

3a. O sistema falha ao criar uma conta para o usuário

1. O sistema exibe o motivo da falha, liberando o usuário para inserir novamente as informações em conformidade com a solicitação do sistema, retornando para o CPS, no passo 2.

- **[UC02] Autenticar**

Requisitos Funcionais

- RF.US 2

Pré-Condições

- O usuário deve possuir uma conta no sistema.

Garantias de Sucesso

- O usuário tem pleno acesso ao sistema.

Cenário Principal de Sucesso

1. O usuário acessa a página de *login*;
2. O usuário insere suas credenciais;
3. O sistema autoriza o acesso;
4. O sistema redireciona o usuário para a página inicial.

Extensões

3a. O sistema falha na autorização de acesso

1. O sistema exibe o motivo da falha, permitindo ao usuário inserir novamente as credenciais, retornando ao CPS, no passo 2.

- **[UC03] Gerenciar Câmeras**

Requisitos Funcionais

- RF.CR 1

Pré-Condições

- O usuário deve estar autenticado no sistema.

Garantias de Sucesso

- A lis*tag*em de câmera permanecer sempre atualizada;
- Todas as câmeras na lis*tag*em podem ser aplicadas ao *pipeline*.

Cenário Principal de Sucesso

1. O usuário navega para gerenciamento de câmeras;
2. O sistema mostra a lista com todas as câmeras já cadastradas;
3. O usuário acessa o cadastro ou edição de câmera;
4. O usuário preenche o formulário com as informações da câmera;
5. O sistema realiza a operação.

Extensões

3a. Exclusão de câmera

1. O usuário seleciona a opção de excluir a câmera;
2. O usuário confirma a remoção;
3. O sistema exclui a câmera.

3b. Pesquisa de câmera

1. O usuário informa o nome da câmera;
2. O sistema exibe todas as câmeras que contém esse nome.

3c. Visualização das imagens da câmera

1. O usuário seleciona a câmera;
2. O sistema busca as imagens em tempo real pelo endereço da câmera que foi cadastrado;

3. O sistema exibe as imagens capturadas.

5a. Falha na operação

1. O sistema mostra o motivo da falha, permitindo ao usuário realizar a operação novamente consoante a solicitação do sistema.

- **[UC04] Gerenciar Pipelines**

Requisitos Funcionais

- RF.PL 1
- RF.PL 3

Pré-Condições

- O usuário deve estar autenticado no sistema.

Garantias de Sucesso

- O sistema deve manter a listagem *pipeline* atualizada.

Cenário Principal de Sucesso

1. O usuário acessa a página de *pipelines*;
2. O sistema mostra a listagem de *pipelines*;
3. O usuário acessa o cadastro de *pipeline*;
4. O usuário preenche o formulário com as informações do *pipeline*;
5. O sistema registra o *pipeline*

Extensões

3a. Ativamento de *pipeline*

1. O usuário ativa o *pipeline*
2. O sistema permite o processamento de imagens pelo *pipeline*

3b. Pesquisa de *pipeline*

1. O usuário informa o nome do *pipeline*
2. O sistema exibe os *pipelines* que contém esse nome

- **[UC05] Adicionar serviço**

Requisitos Funcionais

- RF.PL 2

Pré-Condições

- O usuário deve estar autenticado no sistema;
- O usuário deve ter registrado um *pipeline*.

Garantias de Sucesso

- O sistema deve gerar um fluxo automático para o serviço no *pipeline*;
- O sistema salva o *pipeline* com o serviço, gerando uma versão no histórico.

Cenário Principal de Sucesso

1. O usuário navega para a edição de *pipeline*;
2. O sistema lista os serviços disponíveis;
3. O usuário seleciona um serviço;
4. O usuário configura os parâmetros do serviço;
5. O sistema valida os valores inseridos no parâmetro.

Extensões

5a. Falha na verificação dos parâmetros do serviço

1. O sistema informa a falha, retornando ao passo 4 do CPS.

- **[UC06] Organizar fluxo**

Requisitos Funcionais

- RF.PL 2

Pré-Condições

- O usuário deve estar autenticado no sistema;
- O usuário deve ter registrado um *pipeline*;
- O *pipeline* deve ter ao menos um serviço.

Garantias de Sucesso

- O *pipeline* deve estar bem organizado para sua aplicação em imagens;
- Uma nova versão do *pipeline* deve ser salvo no histórico, possibilitando a restauração do *pipeline* com fluxo anterior.

Cenário Principal de Sucesso

1. O usuário navega para o fluxo do *pipeline*;
2. O usuário define o fluxo do *pipeline*;
3. O sistema valida o fluxo do *pipeline*;

4. O sistema salva o *pipeline*.

Extensões

3a. Falha ao verificar o fluxo

1. O sistema informa que o *pipeline* possui serviço que faz referência a si mesmo, seja direta ou indiretamente;
2. O usuário pode corrigir o fluxo ou cancelar.

- **[UC07] Gerar pré-visualização**

Requisitos Funcionais

- RF.PL 4

Pré-Condições

- O usuário deve estar autenticado no sistema;
- O usuário deve ter cadastrado um *pipeline* com ao menos um serviço;
- O *pipeline* não pode ter dependência cíclica ou seu fluxo possuir serviços com referência cíclica.

Garantias de Sucesso

- O usuário pode configurar novamente o *pipeline* gerando outra pré-visualização.

Cenário Principal de Sucesso

1. O usuário acessa ou salva o *pipeline*;
2. O sistema realiza o processamento do *pipeline*;
3. O sistema exibe o resultado do processamento.

Extensões

2a. Falha no processamento de *pipeline*

1. O sistema informa a falha;
2. O usuário pode configurar novamente o *pipeline*.

- **[UC08] Restaurar versão**

Requisitos Funcionais

- RF.PL 6

Pré-Condições

- O usuário deve estar autenticado no sistema;
- O usuário deve ter registrado um *pipeline*;

Garantias de Sucesso

- O sistema deve restaurar toda a configuração do *pipeline*, incluindo o fluxo dos serviços.

Cenário Principal de Sucesso

1. O usuário navega para a página de histórico do *pipeline*;
2. O sistema lista todas as versões do *pipeline* pela data de criação;
3. O usuário seleciona uma opção;
4. O sistema redireciona o usuário para página de edição de *pipeline*;

Extensões

Não possui extensão

- **[UC09] Adicionar pipeline**

Requisitos Funcionais

- RF.PL 2

Pré-Condições

- O usuário deve estar autenticado no sistema;
- O usuário deve ter registrado pelo menos *pipeline*.

Garantias de Sucesso

- O sistema deve gerar um fluxo automático para o *pipeline*;
- O sistema gera uma nova versão do *pipeline* no histórico.

Cenário Principal de Sucesso

1. O usuário navega para a edição de *pipeline*;
2. O sistema lista os *pipelines* disponíveis;
3. O usuário seleciona um *pipeline*;
4. O sistema adiciona o *pipeline*;
5. O sistema salva o *pipeline*.

Extensões

4a. Falha ao adicionar o *pipeline*

1. O sistema informa que há uma dependência cíclica entre o *pipeline* que está sendo adicionado e o próprio *pipeline* em configuração, retornando ao passo 3 do CPS.

- **[UC10] Gerenciar serviços**

Requisitos Funcionais

- RF.MS 1

Pré-Condições

- O usuário deve estar autenticado no sistema.

Garantias de Sucesso

- A lista dos serviços deve permanecer atualizada;
- Após o cadastro o usuário pode utilizar o serviço em qualquer *pipeline*;
- Os *pipelines* que contém o serviço que será excluído continuaram em funcionamento, sendo gerado um fluxo automático para reparar a remoção do serviço.

Cenário Principal de Sucesso

1. O usuário acessa a página de gerenciamento de serviços;
2. O sistema lista com todos os serviços cadastrados;
3. O usuário cadastra ou edita um serviço;
4. O usuário preenche o formulário com as informações do serviço e dos parâmetros;
5. O sistema realiza a operação.

Extensões

3a. Exclusão de serviço

1. O usuário seleciona a opção de excluir;
2. O sistema informa que a exclusão do serviço acarretará na eliminação do serviço em todos *pipelines*;
3. O usuário confirma a remoção;
4. O sistema exclui o serviço.

3b. Pesquisa de serviço

1. O usuário informa o nome do serviço que deseja;
2. O sistema lista os serviços com o nome informado.

5a. Falha na operação

1. O sistema mostra o motivo da falha;
2. O sistema permite a realização da operação novamente.

- **[UC11] Aplicar pipeline**

Requisitos Funcionais

- RF.PL 5

Pré-Condições

- O usuário deve estar autenticado no sistema;
- O usuário deve ter cadastrado um *pipeline*;
- O *pipeline* deve estar ativo e devidamente configurado, com os serviços, os parâmetros e o fluxo bem definido.

Garantias de Sucesso

- O resultado do processamento da imagem ou vídeo poderá se baixado pelo usuário para armazenamento em seu próprio computador;

Cenário Principal de Sucesso

1. O usuário navega para a página de processamento de *pipeline*;
2. O usuário seleciona a câmera ou realiza o *upload* da imagem ou vídeo;
3. O usuário escolhe o *pipeline* a ser aplicado;
4. O sistema processa o *pipeline*;
5. O sistema exibe o resultado.

Extensões

4a. O sistema falha no processamento do *pipeline*

1. O usuário pode realizar o procedimento novamente ou cancelar e ir configurar o *pipeline*.

2a. Falha ao fazer o *upload* da mídia

1. O sistema informa que o formato da mídia é incompatível e quais formatos são compatíveis;
2. O usuário pode tentar novamente, voltando ao passo 2 do CPS.

- **[UC12] Baixar resultado**

Requisitos Funcionais

- RF.PL 5

Pré-Condições

- O usuário deve estar autenticado no sistema;
- O usuário deve ter processado o *pipeline* com uma mídia..

Garantias de Sucesso

- O formato da mídia baixado deve ser igual ao da imagem ou vídeo processado.

Cenário Principal de Sucesso

1. O usuário realiza o processamento de *pipeline* com imagem ou vídeo;
2. O sistema exibe o resultado do processamento e disponibiliza uma opção para download;
3. O usuário baixa o arquivo para seu computador.

Extensões

Não há extensão

- **[UC13] Alterar senha**

Requisitos Funcionais

- RF.US 2

Pré-Condições

- O usuário deve estar autenticado no sistema.

Garantias de Sucesso

- O usuário deve usar a nova senha em sua próxima autenticação no sistema.

Cenário Principal de Sucesso

1. O usuário navega para página de alterar senha;
2. O usuário preenche o formulário com a senha atual e nova senha;
3. O sistema altera a senha.

Extensões

3a. O sistema falha ao alterar a senha

1. O sistema informa o motivo da falha;
2. O usuário pode tentar novamente ou desistir de alterar sua senha atual.

- **[UC14] Recuperar senha**

Requisitos Funcionais

- RF.US 2

Pré-Condições

- O usuário deve ter conta no sistema.

Garantias de Sucesso

- O usuário pode fazer a autenticação no sistema com a nova senha.

Cenário Principal de Sucesso

1. O usuário acessa a página de recuperação de senha;
2. O usuário informa seu e-mail;
3. O sistema verifica o e-mail;
4. O sistema envia o *link* para recuperação de senha por e-mail;
5. O usuário aciona o *link*;
6. O usuário informa a nova senha;
7. O sistema altera a senha;
8. O sistema redireciona o usuário para a página de *login*.

Extensões

3a. Falha na verificação do e-mail

1. O sistema informa que o e-mail é invalido;
2. O usuário pode inserir novamente o e-mail.

4a. O sistema falha ao enviar e-mail

1. O usuário pode tentar novamente.

5a. Falha ao acionar o *link*

1. O usuário pode refazer todo o CPS.

7a. Falha na alteração de senha

1. O sistema informa o motivo;
2. O usuário pode tentar novamente.

3.2.1.2 DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência "Demonstra as interações entre diferentes objetos na execução de uma operação, destacando ainda a ordem em que tais

ações acontecem num intervalo de tempo. A sequência em que as diversas operações são executadas ocorre na vertical, de cima para baixo." (DEVMEDIA, 2023).

Uma das principais e mais recorrentes atividades de um usuário é a de configuração de um *pipeline*. Para representar visualmente as interações entre o usuário, *interface* e a API do *software* no processo de configuração de um *pipeline*, foi desenvolvido um diagrama de sequência que descreve as principais funcionalidades desta atividade, disposto logo abaixo na Figura 2.

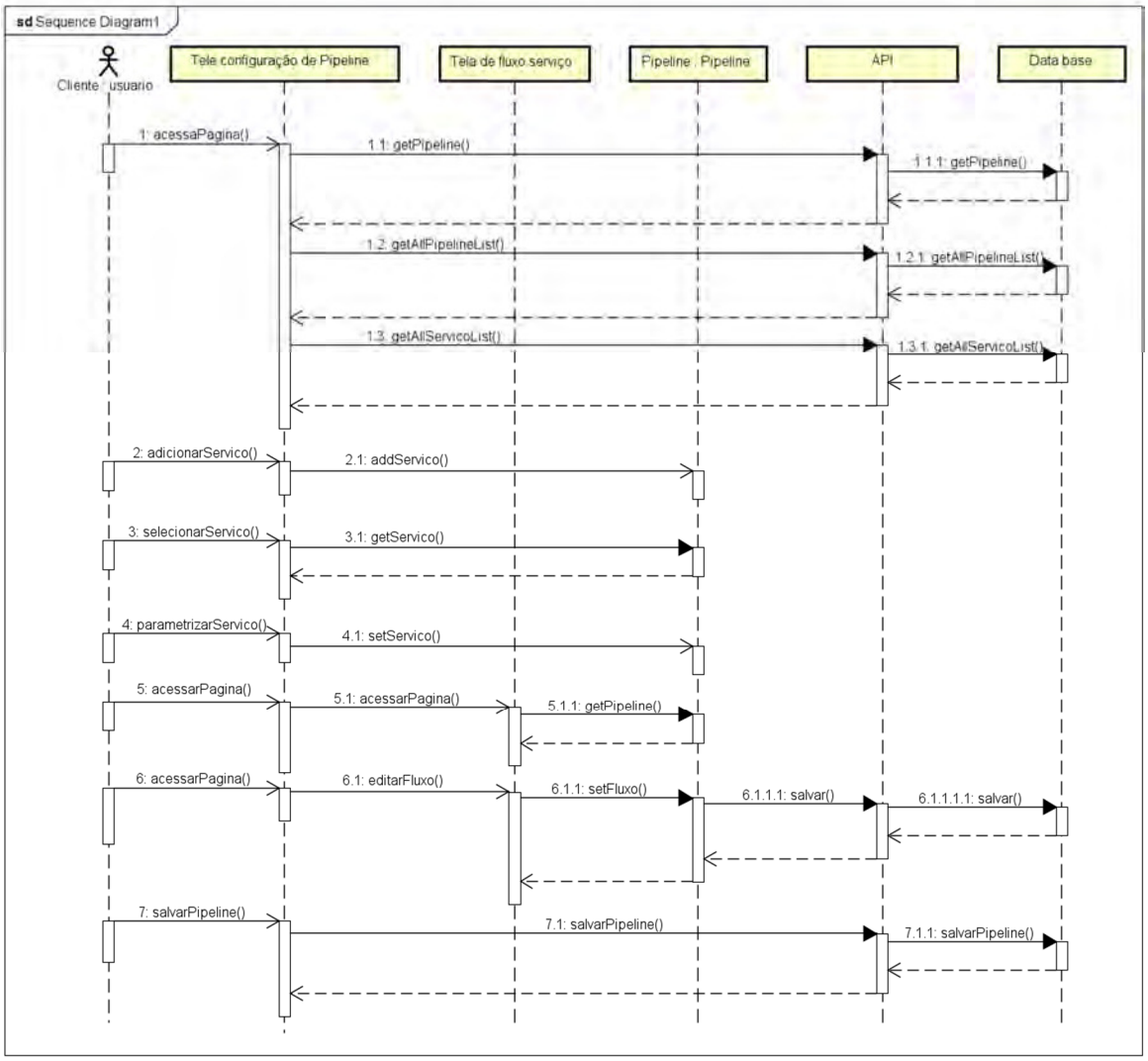


Figura 2 - Diagrama de sequência, configuração de *Pipeline*

Como apresentado na figura acima, a *interface* de configuração de *pipeline* é composta por duas telas. A tela de configuração de *pipeline* é a primeira

acessada pelo usuário após selecionar o *pipeline* que deseja configurar, dando início a sequência. Após esse evento, a *interface* (*Frontend*) irá solicitar à API (*Backend*) todas as informações referentes ao *pipeline* em questão. Dado que recebeu estas informações, as apresenta ao usuário que, a partir de agora, pode seguir com as funcionalidades de configuração.

Ainda na tela de configuração de *pipeline*, o usuário pode interagir com a *interface* para adicionar novos serviços ao *pipeline*, assim como selecionar estes novos serviços ou serviços já existentes no *pipeline* para exibir e parametrizar seus atributos. Estas alterações realizadas no *pipeline* afetam a instância de *pipeline* recuperada inicialmente que está sobre posse do *Frontend*, mas que não será persistida no banco de dados até que o usuário interaja novamente com a *interface* para salvar as alterações realizadas.

Já na tela de fluxo de serviço, que só pode ser acessada pelo usuário mediante a interação com a tela de configuração de *pipeline*, as alterações devem ser salvas ou serão descartadas ao retornar à página anterior. Após o usuário interagir com a *interface* para salvar o fluxo, o mesmo é validado e persistido através da interação entre a *interface* e a API.

3.2.1.3 Diagrama de atividades

Segundo o guia de tipos de diagramas UML do *Create*ly, "Os diagramas de atividade representam fluxos de trabalho de uma forma gráfica. Podem ser utilizados para descrever o fluxo de trabalho empresarial ou o fluxo de trabalho operacional de qualquer componente de um sistema" (*Create*ly, 2023).

Para um melhor entendimento sobre a divisão de responsabilidades entre o usuário e o sistema na configuração de um *pipeline*, foi desenvolvido um diagrama de atividades que dispõe do fluxo de ações para a realização de tal funcionalidade.

Na Figura 3, pode-se ver o fluxo desta funcionalidade. Divide-se em duas colunas, usuário e sistema, especificando assim quem é o responsável pela sua execução.

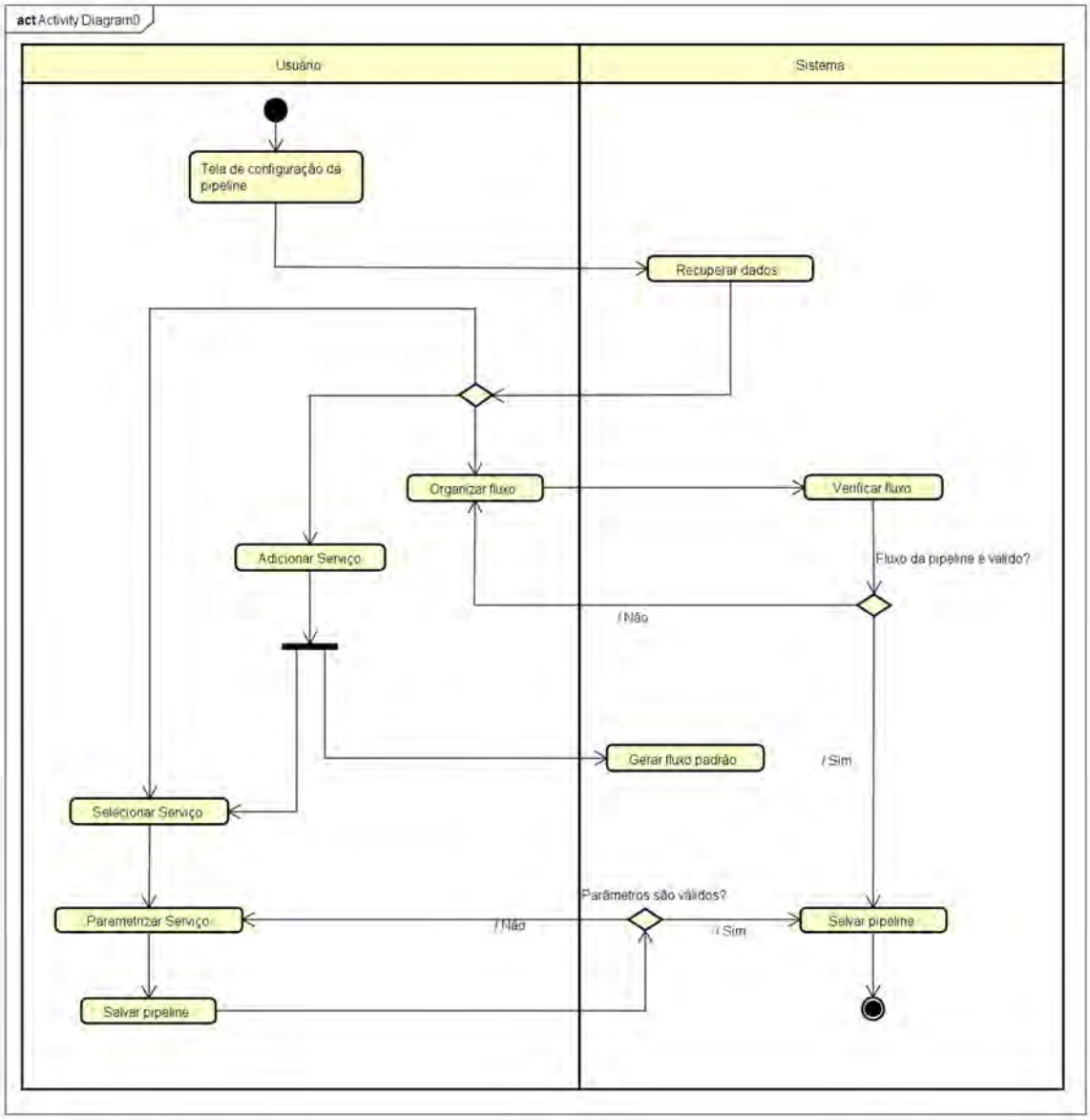


Figura 3 - Diagrama de atividades, configuração de *Pipeline*

A atividade é iniciada com o usuário abrindo a tela de configuração de um *pipeline*, assim como mostrado na Figura 3. O contexto pode ser de um *pipeline* já previamente configurado ou de um *pipeline* recém criado, em ambos os cenários o fluxo desta funcionalidade permanece o mesmo.

Após esta iniciativa do usuário, o sistema encarrega-se de recuperar do banco de dados, as informações pertinentes ao *pipeline* em questão, e apresentá-las visualmente na página.

Sendo assim, o usuário tem como possibilidade:

- Adição de novos serviços, sendo o sistema responsável por gerar um fluxo padrão a partir desta adição.

- Seleção de um serviço, permitindo a alteração dos valores dos parâmetros deste serviço. Sendo o sistema responsável pela validação destes parâmetros.

- Organização de fluxo, sendo o sistema responsável pela validação do fluxo criado pelo usuário.

Além das validações e geração de fluxo padrão citadas anteriormente, o sistema também é responsável por permitir que o *pipeline* seja salvo, persistindo as novas informações no banco de dados. Porém, caso detectado qualquer inconsistência na validação das informações passadas pelo usuário, o sistema fica responsável por impedir a persistência dos dados e apresentar o problema ao usuário, que por sua vez poderá repetir a etapa anterior à validação.

3.2.2 Diagramas estruturais

*Model*os estruturais mostram a forma como o sistema organiza seus componentes e respectivos relacionamentos. Quando o *model*o apresenta a estrutura do projeto do sistema diz-se que o *model*o é estático, já quando a organização representa o sistema em execução diz-se que o *model*o é dinâmico (SOMMERVILLE, 2011).

3.2.2.1 Diagrama de classes

Conforme mencionado em um artigo do *Create*ly, "Os diagramas de classes são o principal componente de qualquer solução orientada a objetos. Mostra as classes em um sistema, atributos e operações de cada classe e a relação entre cada classe" (*Create*ly, 2023).

O diagrama de classes foi construído para possibilitar uma visualização simplificada de todo o projeto estrutural e relacionamento lógico do sistema.

Sendo assim, serviu de direcionamento para a equipe de desenvolvimento, pois nele contém as principais informações para o planejamento de desenvolvimento do projeto.

Dentre essas informações, estão todas as classes que compõem o projeto e sua nomenclatura, os atributos de que compõem cada classe individualmente, e os relacionamentos entre estas classes, trazendo uma perspectiva sobre as dependências e responsabilidades de cada uma. Conforme apresentado na Figura 4 - Diagrama de classes, logo abaixo.

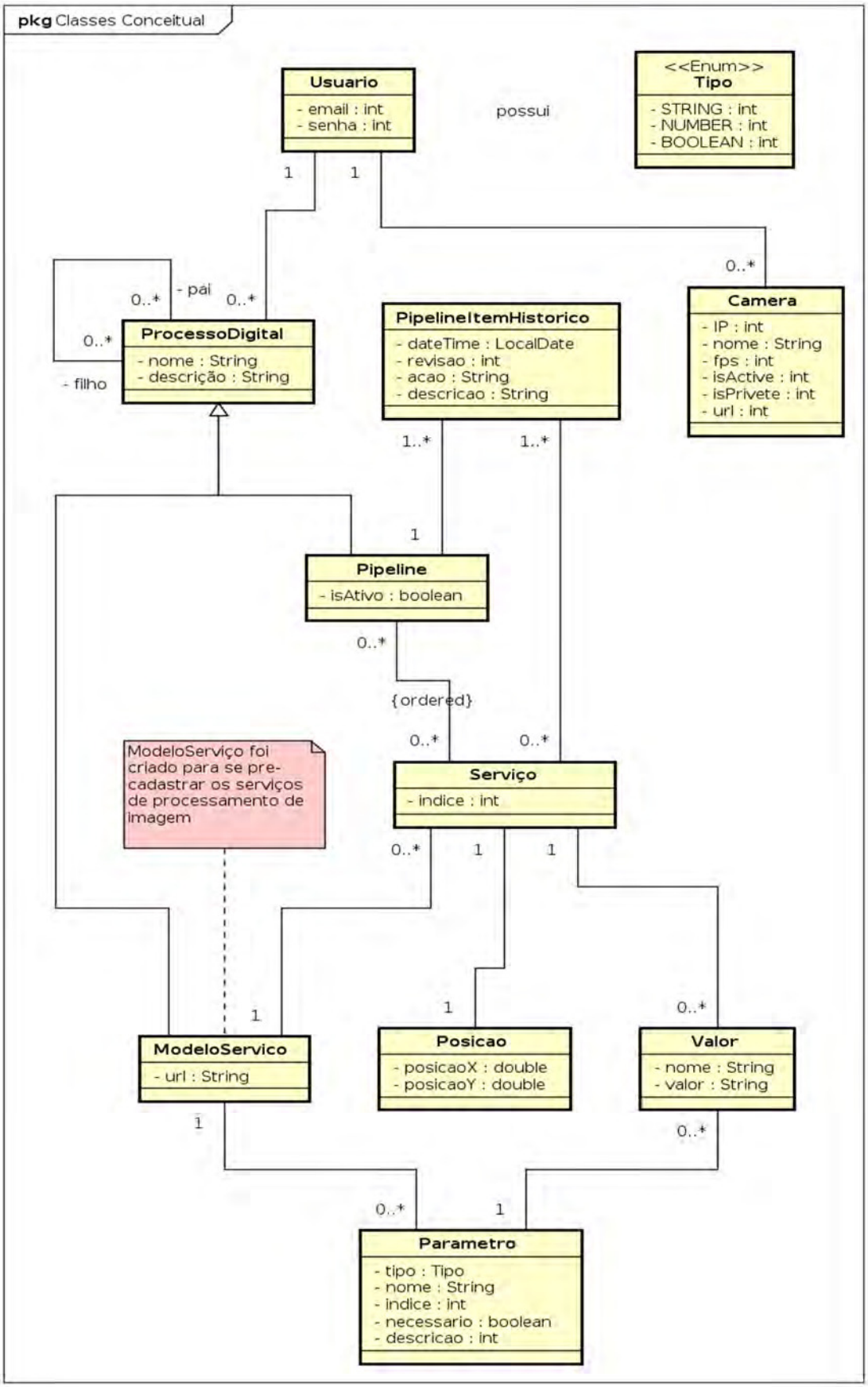


Figura 4 - Diagrama de classes

3.2.2.2 Diagrama de componentes

Conforme mencionado na documentação da IBM, "Na UML, os diagramas de componentes mostram a estrutura do sistema de *software*, que descreve os componentes do *software*, suas *interface*s e suas dependências" (IBM, 2023).

Seguindo o *model*o estrutural planejado para o *software*, foram criados dois diagramas de componentes, como visto na Figura 5 e Figura 6, *Backend* e *Frontend* respectivamente apresentados.

Na Figura 5, o diagrama apresenta uma representação visual de um nível alto de visualização do projeto, onde ver-se a estrutura organizacional da aplicação JAVA, englobada pelo Spring *framework*, tecnologia fundamental na estrutura do *software*, dividida em pom.xml, documento onde se encontram as dependências das principais tecnologias utilizadas, necessárias para a execução do *software*, e Imagempipeline, pacote onde estão centralizados todas as entidades, seus respectivos setores e relacionamentos.

O *Backend* do *software* foi pensado e desenvolvido seguindo a arquitetura de *software* MVC (*Model-View-Controller*), onde setor de *resources* que representa o *controller*, é responsável por intermediar requisições da API REST, e o setor de serviço, que por sua vez fica encarregado de prover soluções através do gerenciamento de todo o setor de *model* e conexão com o banco de dados, além do gerenciamento e envio de requisições à API externa de processamento de dados.

A API de processamento de dados, representado como um subsistema no diagrama, é uma API externa, desenvolvida na linguagem Python, responsável unicamente por receber um arquivo de mídia, juntamente a dados de instruções referentes ao processamento deste arquivo, aplicar os devidos processos e por fim devolver o arquivo resultante ao solicitante.

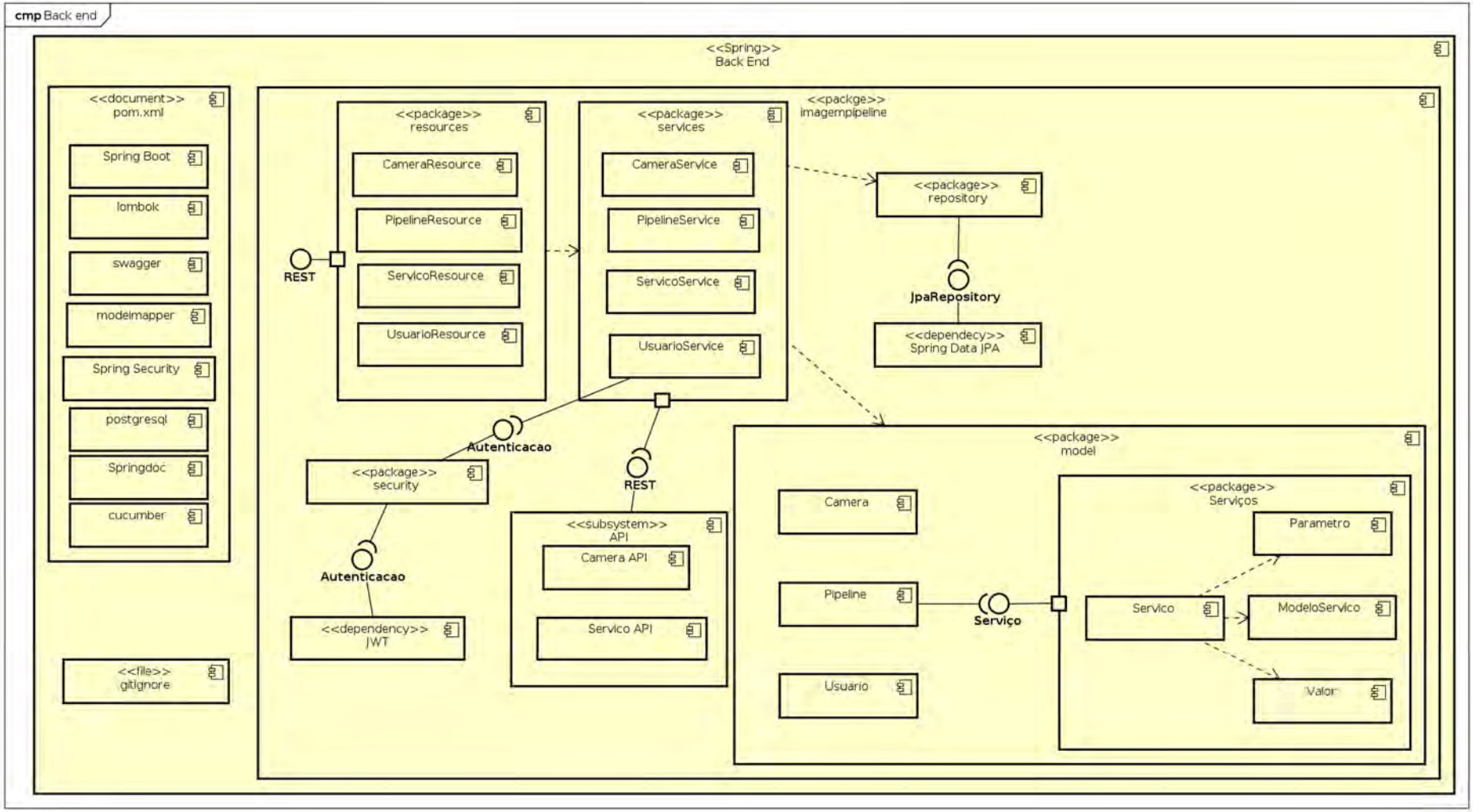


Figura 5 - Diagrama de componentes: *Backend*

O diagrama de componentes na Figura 6, apresenta a estrutura organizacional de toda a aplicação *Frontend* do *software* que, como visto no diagrama, é baseada na estrutura padrão gerada pela biblioteca React.js.

No primeiro nível, estão arquivos de configuração básica da aplicação, como o package.json, onde estão concentradas as declarações de dependências da aplicação, assim como as pastas public e src.

No interior da pasta public pode-se encontrar os arquivos index.html e styles.css, arquivos principais para criação de telas e gerenciamento de componentes.

Na pasta src, estão agrupados componentes, páginas e demais arquivos de serviços e configurações. Todos os componentes da aplicação concentram-se na subpasta components, onde cada componente constitui-se de um arquivo ".jsx" onde se encontra toda a lógica do componente, e um arquivo ".css" onde são declaradas particularidades de estilo de cada componente.

Todas as páginas da aplicação estão na subpasta *pages*. Assim como os componentes, as páginas são compostas por um arquivo ".jsx" e um arquivo ".css". Dentro da pasta src ainda existem arquivos importantes como app.js e outras subdivisões como as pastas *service*, routes e assets responsáveis respectivamente por agrupar arquivos de serviço, separar arquivo de configuração de rotas da aplicação e agrupar arquivos de apoio material ou de mídia.

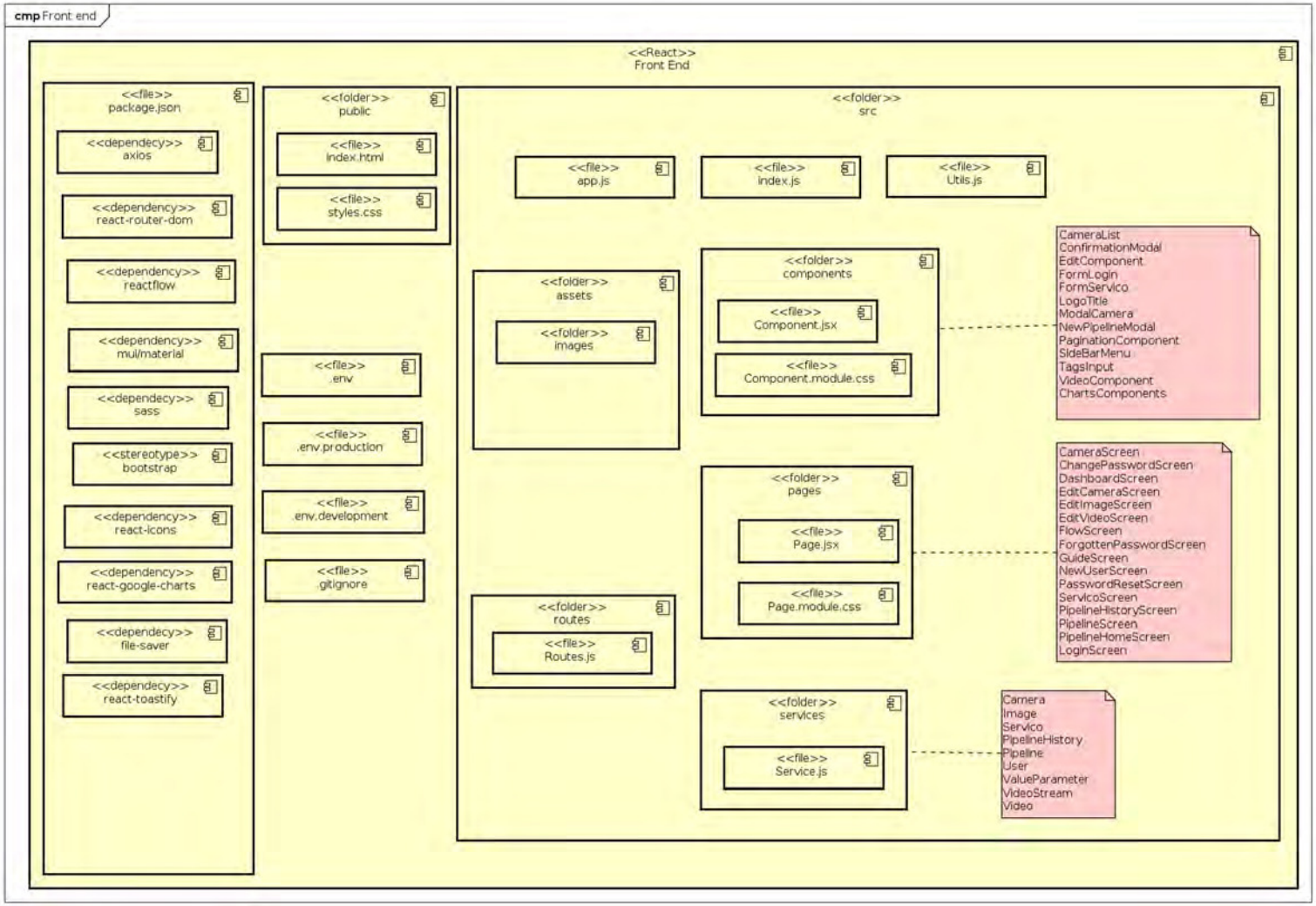


Figura 6 - Diagrama de componentes: *Frontend*

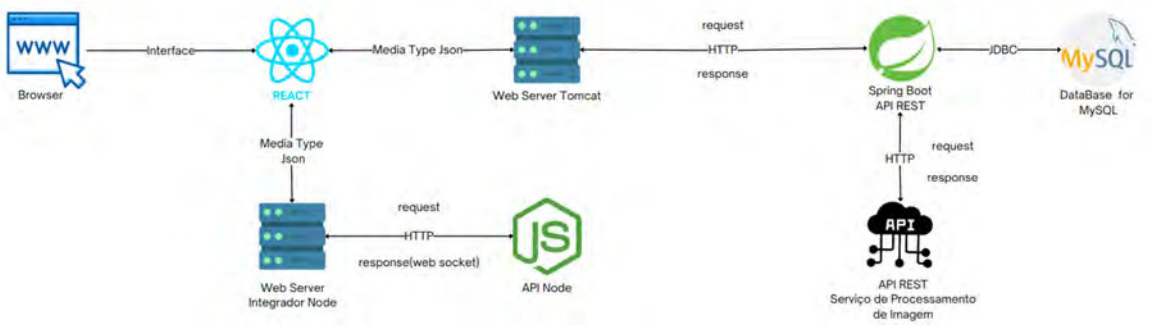
3.3 PROJETO ARQUITETURAL

A arquitetura do *software* detalhada na seção 3.2.2.2 foi dividida em duas aplicações, a aplicação *backend* e a aplicação *frontend*, para separar as responsabilidades e garantir um melhor gerenciamento.

3.3.1 Visão de Componentes

A comunicação entre as duas aplicações e o gerenciamento geral foram definidos da seguinte forma:

Figura 7 - Visão dos componentes



Fonte: Fonte própria.

3.3.1.1 COMUNICAÇÃO ENTRE O BD E O NEGÓCIO:

A comunicação entre o Banco de Dados (PostgreSQL) e o negócio é feita em base JDBC, com a utilização de API padrão Java Persistence API - JPA com Hibernate para o mapeamento objeto-relacional.

3.3.1.2 COMUNICAÇÃO ENTRE O NEGÓCIO E O SERVIDOR WEB:

A comunicação entre o negócio e o servidor *WEB* é realizada através do protocolo HTTP utilizando a API REST para o consumo de informações estruturadas em um formato JSON.

3.3.1.3 COMUNICAÇÃO ENTRE O SERVIDOR WEB E O VÍDEO STREAM:

O *Video Stream* é uma API criada pela equipe para receber a transmissão de vídeos em tempo real com o objetivo de possibilitar a manipulação de câmeras de segurança conectadas à internet.

A comunicação entre o servidor *Web* e o *Vídeo Stream* é realizada em duas etapas, inicialmente é usado o protocolo HTTP para iniciar uma instância do *Video Stream*, após isso, é feita a comunicação através de *Web Socket*, disponibilizando uma porta para o recebimento do *video stream* do servidor *Web*.

3.3.1.4 COMUNICAÇÃO ENTRE O NEGÓCIO E A API DO SERVIÇO:

Essa comunicação é feita através do protocolo HTTP usando a API REST, onde a conexão com a API é feita de forma dinâmica e o próprio usuário estabelece a conexão, informando os parâmetros necessários para conectar a sua API ou a alguma API externa desejada.

3.3.2 Módulo de Gerenciamento Geral

O Módulo de gerenciamento geral foi dividido da seguinte forma:

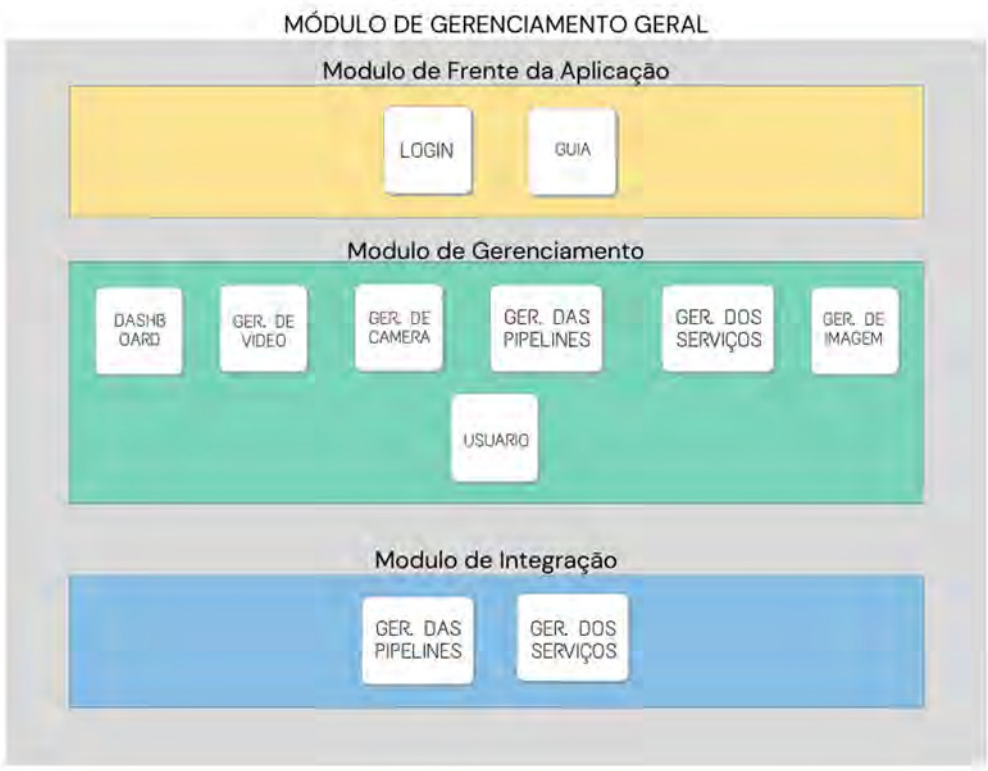


Figura 8 - Módulos do sistema

3.3.2.1 MÓDULO FRENTE DA APLICAÇÃO

Módulo onde serão exibidas, de forma pública, as telas para os usuários que estão *offlines* e com interesse de consumir o sistema.

3.3.2.2 MÓDULO DE GERENCIAMENTO

Módulo que irá gerenciar todo o sistema, como serviços, *pipelines*, entre outros, fundamentais no sistema para o seu pleno funcionamento e objetivo.

3.3.2.3 MÓDULO DE INTEGRAÇÃO

Módulo que integrará com as APIs externas, responsáveis pelos serviços de processamento. Para onde serão enviados os *pipelines* com a lista dos serviços e recebida as informações sobre a execução do *pipeline* e o resultado final da aplicação.

3.4 PROJETO GERENCIAL DO SISTEMA

O gerenciamento do projeto seguiu o *model*o de iterações, que é um conceito matemático que consiste em realizar uma mesma tarefa diversas vezes a fim de aprimorar o resultado, por meio de ciclos curtos de *feedback*, definido pela professora da disciplina.

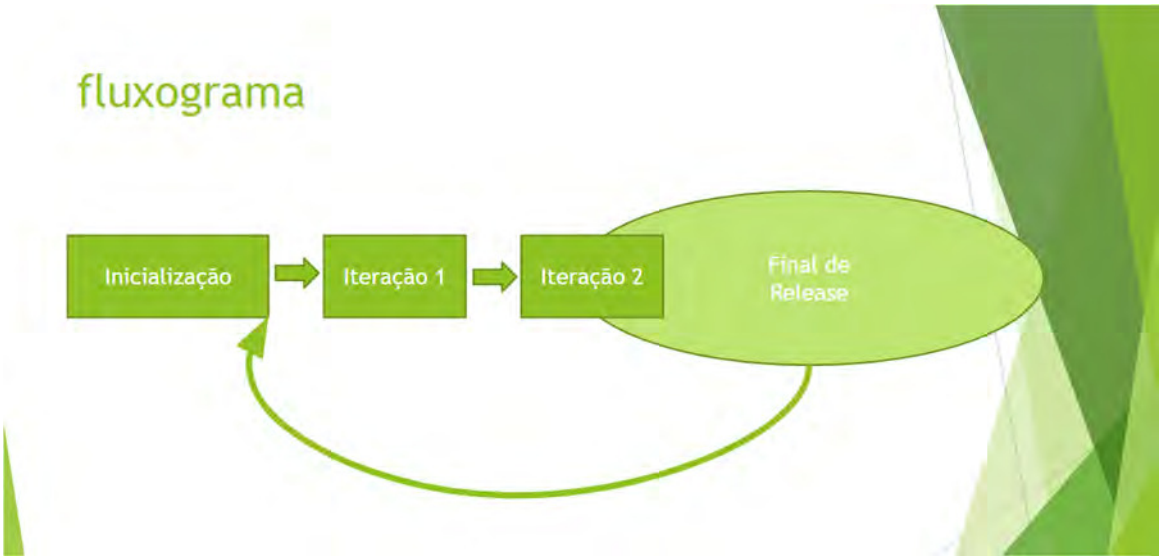


Figura 9 - Fluxograma de iterações

O Desenvolvimento do projeto, desde sua fase inicial até sua conclusão, foi dividido em 6 iterações. Cada iteração possuía um objetivo e itens do *backlog* do projeto a serem implementados para atingir o objetivo desejado. Ao final de cada iteração foi realizada uma reunião com o cliente para receber *feedback* do que foi finalizado e, caso necessário, ajustar funcionalidades até a satisfação do cliente.

Com exceção do membro da equipe Pedro, que ficou encarregado de realizar os testes de cada iteração, o resto da equipe ficou com responsabilidade de *full-stack*, ou seja, implementaram funcionalidades do *backend* e do *frontend*. Como havia três membros exercendo a mesma função, a divisão de itens do *backlog* foi definida de forma equilibrada, estratégica e com base no conhecimento de cada membro sobre as tecnologias requeridas pela iteração.

As iterações foram separadas da seguinte forma:

- Iteração 1

  Objetivo

  Finalizar e testar CRUDs básicos no *frontend* integrando-o com o *backend* e definir a página de guia do usuário.

Tabela 1 - Atividades da iteração 1

Item	Desenvolvimento	Testes	Demonstração para o cliente
CRUD de Serviço	09/09 - 15/09	16/09 - 23/09	26/09
Tela de cadastro de usuário	09/09 - 12/09	13/09 - 19/09	26/09
Tela de guia	13/09 - 19/09	20/09 - 23/09	26/09
Tela de alterar senha	20/09 - 22/09	23/09 - 27/09	26/09
Tela de CRUD de câmera	09/09 - 15/09	16/09 - 22/09	26/09
Envio de Email	09/09 - 15/09	16/09 - 11/09	26/09

Recuperação de senha	09/09 - 20/09	21/09 - 23/09	26/09

Tabela 2 - Divisão de atividades

Aluno	Atividade
Agemiro Alves de Sousa Neto	Tela de cadastro de usuário Tela de guia Tela de alterar senha
Victor Fernando Viana de Macêdo	Tela de CRUD de Serviços
Jordielson Emanuel Caldeira da Silva	Recuperação de senha Tela de CRUD de câmera Envio de Email
Pedro Lucas Gonzaga Prata	Teste Geral

- Iteração 2

  Objetivo

  Finalizar e testar funcionalidades de integração do *pipeline* em um vídeo, câmera e imagem.

Tabela 3 - Atividades da iteração 2

Item	Desenvolvimento	Testes	Demonstração para o cliente
Tela de adição do *Pipeline* na imagem	30/09 - 6/10	06/10 - 12/10	11/10
Tela de adição do *Pipeline* no vídeo	30/09 - 10/10	10/10 - 12/10	11/10
Tela de adição do *Pipeline* na	30/09 - 10/10	10/10 - 12/10	11/10

câmera			
Serviços de gerenciamento de arquivos (imagem e vídeo) .	30/09 - 06/10	06/10 - 12/10	11/10

Tabela 4 - Divisão de atividades da iteração 2

Aluno	Atividade
Agemiro Alves de Sousa Neto	Tela de adição do *Pipeline* na câmera
Victor Fernando Viana de Macêdo	Tela de adição do *Pipeline* na imagem Tela de adição do *Pipeline* no vídeo
Jordielson Emanuel Caldeira da Silva	Serviços de gerenciamento de arquivos (imagem e vídeo)
Pedro Lucas Gonzaga Prata	Teste Geral

- Iteração 3

  Objetivo

  Finalizar e testar funcionalidades de gerenciamento de *pipeline* e o histórico de edição do *pipeline*

Tabela 5 - Atividades da iteração 3

Item	Desenvolvimento	Testes	Demonstração para o cliente
Excluir *pipeline*	14/10 - 22/10	22/10 - 26/10	27/10
Ativar/Desativar *pipeline*	14/10 - 22/10	22/10 - 22/10	27/10
Criar *pipeline*	14/10 - 22/10	22/10 - 24/10	27/10

Finalizar parâmetros de *pipeline*	14/10 - 22/10	22/10 - 25/10	27/10
Edição do nome do *pipeline*	14/10 - 19/10	21/10 - 21/10	27/10
Carregamento de *pipeline* selecionada	14/10 - 19/10	19/10 - 21/10	27/10
Refazer funcionalidade de Serviços no *pipeline*	14/10 - 19/10	19/10 - 23/10	27/10
Histórico de *pipeline*	14/10 - 22/10	22/10 -25/10	27/10

Tabela 6 - Divisão de atividades da iteração 3

Aluno	Atividade
Agemiro Alves de Sousa Neto	Excluir *pipeline* Ativar/Desativar *pipeline* Criar *pipeline* Finalizar parâmetros de *pipeline* Correção de *bugs* relacionados
Victor Fernando Viana de Macêdo	Edição do nome do *pipeline* Carregamento do *pipeline* selecionado Refazer funcionalidade de serviços no *pipeline* Correção de *bugs* relacionados
Jordielson Emanuel Caldeira da Silva	Tela de histórico do *pipeline* Gerenciamento do histórico do *pipeline* no *backend* Correção de *bugs* relacionados
Pedro Lucas Gonzaga Prata	Teste Geral

- Iteração 4

  Objetivo

Finalizar e testar funcionalidades de gerenciamento do fluxo do *pipeline*.

Tabela 7 - Atividades da iteração 4

Item	Desenvolvimento	Testes	Demonstração para o cliente
Salvar fluxo	28/10 - 04/11	05/11 - 08/11	09/11
Deletar edge	28/10 - 04/11	05/11 - 08/11	09/11
Interface do fluxo	28/10 - 04/11	05/11 - 08/11	09/11
Carregar fluxo	28/10 - 04/11	05/11 - 08/11	09/11
Transformar excluir em botão	28/10 - 04/11	05/11 - 08/11	09/11
Transformar input criar em modal	28/10 - 04/11	05/11 - 08/11	09/11
Criar página inicial de *pipeline* para selecionar *pipeline*	28/10 - 04/11	05/11 - 08/11	09/11
Adicionar dados de manipulação do fluxo do pipeline no *back-end*	28/10 - 04/11	05/11 - 08/11	09/11

Tabela 8 - Divisão de atividades da iteração 4

Aluno	Atividade

Agemiro Alves de Sousa Neto	Transformar excluir em botão Transformar *input* criar em modal Criar página inicial de *pipeline* para selecionar *pipeline*
Victor Fernando Viana de Macêdo	Salvar fluxo Deletar *edge* *Interface* do fluxo Carregar fluxo
Jordielson Emanuel Caldeira da Silva	Adicionar dados de manipulação do fluxo do *pipeline* no *backend*
Pedro Lucas Gonzaga Prata	Teste Geral

- Iteração 5

  Objetivo

  Finalizar *template* de *dashboard* com gráficos *Mockados* (gráfico preenchido com dados fictícios).

Tabela 9 - Atividades da iteração 5

Item	Desenvolvimento	Demonstração para o cliente
Gráficos *Mockados Dashboard*	11/11 - 14/11	23/11
Inicialização da API de processo de Imagem	11/11 - 23/11	23/11
Criar um nó representativo de entrada e de saída do fluxo para ficar esteticamente visível e elegante do ponto de vista da UX	11/11 - 15/11	23/11

Alterar o nome representativo de “PDI” para “Serviço” em todo o sistema.	11/11 - 14/11	23/11

Tabela 10 - Divisão de atividades da iteração 5

Aluno	Atividade
Agemiro Alves de Sousa Neto	Gráficos *Mockados Dashboard* inicialização da API de processo de Imagem
Victor Fernando Viana de Macêdo	Gráficos *Mockados Dashboard* inicialização da API de processo de Imagem Criar um nó representativo de entrada e de saída do fluxo para ficar esteticamente visível e elegante do ponto de vista da UX Alterar o nome representativo de “PDI” para “Serviço” em todo o sistema.
Jordielson Emanuel Caldeira da Silva	Gráficos *Mockados Dashboard* inicialização da API de processo de Imagem
Pedro Lucas Gonzaga Prata	Gráficos *Mockados Dashboard* inicialização da API de processo de Imagem

- Iteração 6

  Objetivo

  Melhorias na estilização geral, na restrição de fluxo e na criação de serviços e *pipeline*.

Tabela 11 - Atividades da iteração 6

Item	Desenvolvimento	Testes	Demonstração para o cliente
Ordenar espaçamento entre os *nodes* renderizados	25/11 - 05/12	06/12 - 13/12	14/12
Proibir *loop* de *edge*	25/11 - 08/12	09/12 - 13/12	14/12
Proibir *input* de ter pai	25/11 - 06/12	07/12 - 13/12	14/12
Proibir *output* de ter filho	25/11 - 02/12	03/12 - 13/12	14/12
Criar parâmetros *select* em serviços	25/11 - 05/12	07/12 - 13/12	14/12
Criar parâmetros cor em serviços	25/11 - 05/12	06/12 - 13/12	14/12
Adicionar descrição no título dos parâmetros de *pipeline*	25/11 - 06/12	08/12 - 13/12	14/12

Tabela 12 - Divisão de atividades da iteração 6

Item	Desenvolvimento	Testes	Demonstração para o cliente
Substituir vídeo por imagem em pré-visualização	25/11 - 06/12	06/12 - 10/12	14/12
Alinhar e padronizar botões e *input* (botões do mesmo tamanho, menu alinhado)	25/11 - 08/12	09/12 - 11/12	14/12

Deixar *input* de nome do *pipeline* mais intuitiva para o usuário saber que da pra editar	25/11 - 07/12	08/12 - 12/12	14/12
Adicionar modal de confirmação para excluir *pipeline*	25/11 - 09/12	10/12 - 12/12	14/12
Refazer botão de fechar no fluxo do *pipeline*	25/11 - 08/12	09/12 - 12/12	14/12

3.5 IMPLEMENTAÇÃO DO SOFTWARE

A implementação do *software* é a etapa que envolve os processos de projeto e a programação do *software*, utilizando como base os artefatos elaborados nas etapas anteriores. Assim, as necessidades da aplicação são consideradas para elaboração de um sistema executável (SOMMERVILLE, 2011).

Nesta seção será descrito o processo de desenvolvimento do *software*, que envolveu os *stakeholders* na validação dos requisitos já implementados e na melhoria de funcionalidades para atender as necessidades dos usuários finais. Desse modo, as alterações propostas e o incremento de novos requisitos foram bem-vindos mesmo durante o desenvolvimento.

O aplicativo *Web* foi desenvolvido pensando em reduzir a taxa de erros, informando o motivo da falha, a fim de permitir uma rápida correção, e aumentar a produtividade permitindo ao usuário a realização de suas tarefas de modo intuitivo e eficiente como, por exemplo, a configuração do fluxo do *pipeline*. Além disso, foi implementado o conceito de *Single Page Application* no qual as páginas são reescritas dinamicamente conforme a obtenção dos dados pelo *back-end*, ao invés de fazer o carregamento constante de páginas.

A *interface* foi implementada com responsividade para a correta adaptação do *layout* com resolução de tela do usuário. Portanto, as imagens, o tamanho dos botões e os demais componentes são redimensionados de acordo com a tela. No entanto, essa responsividade não é totalmente funcional para dispositivos *mobile* já que o sistema foi desenvolvido para ser usado apenas em ambientes *desktop*.

É importante destacar que todas as lis*tag*ens presentes no sistema apresentados nesta seção possuem paginação, visando melhor performance no carregamento das páginas e melhor experiência para o usuário. Acima de 20 itens contidos numa lis*tag*em, um componente com botões de navegação irá aparecer, possibilitando ao usuário navegar entre as páginas e cada página contém até 20 itens.

3.5.1 Páginas de Acesso ao Sistema

A Figura 10 apresenta a página de *login* que possui o formulário com as informações necessárias para o acesso ao sistema, sendo obrigatório o prévio cadastramento do email e senha. Se as credenciais do usuário não forem válidas, o sistema bloqueia o acesso e informa o motivo do erro. Nesta página ainda há as opções para o usuário se cadastrar no sistema, recuperar sua senha e acessar o guia de utilização do sistema.

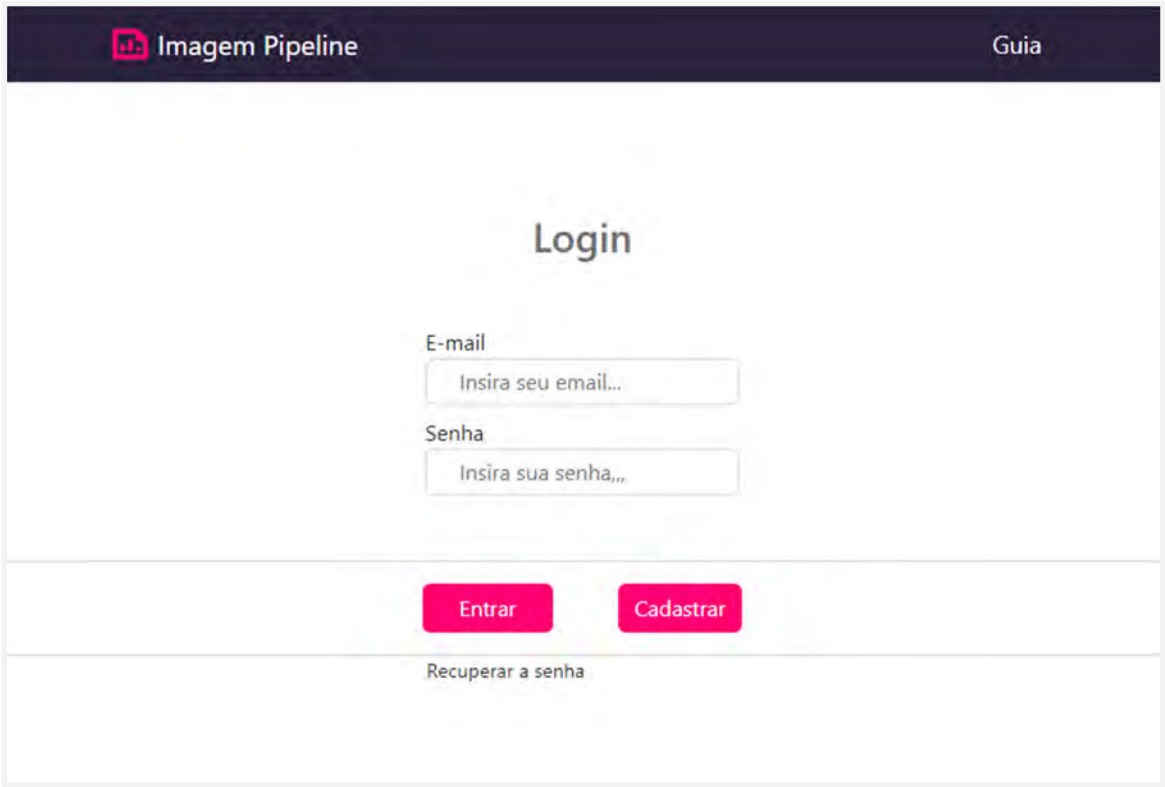


Figura 10 - Página de *Login*

Caso o usuário se esqueça de sua senha, ele poderá recuperar sua conta informando seu email de acesso ao sistema, que envia o *link* da recuperação (Figura 12). Então o usuário pode acessar esse *link* e redefinir a senha preenchendo o formulário (Figura 13). É importante mencionar que o tempo de validade do *link* de recuperação é de 3 horas e só poderá ser usado uma única vez, após esse período, caso a senha não tenha sido redefinida, o usuário deverá solicitar outro email.

Figura 11 - Solicitação de recuperação de senha

Figura 12 - Recuperação de senha

O cadastro de um usuário no sistema é realizado através do preenchimento de um formulário com email e senha, como mostra a Figura 13. Se as informações fornecidas forem invalidadas é exibido o motivo no canto superior da tela.

Figura 13 - Cadastro de usuário

3.5.2 PÁGINA DE GUIA

A Figura 14 mostra o guia do usuário, onde podem ser obtidas informações do sistema através de materiais didáticos com demonstrações dos procedimentos de configuração. Assim, o usuário pode ter uma visão geral das principais funcionalidades do *software*, obter conhecimento de *pipelines*, inclusive como a criação e edição são realizadas, da definição do fluxo de *pipeline* e ainda como ocorrerá a aplicação dos processos pelo *pipeline*.

Figura 14 - Guia de Usuário

3.5.3 PÁGINA DE DASHBOARD

O *dashboard*, representado na Figura 15, tem por objetivo apresentar, de maneira intuitiva e com gráficos, as informações e dados relacionados aos *pipelines* e aos serviços de processamento das imagens. Entretanto, os gráficos e os dados são fictícios, pois esses dados são providos pelas APIs de processamento de imagens do cliente, que até o momento não estão disponíveis, o que impossibilitou o real monitoramento e análise do consumo desses serviços pelo usuário.

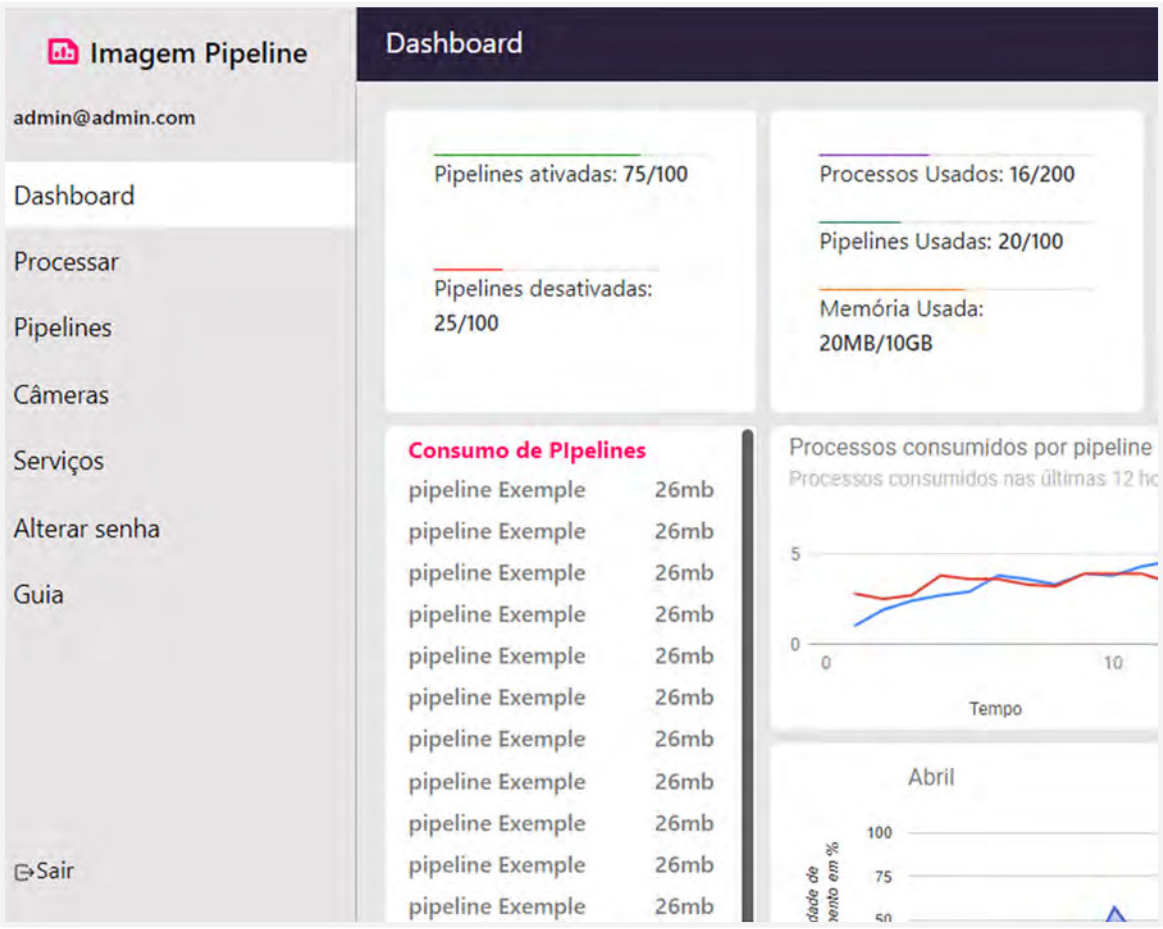


Figura 15 - Página de *Dashboard*

3.5.4 PÁGINA DE PROCESSAMENTO

A página de processamento de mídias permite a aplicação de um *pipeline* em uma imagem, vídeo ou câmera. Esse processo é um fluxo que pode ser dividido em três etapas: selecionar mídia, selecionar *pipeline* e visualizar o resultado.

Na primeira etapa, o usuário pode selecionar uma imagem ou vídeo para processamento, tendo a opção de fazer o *upload* do arquivo desejado, como mostra a Figura 16. Se a mídia a ser processada for proveniente de uma câmera, o sistema lista todas as câmeras disponíveis para que o usuário possa selecionar.

Na segunda etapa, o usuário escolhe um *pipeline* previamente cadastrado e configurado, o qual contém a sequência de serviços e operações que serão aplicados à mídia selecionada, como ilustrado na Figura 17.

Na última etapa, após o processamento da mídia pelos serviços do *pipeline*, o resultado é apresentado ao usuário. A Figura 18 representa essa visualização do resultado.

Ao finalizar o processamento de imagem ou vídeo, o usuário tem a opção de baixar o resultado obtido ou reiniciar o processo, voltando à etapa de seleção de mídia. Portanto, a página desta seção fornece ao usuário etapas intuitivas com um fluxo estruturado para o processamento de mídia.

Figura 16 - Processo de Mídias (Seleção da Mídia)

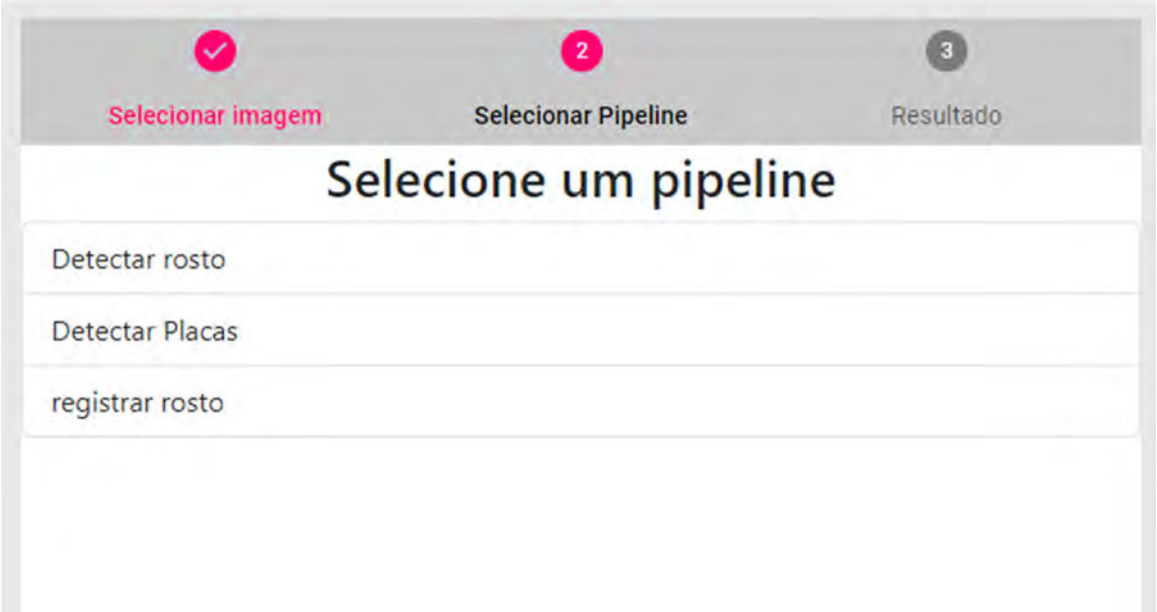


Figura 17 - Processo de Mídias (Seleção do *Pipeline*)

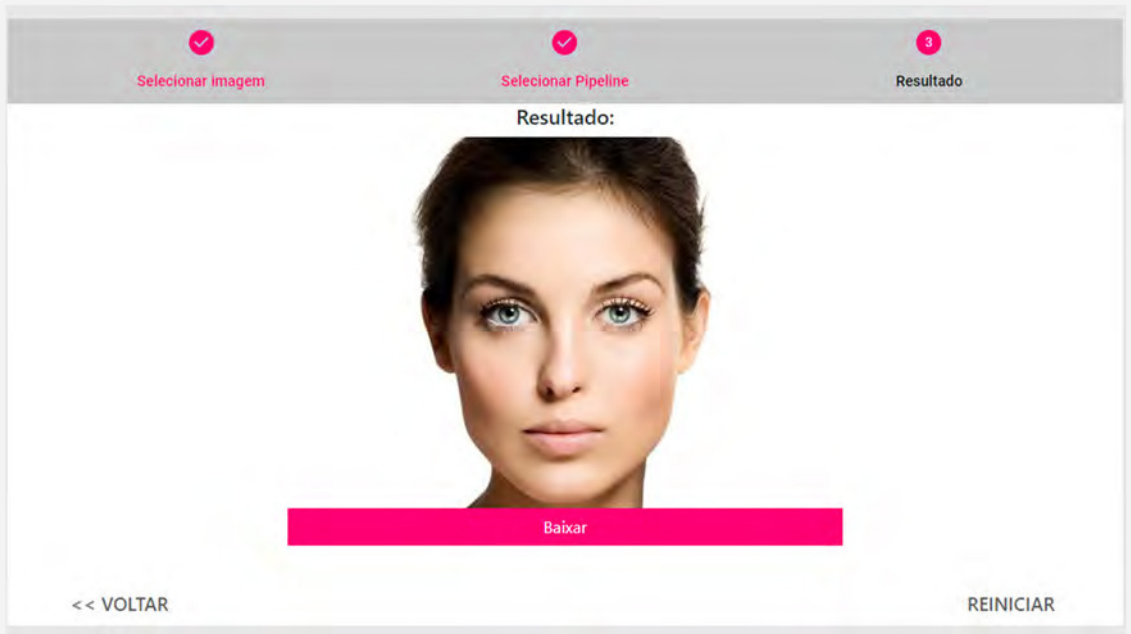


Figura 18 - Processo de Mídias (visualização do processo)

3.5.5 PÁGINA DE PIPELINES

Na página de *pipeline*, o usuário é apresentado a uma lista contendo todos os *pipelines*, onde é possível realizar ações de ativar, desativar, excluir e editar em cada um deles. Ainda na página de *pipelines* é possível filtrar um *pipeline* pelo seu nome na barra de pesquisa e criar um novo *pipeline*. Nesta última opção é aberto um modal onde o usuário precisa informar o nome do novo *pipeline* e clicar em salvar para ser criado, como pode ser visto na Figura 20.

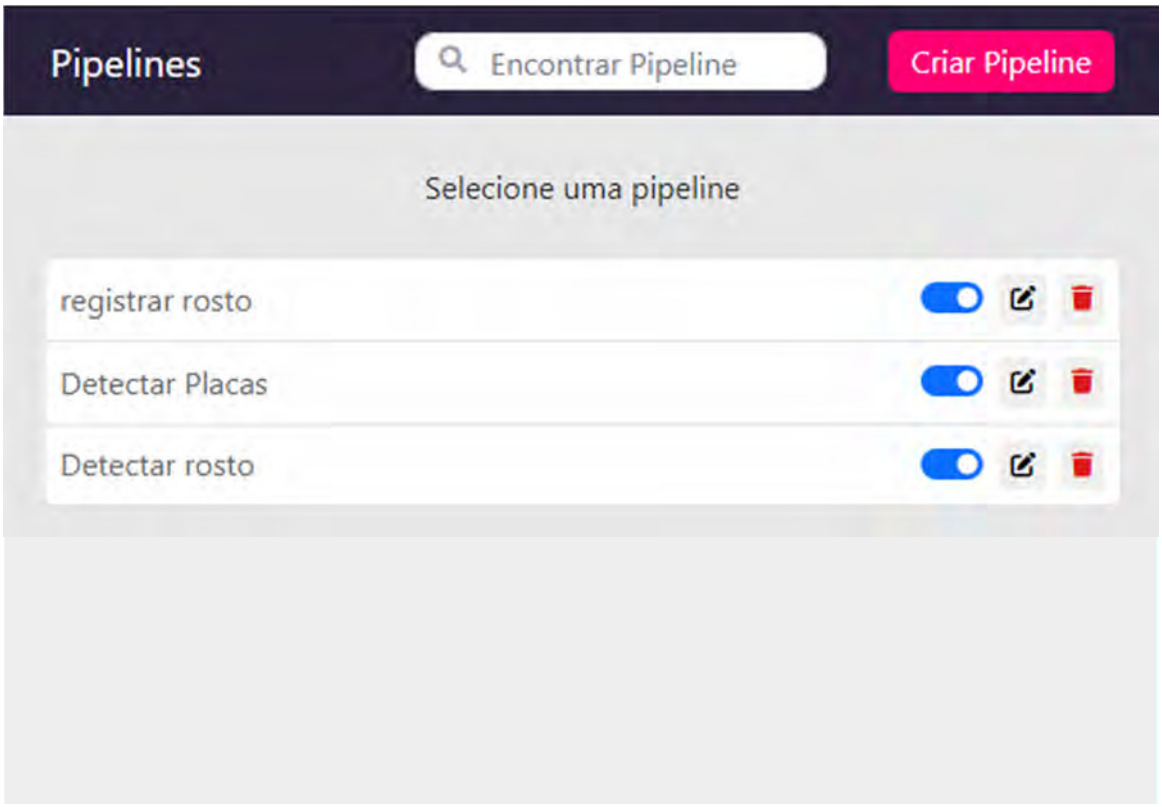


Figura 19 - Lista de *Pipelines*

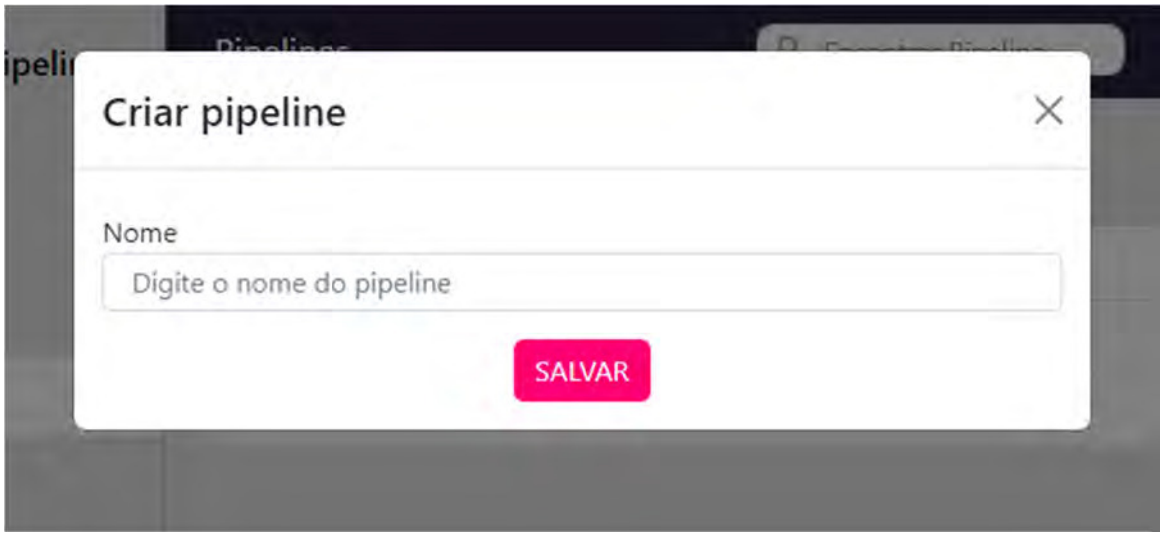


Figura 20 - Cadastro de *Pipeline*

Ao selecionar um *pipeline* da lista o usuário é redirecionado para a página de edição de *pipeline* que, além da presença das funcionalidades contidas na lista de *pipelines* (exceto a funcionalidade de criação), O usuário também pode alterar o nome do *pipeline* clicando no próprio nome ou no ícone de edição ao lado esquerdo do nome do *pipeline*. Também é possível configurar o funcionamento do *pipeline* e visualizar o histórico de alterações.

Na configuração do *pipeline* é possível adicionar serviços ou outros *pipelines* dentro do que está sendo editado. Ao clicar em um serviço adicionado, são apresentados campos de parâmetros requisitados pelo serviço, que podem ser obrigatórios ou não para o seu funcionamento (é informado ao lado do nome do parâmetro). Ainda na configuração, é possível editar o fluxo dos serviços e pipelines e por fim, pré-visualizar o resultado do *pipeline* em uma imagem estática fornecida pela plataforma ao clicar no botão de salvar, como ser visto na Figura 21.

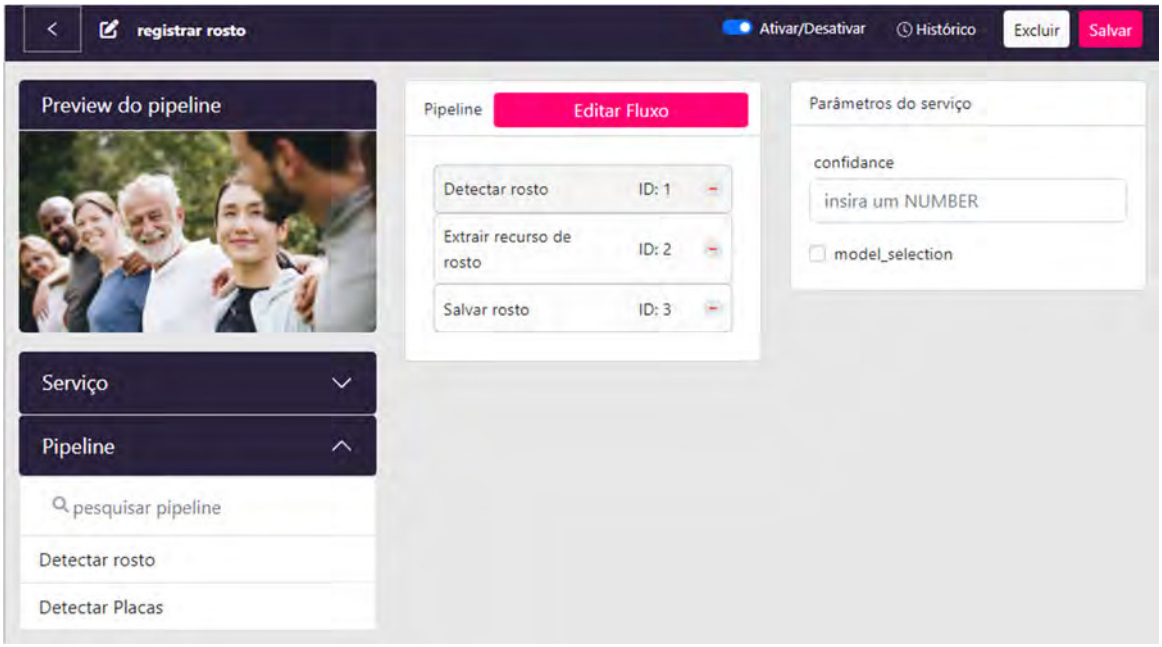


Figura 21 - Edição de *pipeline*

O histórico de *pipeline* contém o histórico de edições realizadas pelo *pipeline* selecionado, com a data e hora da alteração. Esse histórico fornece recurso para descrever a alteração feita, para que o usuário tenha um melhor entendimento a respeito do que foi realizado no canto superior esquerdo existe um botão para retornar ao *pipeline*, como pode ser visto na Figura 22.

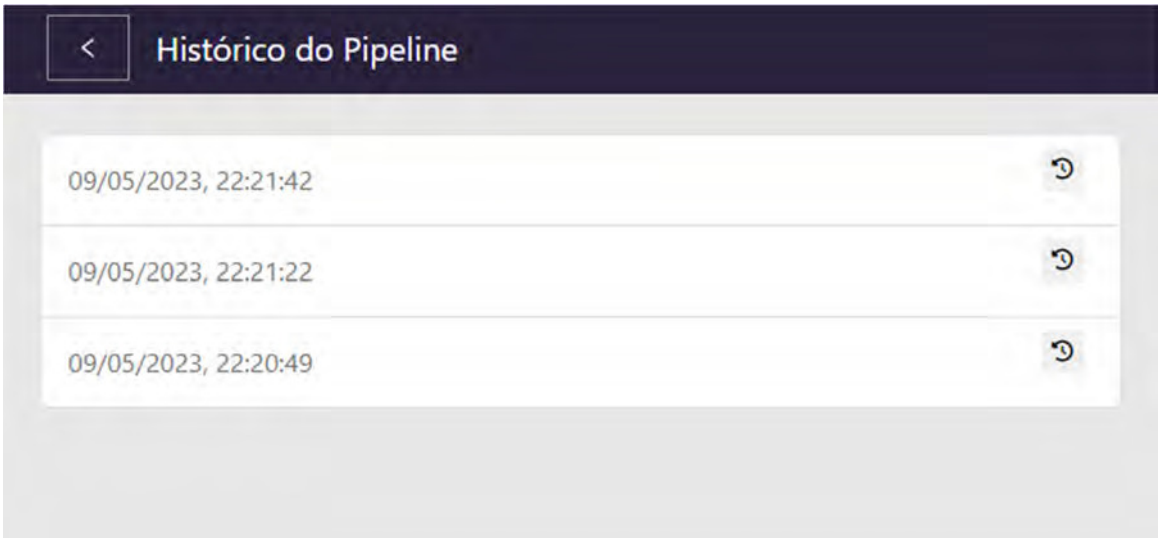


Figura 22 - Histórico do *Pipeline*

Clicando no botão de editar fluxo, o usuário será direcionado para a página de fluxo, onde é configurada a ordem dos processos (serviços e *pipelines*). Nesta página são apresentados componentes quadrados, que representam serviços e *pipelines* com seu nome e número de identificação. Esses componentes podem ser posicionados com e conectados um ao outro com técnica de arrasta e solta com o *mouse.*

Ainda em edição do fluxo (Figura 23), é possível salvar o fluxo, posicionar processos, conectar um processo a outro, excluir conexões e fechar a edição do fluxo para voltar para a janela de edição de *pipeline*.

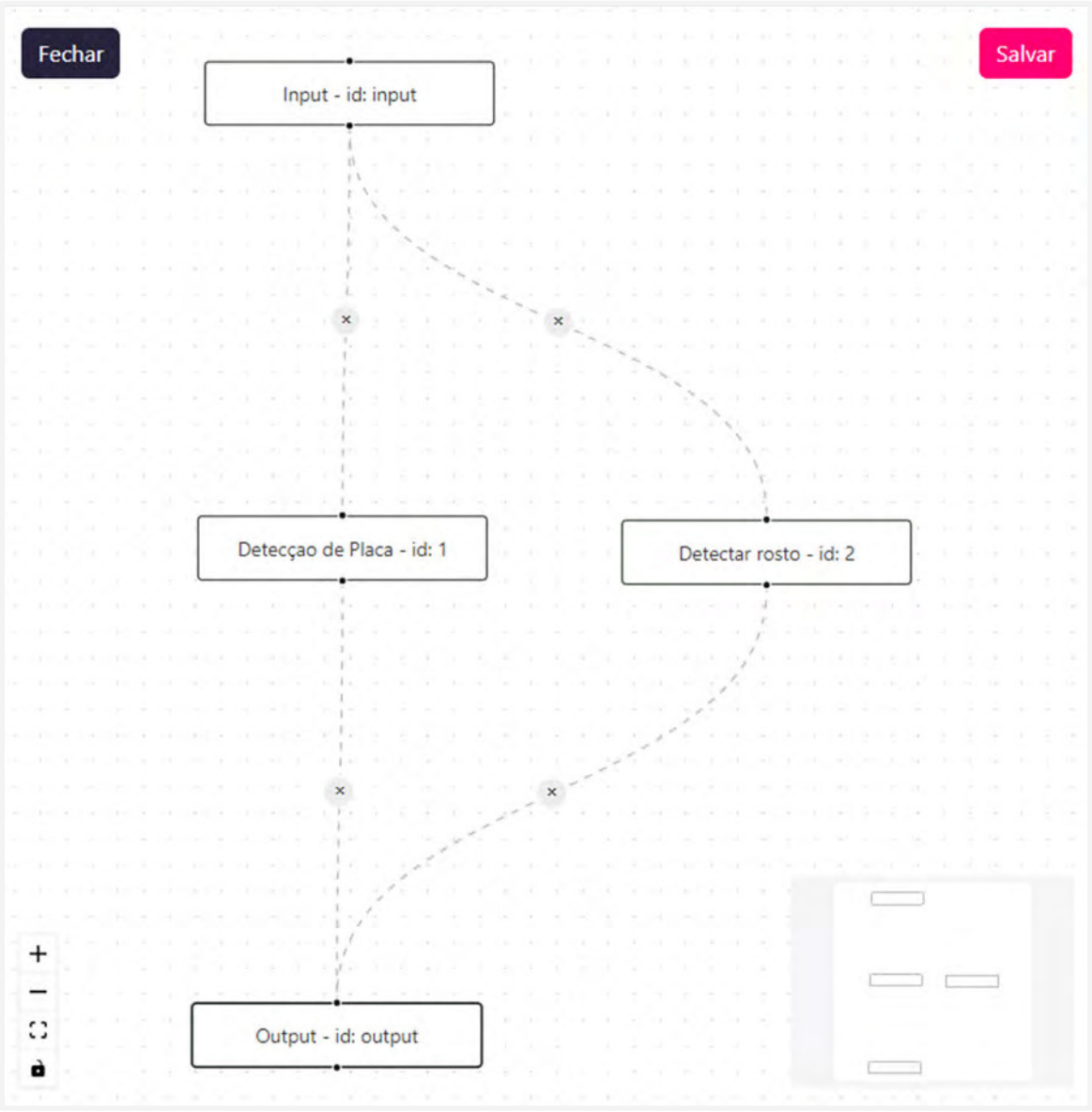


Figura 23 - Fluxo do *Pipeline*

3.5.6 PÁGINA DE CÂMERAS

A página de câmeras possui recursos de pesquisar, para que se possa filtrar uma lista de câmeras pelo nome informado, o botão de adicionar câmera e uma lista de câmeras adicionadas.

Na lista de câmeras, para cada uma é fornecida a ação de visualizar, editar, ativar ou desativar e excluir, como mostrado na Figura 24.

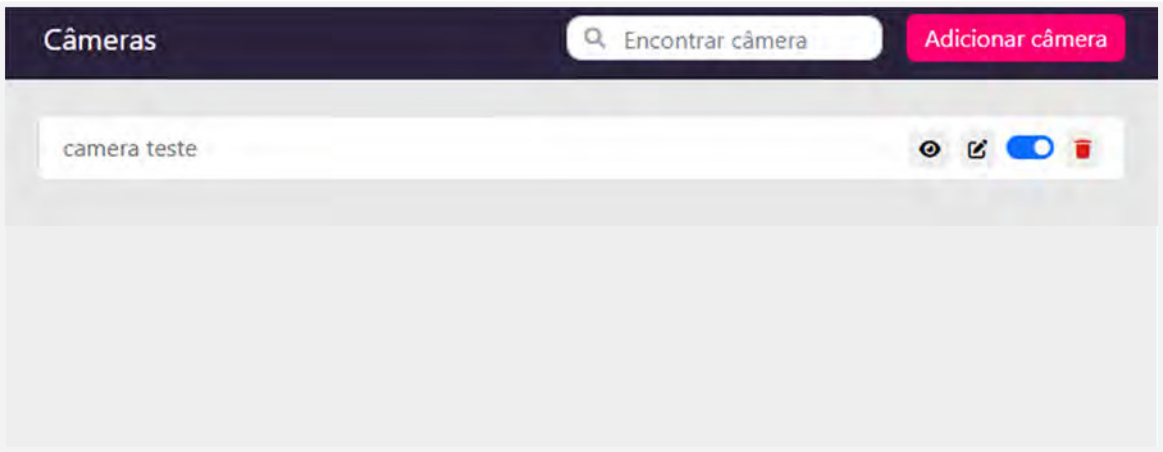


Figura 24 - Lista de Câmeras

Ao clicar no botão de adicionar câmera, é aberto um modal (Figura 25) para informar um nome para a câmera e um formulário para preencher com informações de conectividade. Essas informações de conectividade são: URL da câmera; se ela é privada ou não; coordenadas aproximadas (longitude e latitude); limitador de FPS (*Frames* por segundo).

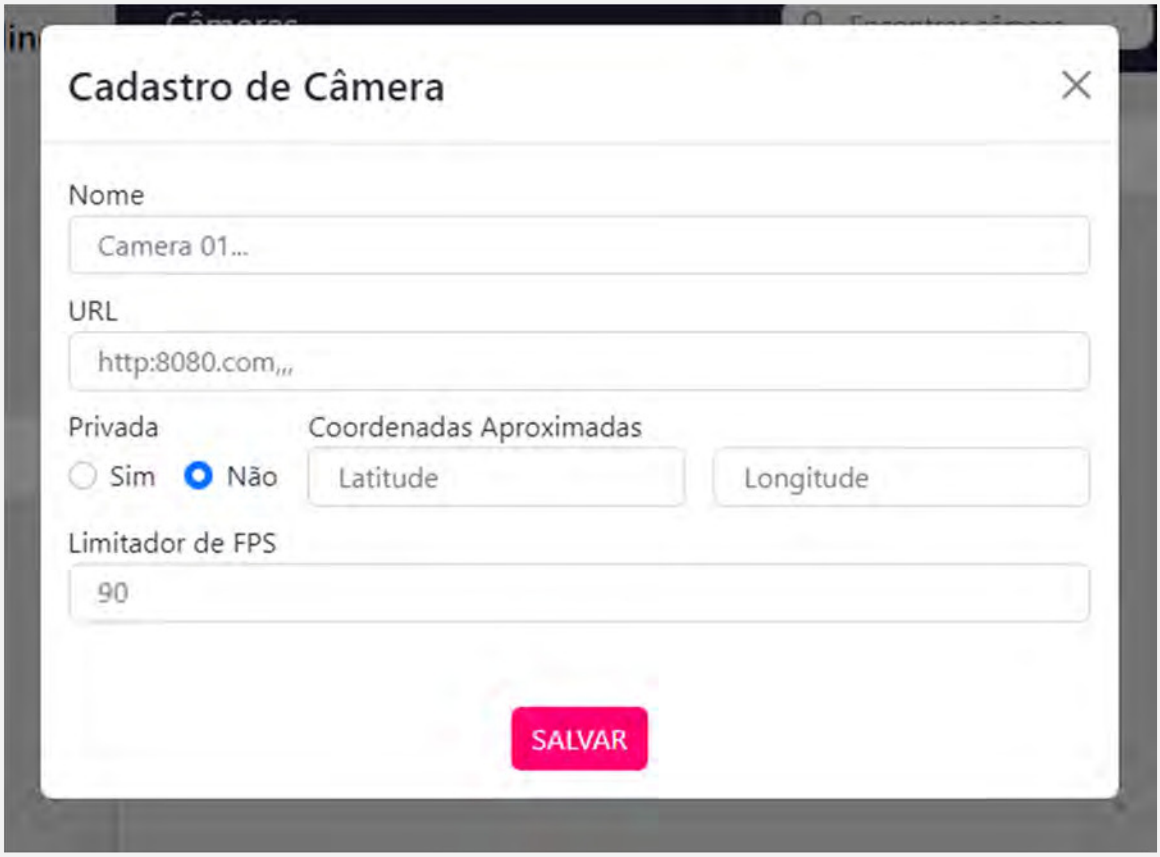


Figura 25 - Cadastro de Câmeras

Ao clicar na opção de visualizar câmera, será aberto um modal (Figura 26) com uma transmissão da câmera em tempo real.

Figura 26 - Visualização de Câmera

A edição de câmera é semelhante ao modal de adição de câmera, com a diferença nos dados da câmera, que já se encontram preenchidos e podem ser editados.

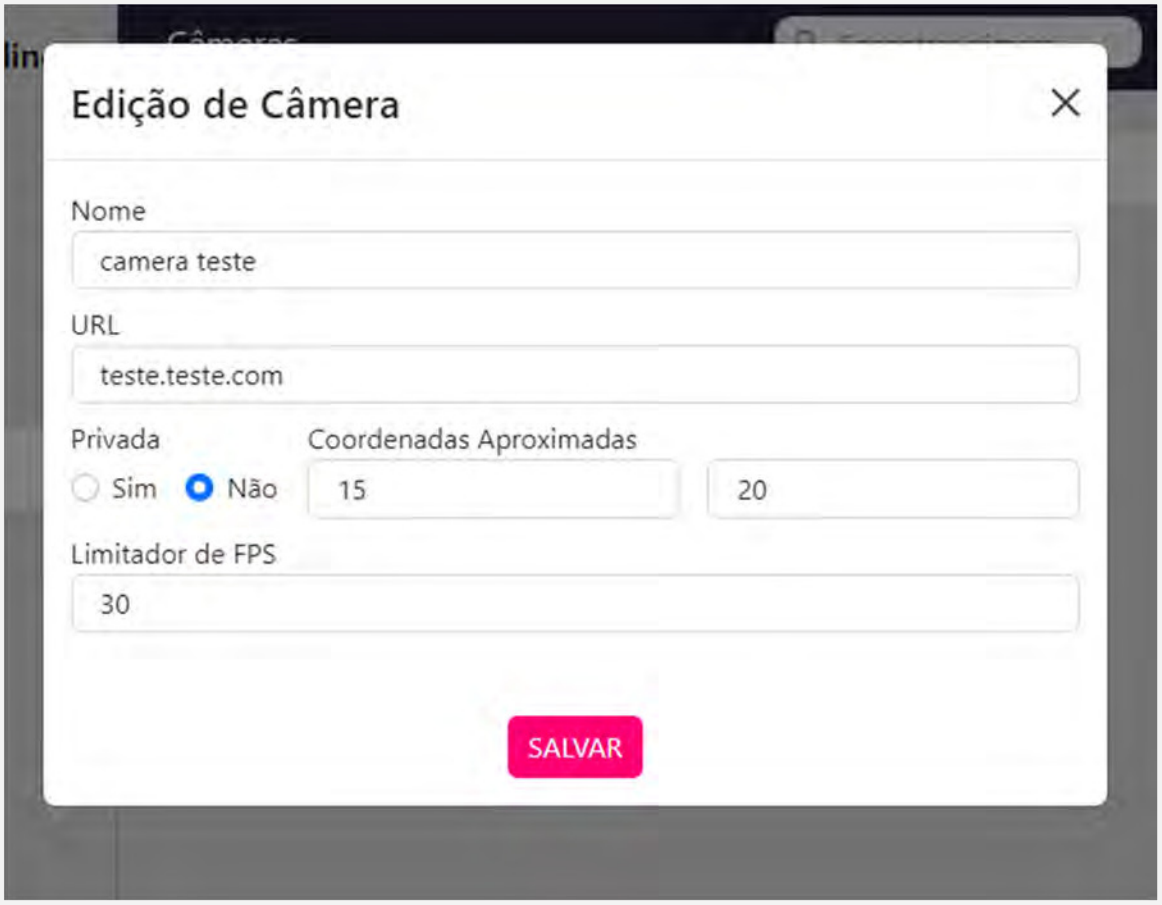


Figura 27 - Edição de Câmeras

3.5.7 PÁGINA DE SERVIÇOS

Nesta página são apresentados recursos de filtragem e de adição de serviços semelhantes à página de câmeras e *pipelines*, além de uma lista de serviços cadastrados contendo o nome do serviço e a opção de editar e excluir em cada serviço, como pode ser visto na Figura 28.

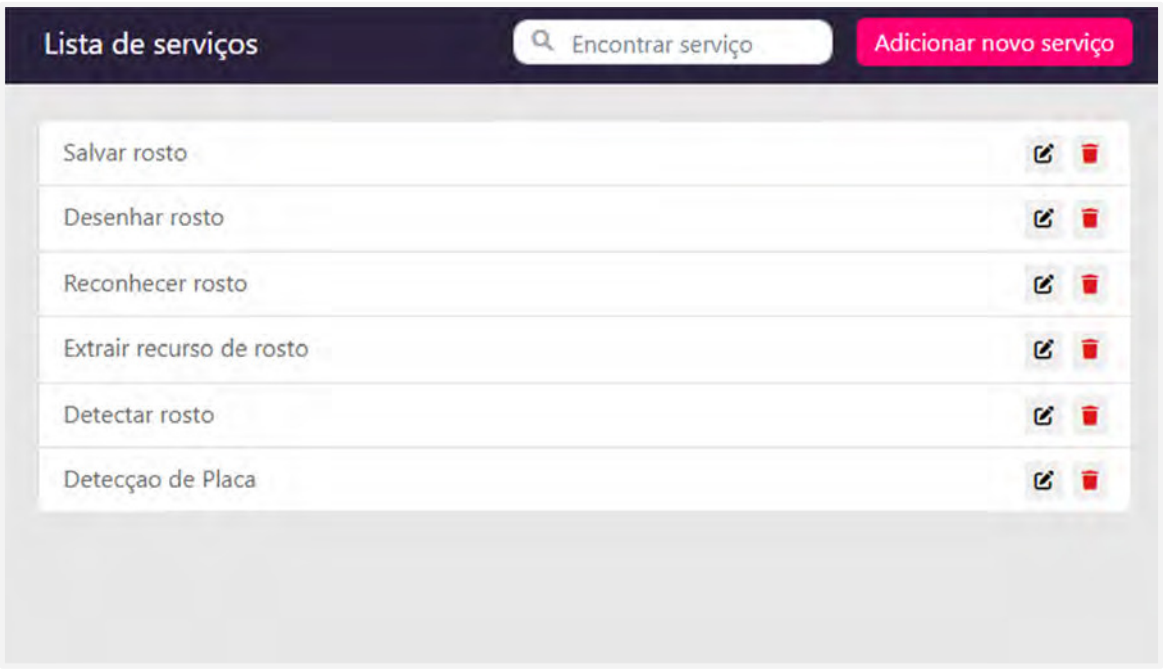


Figura 28 - Lista de Serviços

Ao clicar no botão de adicionar serviço, é aberto um modal (Figura 29) com os campos de nome, URL e descrição a serem preenchidos. Um botão de salvar e de adicionar parâmetro também são disponibilizados no modal.

Para os parâmetros adicionados no serviço, será necessário informar o nome do parâmetro, sua descrição, uma opção para marcar, caso o parâmetro seja obrigatório, e o tipo do parâmetro.

Os tipos de parâmetros podem ser:

- *STRING*: solicita um texto como parâmetro
- *NUMBER*: solicita um número como parâmetro
- *BOOLEAN*: solicita *true* ou *false* como parâmetro
- *FILE*: solicita um arquivo como parâmetro
- *SELECT*: solicita um uma opção da lista de opções definidas na criação do serviço como parâmetro
- *COLOR*: solicita uma cor como parâmetro

No parâmetro do tipo *select*, um campo de criação de opções é adicionado dentro do campo de criação do parâmetro como na Figura 30, para o usuário adicionar textos que servirão de opções para o *select*.

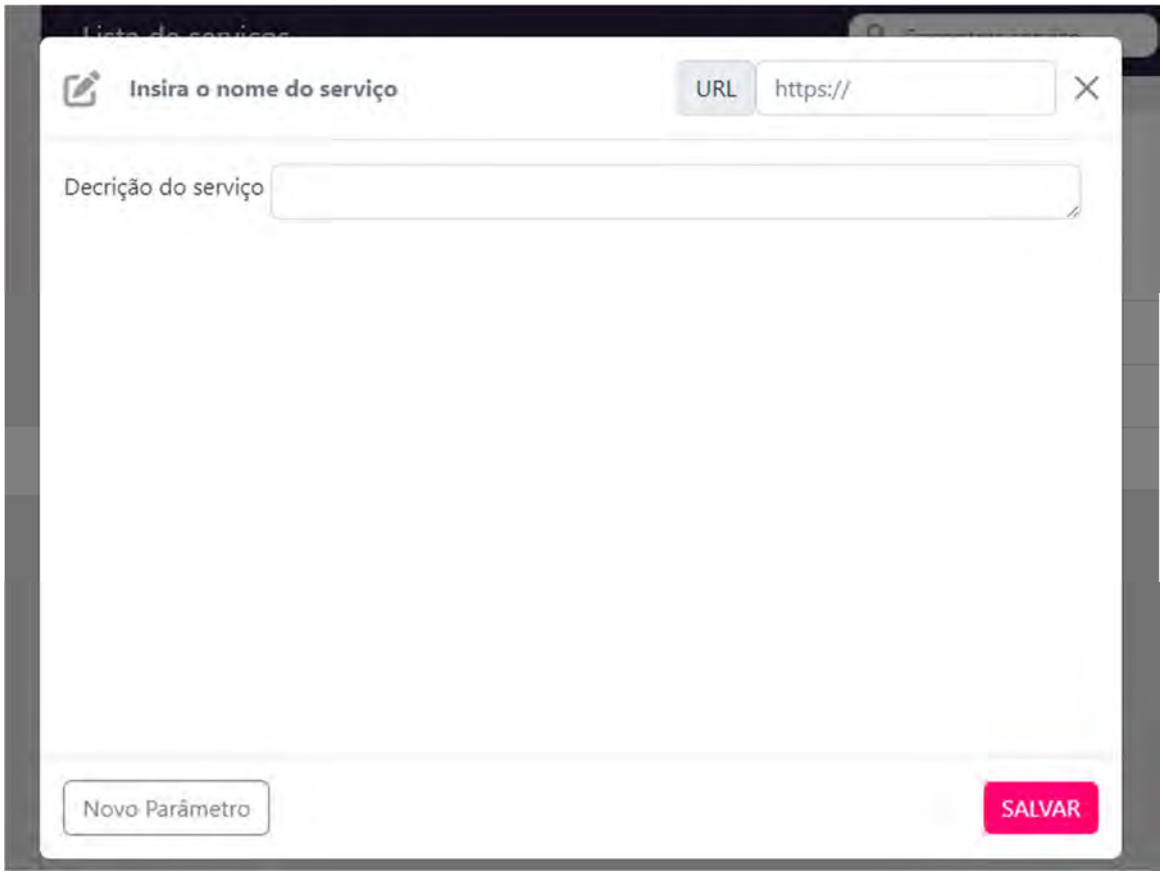


Figura 29 - Cadastro de Serviço sem parâmetros

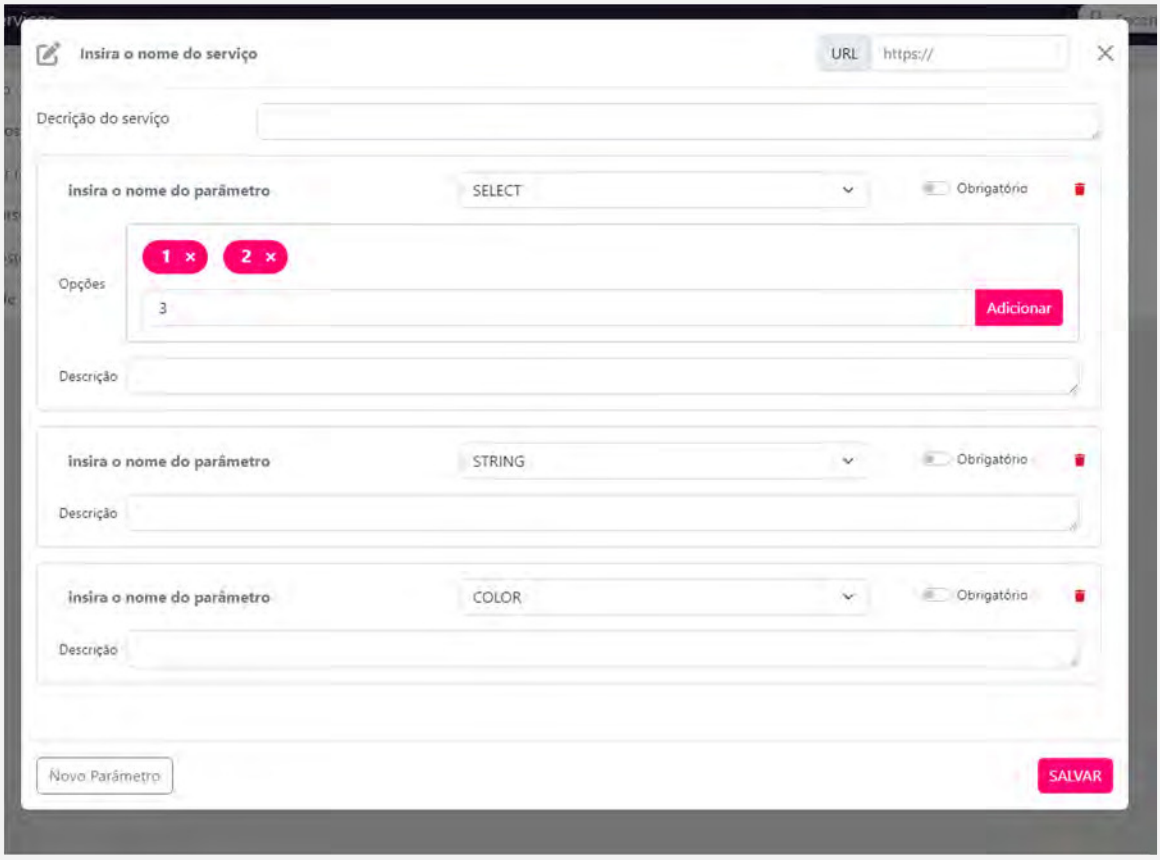


Figura 30 - Cadastro de Serviço com parâmetros

A edição de serviço segue a mesma regra de edição de câmera, um modal semelhante ao modal de adição (Figura 31) mas com os dados já preenchidos onde é possível editá-los.

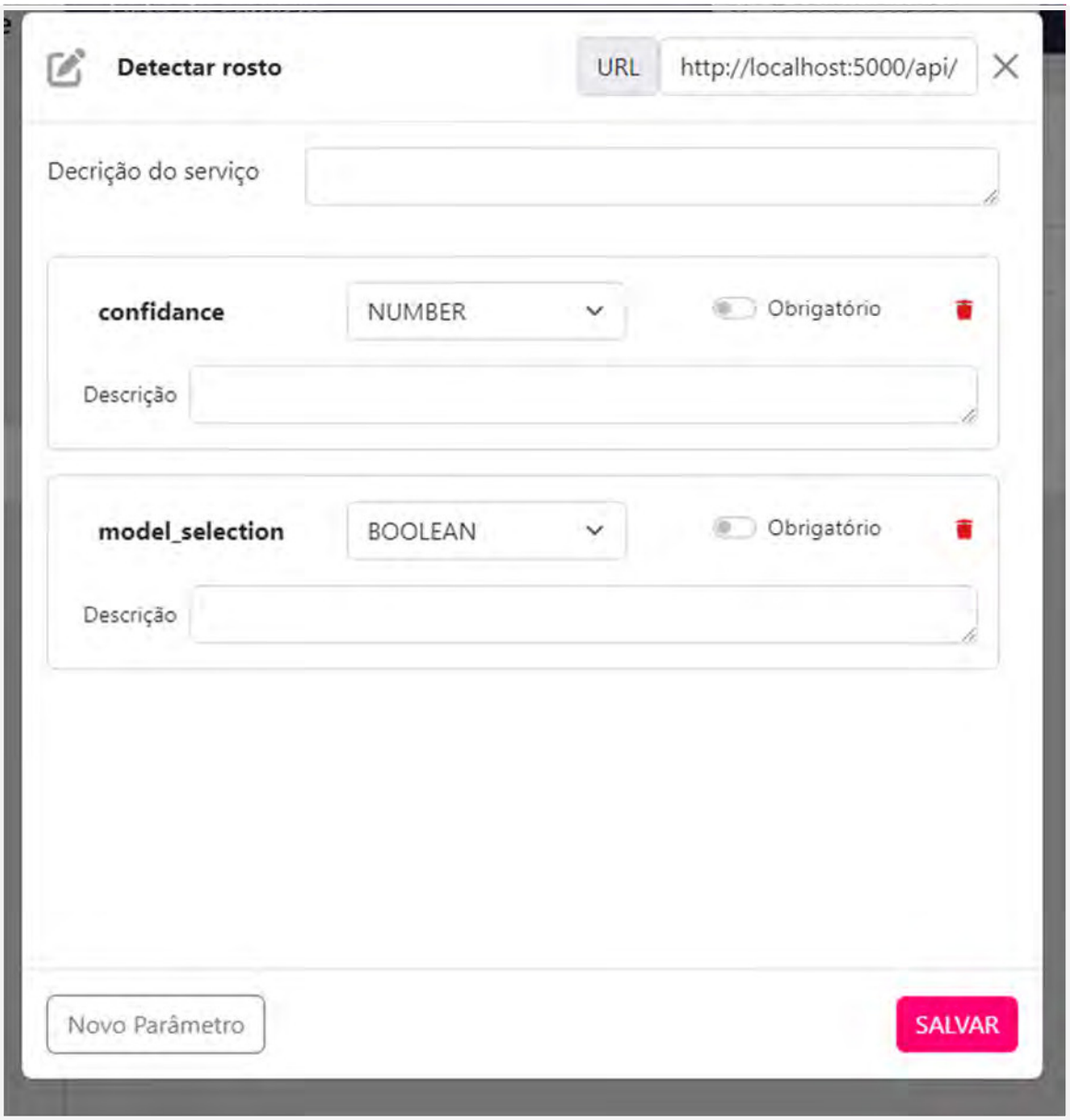


Figura 31 - Edição de Serviços

3.5.8 Página de Alterar senha

Por último, temos a página de alterar senha que, para acessá-la, é preciso estar autenticado na aplicação, caso contrário, só é possível alterar a senha pela página de *login*, clicando em recuperar senha, já apresentado na seção 3.5.1.

Para alterar a senha, o usuário deve informar a senha atual, a nova senha e por último, repetir a nova senha para confirmar que ele não errou a senha desejada e clicar em alterar. Os dados serão validados e a senha, alterada.

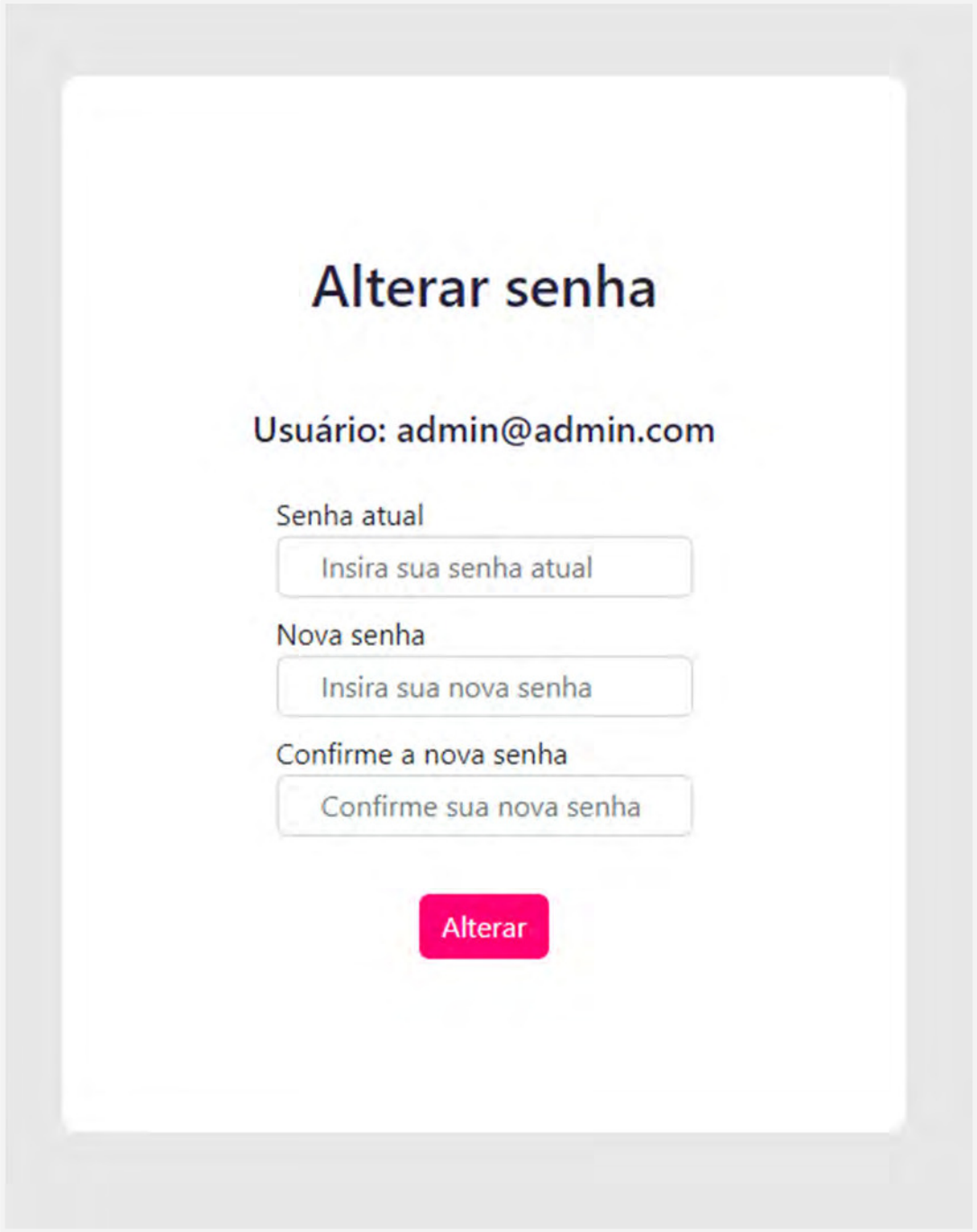


Figura 32 - Alterar Senha

3.6 PROJETO E EXECUÇÃO DE TESTES E VERIFICAÇÃO DE QUALIDADE DO PROTÓTIPO

Teste de *software* é o processo de execução de um produto para determinar se ele atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado. O seu objetivo

> é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento antes da entrega final.(NETO; CLAUDIO, 2007, p. 22)

Para validar o funcionamento dos requisitos do sistema e suas funcionalidades, foram realizados três tipos de testes: testes unitários, testes de integração e testes funcionais. Foram utilizadas algumas tecnologias específicas para o desenvolvimento geral dos testes, são elas: JUnit, Selenium e Cucumber.

3.6.1 TESTES UNITÁRIOS

De acordo com Pressman e Maxim (2016), o teste unitário é uma prática fundamental no desenvolvimento de *software*, onde cada módulo é testado de forma independente para garantir que sua lógica interna esteja correta e que as entradas e saídas esperadas sejam consistentes. Ele contribui para a detecção precoce de erros e para a construção de um sistema mais robusto.

Os testes unitários foram responsáveis por verificar o correto funcionamento das principais entidades isoladamente, garantindo a conformidade dos componentes individuais com suas especificações. Foram contempladas as classes: Camera, *Model*PDI, *Pipeline* e *User*. Ambas as classes de teste seguem um *model*o de teste unitário, cujo principal objetivo é validar o funcionamento de CRUD(*Create*, *Read*, *Update*, *Delete*) de cada classe, conforme as suas regras de negócio e características individuais como casos de erro esperados.

Os resultados dos testes unitários não são armazenados, entende-se que são testes de execução rápida que podem e, preferencialmente, devem ser executados periodicamente após alterações de código-fonte do sistema. Sendo assim, a visualização de resultado é feita através do painel de execução do JUnit dentro da IDE, conforme exemplificado na Figura 33 abaixo.

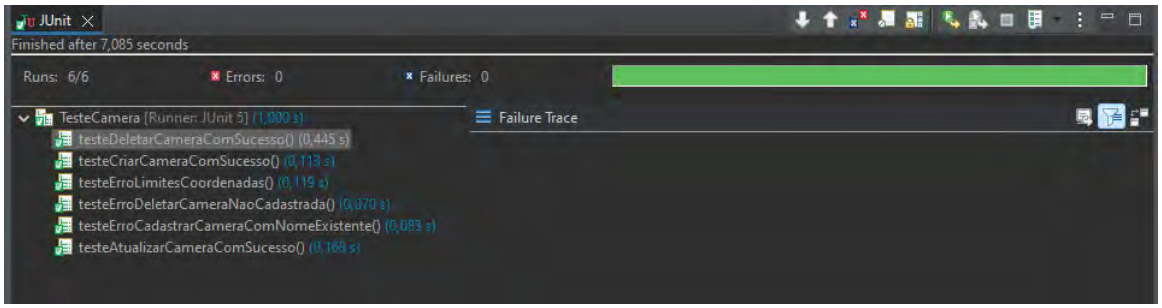


Figura 33 - Teste Unitário Camera

A Figura 33 mostra o resultado dos testes unitários referentes a classe Camera. Esta classe de teste unitário, semelhante as outras, apresenta 6 (seis) cenários de teste diferentes, 3 (três) cenários de “caminho feliz” onde a funcionalidade deve ser executada da forma esperada e com o resultado esperado, e 3 (três) cenários de “Caminho triste” onde o teste viola uma regra de negócio e apresenta um resultado esperado.

3.6.2 TESTES DE INTEGRAÇÃO

De acordo com Pressman e Maxim (2016) O objetivo principal do teste de integração é descobrir erros na *interface* e na comunicação entre os componentes, garantindo que o sistema funcione de forma integrada e sem problemas.

Os testes de integração foram realizados para verificar a interação adequada entre os diversos módulos e componentes do sistema, buscando identificar possíveis problemas de comunicação e integração.

Não foram desenvolvidas classes específicas para a execução dos testes de integração. Isso se deve ao fato de que esses testes foram executados de forma indireta nos testes unitários e testes funcionais do sistema, garantindo assim uma cobertura abrangente das funcionalidades, em um curto período de tempo, minimizando o escopo e custos de desenvolvimento.

Validações de testes de integração podem ser vistos embutidos em testes unitários ou funcionais como alguns exemplificados abaixo.

```java
@Test
public void testeSalvarUserComSucesso() {

    //Dado
    assertThrows(EntityNotFoundException.class, () -> {
        userService.getByEmail(user.getEmail());
    });

    //Quando
    User userReturn = userService.create(user);

    //Entao
    assertNotNull(userService.getByEmail(userReturn.getEmail()));
    assertEquals(userService.getByEmail(userReturn.getEmail()).getEmail(), user.getEmail());

}
```

Figura 34 - Teste unitário *User* (Teste de Integração)

A Figura 34 mostra um trecho de código dentro da classe de teste unitário "Test*User*", que dentre outros semelhantes, faz uma validação de persistência de um objeto *User* no banco de dados. Seguindo esta sequência lógica:

- Dado: Que o usuário não está no banco de dados.
- Quando: O usuário for cadastrado e inserido no banco de dados através do *userService*.
- Então: O usuário é persistido no banco de dados e agora pode ser recuperado através do *userService*.

```java
@Então("^usuario (.*)( não)? deve estar no banco$")
public void usuario_deve_estar_no_banco(String email, String condicao) {
    Optional<User> userRecuperado = userRepository.findByEmail(email);
    if (condicao == null) {
        assertTrue(userRecuperado.isPresent());
    } else {
        assertFalse(userRecuperado.isPresent());
    }

}
```

Figura 35 - Teste Funcional IntegracaoStep (Teste de Integração)

A Figura 35 mostra um trecho de código dentro da classe “Integracao*Step*” pertencente ao conjunto de classes de testes funcionais. Este método específico pode ser usado em qualquer *feature* pertencente a estrutura de testes funcionais, o mesmo valida se um usuário está ou não no banco de dados, e se este usuário deveria ou não está no banco de dados, adaptando-se a necessidade da *feature* que o utiliza.

Estes são exemplos dentre diversos outros testes que indiretamente validam a boa execução da comunicação e integração entre o *Backend* e o bando de dados.

3.6.3 TESTES FUNCIONAIS

De acordo com Pressman e Maxim (2016), os testes funcionais visam verificar se o sistema atende aos requisitos funcionais estabelecidos, assegurando assim a correta implementação das funcionalidades previstas.

Os testes funcionais tiveram como objetivo verificar se o sistema atendeu aos requisitos funcionais definidos, ou seja, se todas as funcionalidades previstas foram implementadas corretamente. Todos os testes funcionais foram realizados simulando a interação do usuário com a *interface Frontend*, através do Selenium, utilizando o *WebDrive* do navegador Google Chrome.

As validações foram realizadas utilizando JUnit 4, o Spring Boot foi integrado aos testes funcionais através da anotação @SpringBootTest para que as classes de *service* e a conexão com o banco de dados fosse possível durante a execução dos testes, e toda estrutura organizacional de classes foi projetada para a utilização do Cucumber.

Esta estrutura é separada majoritariamente em *Pages*, *Steps*, *Runners* e *features*, além destas, contém classes de configuração e arquivos HTML gerados automaticamente que apresentam um relatório da última execução de cada *Runner*. A Figura 36 abaixo mostra essa estrutura de divisão dentro do Eclipse IDE.

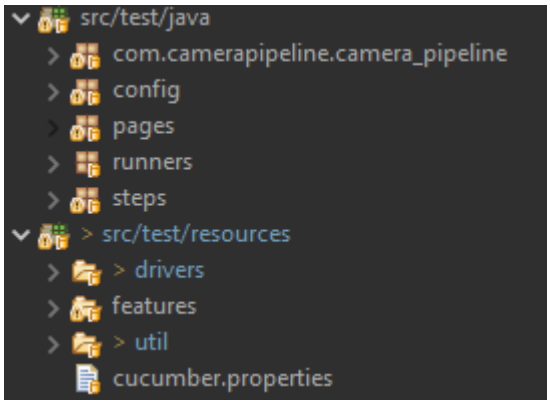


Figura 36 - Estrutura Organizacional Cucumber

3.6.3.1 PAGES

As classes de *Page* tem como objetivo concentrar o mapeamento HTML de uma página específica através do xpath e através disto possibilitar o acesso, interação e inserção de dados em elementos nesta página.

A Figura 37 mostra um trecho de código onde o sistema recupera respectivamente um elemento único que aponta para um botão na página e uma lista de elementos que apontam para as opções de um *select* na página.

```
45
46    @FindBy(xpath = "//*[@class=\"Tags_inputC__IWidP\"]/button")
47    private WebElement botaoAdicionarOpcaoSelectType;
48
49    @FindBy(xpath = "//*[@class=\"Tags_tagItem__WQEh+\"]")
50    private List<WebElement> opcoesSelectType;
```

Figura 37 - Mapeamento da página de serviços (Trecho)

Estes elementos são recuperados como um *WebElement* (classe pertencente ao Selenium) que dispõe de inúmeros métodos para a manipulação destes objetos, como visto na Figura 38 abaixo.

```
189    public void inserirOpcaoSelectType(String value) {
190        campoOpcaoSelectType.clear();
191        campoOpcaoSelectType.sendKeys(value);
192        botaoAdicionarOpcaoSelectType.click();
193    }
194
195    public WebElement recuperarOpcaoSelectTypeAdicionadaPorNome(String value) {
196        for(WebElement e : opcoesSelectType) {
197            String nome = e.findElement(By.xpath("span[1]")).getText();
198            if(nome.trim().equals(value.toUpperCase().trim())) {
199                return e;
200            }
201        }
202        System.err.println("Opcao " + value + " Não encontrado");
203        return null;
204    }
205
```

Figura 38 - Métodos ServicosPage

Estes métodos serão utilizados posteriormente por uma classe de *Step* ou pela própria classe *Page* em um método de uso interno.

3.6.3.2 STEPS

As classes de *Step* tem como objetivo concentrar métodos que fazem uma mediação entre uma *feature* e uma *Page*, são responsáveis por conter os códigos de validação com JUnit, receber as informações de entrada da *feature*, tratá-las se necessário e encaminhá-las para a página a qual a *Step* foi destinada.

```
15    @Dado("^que informei email (.*) e senha (.*)$")
16    public void que_informei_email_e_senha(String email, String senha) {
17        Na(LoginPage.class).inserirEmailLogin(email);
18        Na(LoginPage.class).inserirSenhaLogin(senha);
19    }
20
21    @Quando("^tentar logar$")
22    public void tentar_logar() {
23        Na(LoginPage.class).clicarBotaoEntrar();
24    }
```

Figura 39 - Métodos LoginStep

A Figura 39 mostra métodos da *Step* destinada à página de *login* do sistema. As anotações @Dado e @Quando são usadas pela *feature* para chamar um método específico de qualquer *Step* identificado através da expressão regular.

```
if (!mensagemErro.isEmpty()) {
    do {
        esperar(1);
        int tentativa = 0;
        while(tentativa <2) {
            try {
                esperado = driver.findElements(By.id("toastMsg"));
                for(WebElement e : esperado) {
                    if(!e.getAttribute("innerText").equals("Não foi possivel renderizar a pré-visualização")) {
                        erroRecuperado = e.getAttribute("innerText");
                        return;
                    }else {
                        WebElement botaoFechar = e.findElement(By.xpath("../../../button"));
                        botaoFechar.click();
                    }
                }
                break;
            } catch ( org.openqa.selenium.StaleElementReferenceException e) {
                tentativa++;
            }
        }

    } while (erroRecuperado.equals("Processando")
            ||erroRecuperado.equals("Deletando")
            ||erroRecuperado.equals("Salvando")
            ||erroRecuperado.equals("Salvando!")
            ||erroRecuperado.equals("Acorreu um erro ao gerar um resultado")
            );

    try {
        WebElement botaoFechar = driver.findElement(By.xpath("//*[@class=\"Toastify__close-button Toastify__close-button--light\""
                + " or @class=\"Toastify__close-button Toastify__close-button--colored\"]"));
        botaoFechar.click();
    } catch (Exception e) {
        System.out.println(e.getMessage());
        System.err.println("Probelamas na NotificaçãoStep");
    }
    esperar(1);
    assertEquals(erroRecuperado, mensagemErro);
```

Figura 40 - Trecho de método em NotificacaoStep

A Figura 40 mostra um trecho de código dentro do método “sistema_notifica” pertencente a classe de *Step* “NotificacaoStep”. Este *Step* é responsável por recuperar *toasts* de notificação em qualquer página do sistema quando solicitado e validar seu conteúdo, prevendo possíveis exceções e conflitos de *timing* de múltiplos *toasts*.

3.6.3.3 RUNNERS

O *Runner* é essencialmente uma classe vazia que funciona como inicializador através da anotação @RunWith. Algumas configurações necessárias e opcionais são feitas através da anotação @CucumberOptions.

Um *Runner* pode ser direcionado para um grupo específico de *features* através de *tags*, executando apenas as *features* que contém a *tag* declarada.

Também é possível, a partir da anotação @CucumberOptions do *Runner*, inserir um *plugin* para que no final da execução seja criado um relatório contendo o resultado de cada *feature* executada por este *Runner*.

```
@RunWith(Cucumber.class)
@CucumberOptions(
        features = "src/test/resources/features",
        glue = {"steps", "config"},
        tags = "@Geral",
        plugin = {"pretty", "html:target/principal/index.html", "json:target/report.json"},
        monochrome = true,
        snippets = SnippetType.CAMELCASE,
        dryRun = false
        )
public class RunnerPrincipal {


}

```

Figura 41 - RunnerPrincipal

A Figura 41 mostra a classe *Runner*Principal. Este *Runner* utiliza a *tag* @Geral, que está presente em todas as *features* de teste criadas, sendo assim, todas as *features* são executadas por este *Runner*. Outros *Runners* também foram criados para que *features* específicas sejam executadas separadamente quando necessário, tornando o processo mais eficiente.

3.6.3.4 FEATURES

A *Feature* é um arquivo com extensão "*.feature*" e utiliza a linguagem de especificação Gherkin para descrever cenários sequenciais em uma escrita textual simples. Uma *feature* necessita de duas linhas de configuração inicial: "#language: pt" e "enconding:UTF-8". A partir deste ponto pode-se desenvolver no idioma pt-BR.

Foi designada uma única funcionalidade para cada *feature*. Dentro desta funcionalidade diversos cenários podem ser criados. Acima da declaração de funcionalidade estão as anotações de *tags,* que identificam esta *feature*. Não existe limite para o número de *tags* de um *feature*.

Um Cenário pode ser implementado através das palavras-chave "Dado", "Quando", "Entao", "E" e "*", entre outras, com a mesma função, seguido de um

conjunto de caracteres, que identificarão que método de *Step* será executado. Este método extrai os dados esperados através da expressão regular aplicada ao conjunto de caracteres que o identifica.

```
# language: pt
#enconding: UTF-8

@PaginaGuia @Geral
Funcionalidade: Deve validar usabilidade da tela Guia
  Como um Usuario
  Eu quero interagir com aplicação
  Para acessar conteudo da pagina de Guia

Cenário: Usuario alterna entre as abas sem estar logado
  * clicar no botão Guia
  Dado que cliquei na aba <aba>
  Então <condicao>vejo o conteudo <conteudo>

  Exemplos:
  |condicao|      aba        |     conteudo        |
  |        |visão_geral      |visão_geral          |
  |        |pipelines        |pipelines            |
  |        |fluxoDaPipeline  |fluxoDaPipeline      |
  |        |aplicarProcessos|aplicarProcessos      |
  | "não " |visão_geral      |aplicarProcessos      |
  | "não " |pipelines        |fluxoDaPipeline       |
  | "não " |fluxoDaPipeline  |pipelines             |
  | "não " |aplicarProcessos|visão_geral            |
```

Figura 42 - *Feature* Guia

A Figura 42 mostra um trecho da implementação da *feature* Guia, responsável por um processo simples de validação do funcionamento das abas da tela Guia. O Cenário presente na figura usa uma tabela de exemplos onde cada linha é executada sequencialmente e o resultado é transcrito de forma individual no relatório final. Esta estratégia é uma forma de evitar a duplicação de código de um mesmo cenário com valores diferentes e é utilizada em diversos outros cenários.

3.6.3.5 RELATÓRIOS

Os Relatórios de teste são formados automaticamente pelo cucumber e podem ser solicitados em diferentes formatos de arquivos. Para o projeto Imagem Pipeline foram definidos dois tipos de arquivos para os relatórios de execução de teste: JSON e HTML.

Após a finalização de execução de um *runner*, os arquivos são gerados no diretório especificado nas configurações do *runner* executado, definindo também o nome de identificação do arquivo.

O arquivo em formato HTML gerado tem uma estilização padrão própria, embora possa ser modificada, e pode ser aberto por qualquer navegador. Essa estrutura pode ser observada na Figura 43.

Figura 43 - *Home* Relatório HTML

Como visto na Figura 43, o Relatório apresenta algumas informações sobre a execução do teste como: a duração da execução, a versão do Java e do cucumber, o sistema operacional da máquina onde o teste foi executado e quantidade de cenários. Também é possível usar um campo de pesquisa para encontrar palavras ou cenários específicos dentro do documento.

A tela inicial apresenta uma lista de elementos *dropdown*. Cada elemento representa uma *feature* executada, mostrando o caminho até o diretório de cada uma. Ao clicar sobre o elemento de uma *feature*, será apresentado abaixo dela a descrição da sua funcionalidade, Cenários de teste e os passos executados em cada cenário como pode ser visto na Figura 44.

Figura 44 - *Feature* Serviços - Relatório HTML

Ao lado esquerdo de cada passo do cenário existe um símbolo que varia entre três estados: “Aprovado”, “Falhou” e “Não executado”. Quando a etapa é efetuada com sucesso e o teste é aprovado(passou), o símbolo assume a cor

verde, porém, se houver um erro na execução do passo ou o teste falhar(não passou), o símbolo assume a cor vermelha e os próximos passos do cenário um símbolo de cor azul, mostrando que a rotina do teste foi interrompida, como exemplificado na Figura 45.

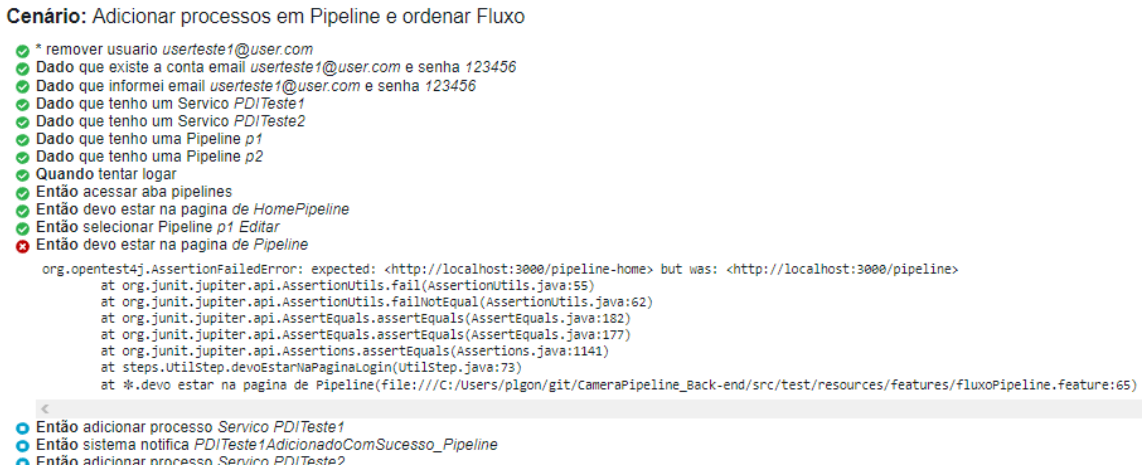


Figura 45 - *Feature* FluxoPipeline - Relatório HTML

O uso destes relatórios é uma forma eficiente de documentar a execução dos testes. A linguagem Gherkin permite uma fácil interpretação da tarefa de cada passo e o objetivo do cenário. Desta forma, mesmo uma pessoa sem um conhecimento técnico, consegue abstrair as informações e compreender o resultado apresentado no relatório.

O relatório em arquivo .HTML pode ser acessado através do arquivo 1.[1]

3.6.4 Conclusões Finais Sobre os Testes

Os testes realizados foram essenciais para validar o funcionamento do sistema e garantir a conformidade com os requisitos estabelecidos. Os testes unitários permitiram verificar individualmente o correto funcionamento das principais entidades do sistema, enquanto os testes de integração garantiram a interação adequada entre os diferentes componentes. Os testes funcionais, por

[1] Arquivo 1 – Relatório Geral de Testes Funcionais - Disponível em: https://drive.google.com/file/d/1LlHdnCxZAEwbYD0nj4ydV_1j2qwoTTzb/view?usp=sharing

sua vez, avaliaram se todas as funcionalidades foram implementadas corretamente, simulando a interação do usuário com o sistema.

A utilização de tecnologias como JUnit, Selenium e Cucumber proporcionou uma abordagem abrangente para os testes, facilitando a execução e a visualização dos resultados. Os relatórios gerados pelo Cucumber foram especialmente úteis na documentação dos testes, fornecendo informações detalhadas sobre a sua execução.

Com base nos resultados obtidos, foi possível identificar possíveis problemas de comunicação, integração e falhas nas funcionalidades. Essas informações foram valiosas para orientar a continuidade do desenvolvimento do sistema, com o objetivo de realizar melhorias e correções necessárias.

Em resumo, os testes realizados desempenharam um papel fundamental na garantia da qualidade do sistema, fornecendo uma base sólida para a conclusão do projeto. Através desses testes, foi possível validar as funcionalidades, identificar problemas e obter *insights* para aprimorar a aplicação. Com isso, o trabalho realizado contribui para a confiabilidade e eficiência do sistema, atendendo às expectativas dos usuários e das partes interessadas.

4 CONSIDERAÇÕES FINAIS

A plataforma Imagem Pipeline ajudará o cliente do projeto, assim como usuários que desenvolvem serviços de processamento de imagem, no desenvolvimento do mesmo, uma vez que o desenvolvedor não precisará gastar tempo implementando uma *interface* para testar, aplicar ou visualizar seus serviços como ocorre atualmente.

4.1 REVISÃO DOS OBJETIVOS

Com a finalização da implementação do Imagem Pipeline, a equipe conseguiu realizar todos os objetivos propostos pelo cliente, finalizando assim, um sistema capaz de fornecer aos usuários uma *interface* com processamento de mídia, gerenciamento de APIs, gerenciamento de *pipelines* e gerenciamento de câmeras. Dessa forma, tanto o cliente quanto usuários desenvolvedores de APIs de processamento de imagem conseguirão utilizar a *interface* para visualizar,

configurar e manipular suas APIs na prática, poupando-os de tempo e dinheiro que seriam gastos desenvolvendo uma *interface* para suas APIs.

4.2 TRABALHOS FUTUROS

Atualmente o sistema não está disponível em um servidor, essa responsabilidade fica a cargo do cliente, que ainda está finalizando as APIs de processamento de imagem em si, que serão exclusivas do sistema.

Como trabalho futuro serão integradas APIs finalizadas ao sistema para que esses serviços fiquem disponíveis para todos os usuários da plataforma utilizarem, caso desejem. Junto com essa integração, serão disponibilizados dados e estatísticas a respeito dessas APIs reservadas do sistema na página de *dashboard* para que o usuário possa visualizar essas informações.

Caso a demanda de usuários por esses serviços exclusivos seja muito alta, o cliente deseja reformular a plataforma do Imagem Pipeline para adicionar um serviço de assinatura, limitando o acesso a esses serviços exclusivos.

REFERÊNCIAS

Astah. **Premier Diagramming, Modeling Software & Tools | Astah**. Disponível em: https://astah.net/. Acesso em: 11 abr. 2023.

Axios. Official Website. Disponível em: https://axios-http.com/ptbr/docs/intro/. Acesso em 16 maio 2023.

BOBOYOROVA C. **An Introduction to No-Code/Low-Code Platforms: Seven Things You Should Know**. Revista Forbes, 7 mar. 2023. Disponível em: https://www.forbes.com/sites/forbestechcouncil/2023/03/07/an-introduction-to-no-codelow-code-platforms-seven-things-you-should-know/?sh=3c6d04805fe2.

CHACON S; STRAUB B. J. **Pro Git**. 2. ed. Berkely: Apress, 2014.

Coodesh. **O que é Cucumber?** Disponível em: https://coodesh.com/blog/dicionario/o-que-e-cucumber/. Acesso em: 08 maio 2023.

Creately. **Guia de Tipos de Diagramas UML: Aprenda sobre todos os tipos de diagramas UML com exemplos**. Disponível em: https://creately.com/blog/pt/diagrama/guia-de-tipos-de-diagramas-uml-aprenda-sobre-todos-os-tipos-de-diagramas-uml-com-exemplos/#ActivityDiagram. Acesso em: 10 maio 2023.

_____. **Os diagramas de classes são o principal componente de qualquer solução orientada a objetos**. Disponível em: https://creately.com/blog/pt/diagrama/guia-de-tipos-de-diagramas-uml-aprenda-sobre-todos-os-tipos-de-diagramas-uml-com-exemplos/#ClassDiagram. Acesso em: 10 maio 2023.

Cucumber. **Gherkin**. Disponível em: https://cucumber.io/docs/guides/overview/. Acesso em: 08 maio 2023.

Devmedia. **Modelagem de Sistemas Através de UML - Uma visão geral**. Disponível em: https://www.devmedia.com.br/modelagem-de-sistemas-atraves-de-uml-uma-visao-geral/27913. Acesso em: 08 maio 2023.

_____. **Testes de Integração com Java e JUnit**. Disponível em: https://www.devmedia.com.br/testes-de-integracao-com-java-e-junit/25662. Acesso em: 08 maio 2023.

FIELDING, R. **Architectural Styles and the Design of Network-based Software Architectures**. Dissertação (Doutorado Ciência da Informação e da Computação)

– Information and Computer Science – University of California Irvine, Califórnia, 2000.

FOWLER, M. **UML Essencial: Um Breve Guia para a Linguagem-Padrão de Modelagem de Objetos**. 3. ed. Porto Alegre: Bookman, 2004.

GONZALEZ, R.; WOODS, R. **Processamento de Imagens Digitais**. São Paulo: Blucher, 2000. ISBN 9788521202646.

GRINBERG, M. **Flask Web Development: Developing Web Applications with Python**. Sebastopol, CA: O'Reilly Media, 2014. ISBN: 978-1-449-37262-0.

IBM. **Na UML, os diagramas de componentes mostram a estrutura do sistema de software**. Disponível em: https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=services-component-diagrams. Acesso em: 10 maio 2023.

Intel. **Solution Brief: Viso Suite: The No-Code Computer Vision Platform**. Disponível em: https://viso.ai/wp-content/uploads/2021/09/Solution-Brief-Viso-Suite-The-No-Code-Computer-Vision-Platform.pdf. Acesso em: 30 jun. 2023.

JOHNSON, R. et al. **Professional Java Development with the Spring Framework**. Indianapolis, IN: Wiley Publishing, Inc. 2005. ISBN 978-0-7645-7483-2

LUTZ, M. **Learning Python**. 5. ed. Sebastopol, CA: O'Reilly Media, 2013. ISBN: 9781449355692.

NETO, Arilo; CLAUDIO, Dias. **Introdução a teste de software**. Engenharia de Software Magazine, v. 1, p. 22, 2007.

NPM. Official Website. Disponível em: https://docs.npmjs.com/about-npm. Acesso em 16 maio 2023.

Oracle. **Software Java**. Disponível em: https://www.oracle.com/br/java/. Acesso em: 30 jun. 2023.

OpenCV. **Home - OpenCV**. Disponível em: https://opencv.org/. Acesso em: 30 jun. 2023.

POWERS, S. **Learning Node: Moving to the Server-Side**. Sebastopol, CA: O'Reilly Media, 2012. ISBN 978-1-449-32307-3.

PRESSMAN, R. S; BRUCE R. M. **Engenharia de Software, Uma Abordagem Profissional**. 8. ed. Porto Alegre: AMGH, 2016. ISBN 978-85-8055- 533-2.

RAMAMOORTHY, C. **Pipeline Architecture**. Computing Surveys, Vol.9, No.1, 1977.

React. Official Website. Disponível em: https://react.dev/. Acesso em 16 maio 2023.

REACT BOOTSTRAP. Official Website. Disponível em: https://react-bootstrap.github.io/. Acesso em 16 maio 2023.

React Flow. Official Website. Disponível em: https://reactflow.dev/. Acesso em 16 maio 2023.

Selenium. **Selenium documentation**. Disponível em: https://www.selenium.dev/pt-br/documentation/. Acesso em: 10 maio 2023.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011. ISBN 978-85-7936-108-1.

TypeScript. **TypeScript is JavaScript with syntax for types**. Disponível em: https://www.typescriptlang.org/. Acesso em: 11 abr. 2023.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Monteiro
Pb-264, S/N, Serrote, CEP 58500-000, Monteiro (PB)
CNPJ: 10.783.898/0008-41 - Telefone: (83) 3351-3700

Documento Digitalizado Ostensivo (Público)

TCC Imagem Pipeline Revisado

Assunto: TCC Imagem Pipeline Revisado
Assinado por: Jordielson Silva
Tipo do Documento: Relatório
Situação: Finalizado
Nível de Acesso: Ostensivo (Público)
Tipo do Conferência: Cópia Simples

Documento assinado eletronicamente por:

- **Jordielson Emanuel Caldeira da Silva**, **ALUNO (201925020014) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - MONTEIRO**, em 10/11/2023 09:56:02.

Este documento foi armazenado no SUAP em 10/11/2023. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 991275
Código de Autenticação: a08510f856