UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

Gustavo Glasgio Bravin Andrade

DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA DE USUÁRIO PARA UM ESPAÇO INTELIGENTE DA UFES

Vitória-ES

Outubro/2021

Gustavo Glasgio Bravin Andrade

DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA DE USUÁRIO PARA UM ESPAÇO INTELIGENTE DA UFES

Parte manuscrita do Projeto de Graduação do aluno Gustavo Glasgio Bravin Andrade, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Vitória-ES

Outubro/2021

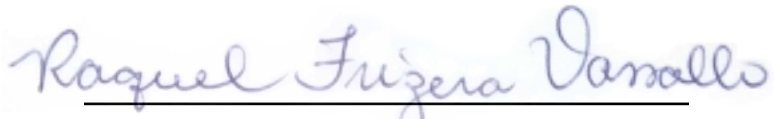
Gustavo Glasgio Bravin Andrade

DESENVOLVIMENTO DE UMA INTERFACE GRÁFICA DE USUÁRIO PARA UM ESPAÇO INTELIGENTE DA UFES

Parte manuscrita do Projeto de Graduação do aluno Gustavo Glasgio Bravin Andrade, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovado em 07 de Outubro de 2021.

COMISSÃO EXAMINADORA:



Profa Dra. Raquel Frizera Vassallo
Universidade Federal do Espírito Santo
Orientador

Profa. Dra. Mariana Rampinelli Fernandes
Instituto Federal do Espírito Santo
Examinador

MSc. Felippe Mendonça de Queiroz
idwall
Examinador

Vitória-ES

Outubro/2021

RESUMO

A crescente complexidade de programas modernos implica em uma curva de aprendizagem de utilização grande por parte dos usuários. O desenvolvimento de interfaces gráficas de usuário é uma peça chave para facilitar a utilização de programas complexos por pessoas que não necessariamente entendem de programação. Interfaces gráficas de usuário possuem papel fundamental em atenuar curvas de aprendizagem de utilização e aumentar a perícia de usuários no manejo de programas. Espaços inteligentes ganharam atenção significativa nos últimos anos devido aos avanços na tecnologia e facilidade de implantação, espera-se dos espaços inteligentes grandes avanços na interação entre pessoas e máquinas. Tendo um dos principais objetivos a interação com pessoas, os espaços inteligentes são naturalmente um sistema que está em um relacionamento constante com usuários, portanto uma boa interface de usuário é indispensável. Neste trabalho foi desenvolvido uma interface gráfica de usuário para o sistema de espaço inteligente do laboratório de Visão Computacional e Robótica localizado na Universidade Federal do Espírito Santo, de forma a permitir a usuários leigos supervisionar tal sistema de forma fluida e sistemática, colaborando para o bom funcionamento e a disseminação do sistema. O desenvolvimento da interface gráfica produzida e a sua implementação se deu de forma fluida ao se apoiar em outros trabalhos e comunidades de desenvolvimento. Os resultados se mostraram satisfatórios quando comparados a trabalhos existentes, principalmente no que se diz respeito a personalização para casos específicos do espaço inteligente estudado, como automatização, recursos únicos, aplicações únicas e aparência.

Palavras-chave: Interface gráfica de usuário. Espaço inteligente.

ABSTRACT

The complexity of modern programs implies a steep learning curve for users to use it. The development of graphical user interfaces is a key part in facilitating the use of complex programs by people who do not necessarily are programmers. Graphical user interfaces play a fundamental role in smoothing learning curves and increasing users' expertise. Intelligent spaces have gained attention in recent years due to advances in technology and ease of deployment, great advances in the interaction between people and machines are expected. Having as one of its main goals being interact with people, intelligent spaces are naturally a system that is in constant relationship with users, so a good user interface is indispensable. In this work, a graphical user interface was developed for the intelligent space system of the Computer Vision and Robotics laboratory (Lab VISIO - laboratório de Visão Computacional e Robótica) located at the Federal University of Espírito Santo (Ufes), in order to allow lay users to supervise such system in a fluid and systematic way, contributing to the good operation and dissemination of the system. The development of the graphical interface and its implementation for it took place in a fluid way, relying on other works and development communities. The results are satisfactory when compared to existing works, especially with regard to customization for specific cases of the studied intelligent space, such as automation, unique features, unique applications and appearance.

Keywords: Graphical user interface. Intelligent space.

LISTA DE FIGURAS

Figura 1 – Arquitetura genérica de espaços inteligentes. 14
Figura 2 – Exemplo de interface entre programas. 16
Figura 3 – Exemplo de comunicação utilizando API REST. 16
Figura 4 – Esquema das camadas da interface gráfica. 18
Figura 5 – Exemplo de GUI para programação em blocos. 19
Figura 6 – Exemplo de terminal ao observar alguns aspectos do IS. 21
Figura 7 – Esquema básica do trabalho. 22
Figura 8 – Esquema final do trabalho. 22
Figura 9 – Captura de tela da *dashboard* padrão do Kubernetes. 28
Figura 10 – Exemplificação do sistema de marcação (exemplo). 30
Figura 11 – Tela de autenticação. 31
Figura 12 – Tela inicial em seus estados de menu lateral retraído (a) e expandido (b) (inicial). 32
Figura 13 – Tela inicial em sua versão *mobile* com o menu lateral retraído (a) e expandido (b). 33
Figura 14 – Cabeçalho da UI (a) e as ações extras de trocar de idioma (b) e ver demais opções (c) (cabeçalho). 34
Figura 15 – Exemplo de uma tela em português (a) e inglês (b). 35
Figura 16 – Menu lateral em suas formas retraída (a) e expandida (b) (menu-lateral). 36
Figura 17 – Tela de painel de controle (dashboard). 37
Figura 18 – Tela de visualização de câmeras (vis-cam). 38
Figura 19 – As várias formas que a UI pode mostrar as imagens das câmeras. Da Esquerda para direita, apenas uma câmera (a), quatro câmeras (b) e quatro câmeras com destaque para uma (c). 38
Figura 20 – Tela de lista de *pods*. 39
Figura 21 – Tela de lista de serviços (lista-serviço). 40
Figura 22 – Detalhamento de um item expandido da lista de serviço (lista-item). . 40
Figura 23 – Formulários de criação (a) e edição (b) de cadastro de serviços. 41
Figura 24 – Visualização direta de um serviço. 42
Figura 25 – Visualização dos *logs* de um pod. 43
Figura 26 – Exemplo de uma tela para usuários com diferentes papéis. Em (a) um usuário administrador e em (b) um usuário comum. 44

LISTA DE ABREVIATURAS E SIGLAS

API	*Application ProgrammingInterface*
CT-II	Centro Tecnológico II
GUI	*Grapfical User Interface*
IPC	*Inter-Process Communication*
IS	*Intelligent Space*
Lab VISIO	Laboratório de Visão Computacional e Robótica
REST	*Representational State Transfer*
Ufes	Universidade Federal do Espírito Santo
UI	*User Interface*
URL	*Uniform Resource Locator*
VM	*Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO 9
1.1 Apresentação 9
1.2 Objetivos 10
1.2.1 Objetivo Geral 10
1.2.2 Objetivos Específicos 10
1.3 Trabalhos Relacionados 11
1.4 Estrutura do Texto 12

2 REFERENCIAL TEÓRICO 13
2.1 Espaço Inteligente 13
2.2 Interface entre Programas 15
2.3 Interface Gráfica de Usuário 17

3 PROPOSTA 20
3.1 Contextualização 20
3.2 Metodologia 21
3.3 IS 23
3.4 Backend 24
3.5 FrontEnd 25

4 RESULTADOS 27
4.1 Alicerce 27
4.2 UI 30
4.2.1 Autenticação 31
4.2.2 Tela Inicial 31
4.2.3 Cabeçalho 33
4.2.4 Menu Lateral 35
4.2.5 Painel de Controle 36
4.2.6 Visualização das Câmeras 37
4.2.7 Lista de *Pods* 38
4.2.8 Lista de Serviços 39
4.2.9 Cadastro de Serviços 40
4.2.10 Ações Extras 42
4.2.11 Papéis de Usuários 43

5 CONCLUSÃO E TRABALHOS FUTUROS 45

REFERÊNCIAS 46

1 INTRODUÇÃO

1.1 Apresentação

Ocorreu um grande desenvolvimento em sensores e tecnologia de detecção nas últimas décadas. Do surgimento das câmeras RGB-Depth ao aperfeiçoamento dos sensores, essas tecnologias têm entregado muito mais do que o simples registro de dados. Por isso, esses sensores e tecnologias têm sido aplicadas não só para fornecer dados do ambiente, mas também permitir uma maior interação de usuários com o ambiente e dispositivos presentes no espaço, assim como agregar mais inteligência a esses ambientes. Por exemplo, os espaços inteligentes são um cenário de aplicações a longo prazo, que requerem a detecção das informações de contexto do ambiente e dos usuários e a compreensão de sua intenção de interação (WU; YU; SHI, 2018).

Espaços inteligentes ganharam atenção significativa nos últimos anos devido aos avanços na tecnologia e facilidade de implantação. Esses espaços combinam *hardware* de dispositivos pequenos e eficientes com mecanismos de gerenciamento de dados para fornecer soluções em vários domínios, incluindo saúde, bem-estar, educação, etc. Um dos aspectos importantes dos espaços inteligentes, comparados aos espaços tradicionais, é que existe uma interação constante entre as pessoas e o ambiente circundante, e essas interações são capturadas pelo *hardware* implantado. Para algumas aplicações, como saúde, é imperativo analisar a interação ser humano-dispositivo para melhor entender os padrões de comportamento das pessoas em tais ambientes. Muitos espaços usam sensores como câmeras e microfones com o objetivo de observar os usuários e procurar entender suas interações e intenções (SHELKE; HARBOUR; AKSANLI, 2018). Por outro lado, à medida que os espaços inteligentes ganham ou acumulam uma infinidade de sensores, aplicações e serviços, o seu monitoramento se torna muito mais complexo. Nesse sentido, as interfaces gráficas podem desempenhar um papel importante.

As interfaces gráficas de usuário estão hoje em toda parte, até mesmo em telefones do cotidiano. De fato, os principais sistemas operacionais de computadores pessoais, como Microsoft Windows, são baseados em interface gráfica de usuário. Portanto, aproveitar dessa interface homem-computador é algo básico, e o resultado será um público mais amplo para um programa (MARTINEZ, 2011).

Uma interface gráfica de usuário fornece uma maneira fácil de pessoas e computadores

interagirem e se comunicarem. Essas tiram proveito da grande capacidade gráfica dos computadores para tornar esta comunicação mais acessível, ocultando os detalhes da programação do usuário. Esse tipo de interface usa janelas, ícones, menus, botões, listas suspensas, caixas de diálogo e mais, como um meio de comunicação entre humanos e computadores. Esses *widgets* gráficos são geralmente ativados quando o usuário manipula a interface com um *mouse* ou outro dispositivo apontador (MARTINEZ, 2011).

Com o fomento de sua pesquisa, os espaços inteligentes atraíram muitos olhos especialistas e leigos para si, além de ser naturalmente um sistema compartilhado por múltiplos usuários (SHELKE; HARBOUR; AKSANLI, 2018). Essas características levam à necessidade de uma interface para usuários completa e acessível, seja para o desenvolvimento, apresentação ou utilização final, de forma a agilizar, facilitar e potencializar sua utilidade.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma ferramenta para a fácil utilização (visualização e gerenciamento) do sistema intitulado *Intelligent Space* (IS) desenvolvido e mantido pelo laboratório Laboratório de Visão Computacional e Robótica (Lab VISIO) da Universidade Federal do Espírito Santo (Ufes) (CARMO et al., 2019), utilizando-se de *frameworks* consolidados em comunidade de *sofware* online, como GitHub (GITHUB INC., 2020), para realizar o desenvolvimento do projeto. A intenção é obter uma interface entre o IS e outros sistemas e uma interface gráfica para usuários leigos e avançados do sistema.

1.2.2 Objetivos Específicos

Como objetivos específicos, pode-se citar:

a) Desenvolver uma interface entre o IS e outros sistemas;

b) Desenvolver uma interface de usuário para o IS;

c) Aplicar ferramentas de proteção, como gerenciamento de usuários;

d) Automatizar a inicialização de aplicações e serviços do IS;

e) Visualizar informações gerais do IS;

f) Gerenciar aplicações e serviços no IS.

1.3 Trabalhos Relacionados

O IS tem como base da sua arquitetura o Kubernetes, que é "um produto Open Source utilizado para automatizar a implantação, o dimensionamento e o gerenciamento de aplicativos em contêiner" (CLOUD NATIVE COMPUTING FOUNDATION, 2020).

De forma simples, o Kubernetes é uma ferramenta de orquestração de contêineres, que são elementos que separam a aplicação da infraestrutura tornando a implantação mais fácil em diferentes ambientes de nuvem ou sistema operacional. Para cumprir o seu objetivo o Kubernetes utiliza de alguns mecanismos como: Pods que são a menor unidade computacional do ambiente Kubernetes; Deployment que promove declarativas para Pods e outros recursos; Namespace que são utilizados para isolar grupos de recursos em um mesmo cluster.

Para se ter algum grau de comparação deste trabalho com outros, foram pesquisadas outras interfaces de usuário (UI, do inglês *User Interface*) para Kubernetes que são populares através da documentação[1] como:

- **Integrada**[2]: é uma interface gráfica básica mantida pela equipe do Kubernetes cobrindo apenas aspectos essenciais do Kubernetes sem focar muito no desenvolvimento da interface gráfica;
- **Skooner**[3]: na mesma linha da interface Integrada com algumas funcionalidades a mais como ser 100% responsivo e uma interface mais amigável e bem desenvolvida;
- **Konstellate**[4]: uma experiência mais gráfica e interativa do que outras no mercado como conectar dois recursos Kubernetes apenas desenhando com o mouse uma linha entre eles;
- **Kubernator**[5]: uma interface que foca no gerenciamento em baixo nível focando em usuários mais experientes que querem mais funcionalidade e menos enfeites na interface.

Apesar de significativas no mercado, as soluções apresentadas não cobrem todos os aspectos do IS e nem levam em conta o contexto em que o IS é aplicado. Outro fator é a UI ser inflexível por ser soluções já prontas.

[1] https://kube-web-view.readthedocs.io/en/latest/alternatives.html
[2] https://github.com/kubernetes/dashboard
[3] https://github.com/skooner-k8s/skooner
[4] https://github.com/jeremykross/konstellate
[5] https://github.com/smpio/kubernator

1.4 Estrutura do Texto

O presente trabalho está estruturado nos seguintes capítulos:

- **Introdução**: tem como objetivo introduzir o leitor ao contexto deste trabalho e informá-lo de seus objetivos e contexto em relação a outros trabalhos;
- **Referencial teórico**: apresenta a base teórica para situar o leitor e dar suporte a um melhor entendimento dos motivos, propostas e resultados deste trabalho;
- **Proposta**: descreve os detalhes sobre a realização deste trabalho como também os passos planejados e executados para o desenvolvimento deste;
- **Resultados**: mostra os resultados obtidos com a implementação da proposta apresentada;
- **Conclusão e trabalhos futuros**: traz uma discussão sobre o que foi realizado e indica os trabalhos futuros para melhorias no estado atual da UI desenvolvida.

2 REFERENCIAL TEÓRICO

Neste capítulo, será apresentada a base teórica para o trabalho aqui dissertado. Ao fim do capítulo, os seguintes assuntos terão sido abordados: espaços inteligentes, interfaces entre programas e interfaces gráficas de usuário.

2.1 Espaço Inteligente

As possíveis aplicações com sensores e atuadores cresceu em conjunto com o desenvolvimento e progresso dos mesmos. Nessa evolução surgiu o conceito *smart* ou inteligente, definição de cunho científico, filosófico e comercial que abrange tanto sensores e atuadores, bem como as aplicações que os utilizam.

Esse conceito, quando aplicado, remete à capacidade de entender o contexto atual e atuar e se comunicar com ele. Por exemplo, uma geladeira inteligente que identifica o contexto sabendo exatamente o que está dentro dela, pode agir requisitando pedidos de compras e se comunica avisando ao usuário. Tais capacidades geralmente são obtidas através de detecção, computação, comunicação e controle integrados (LIU et al., 2007).

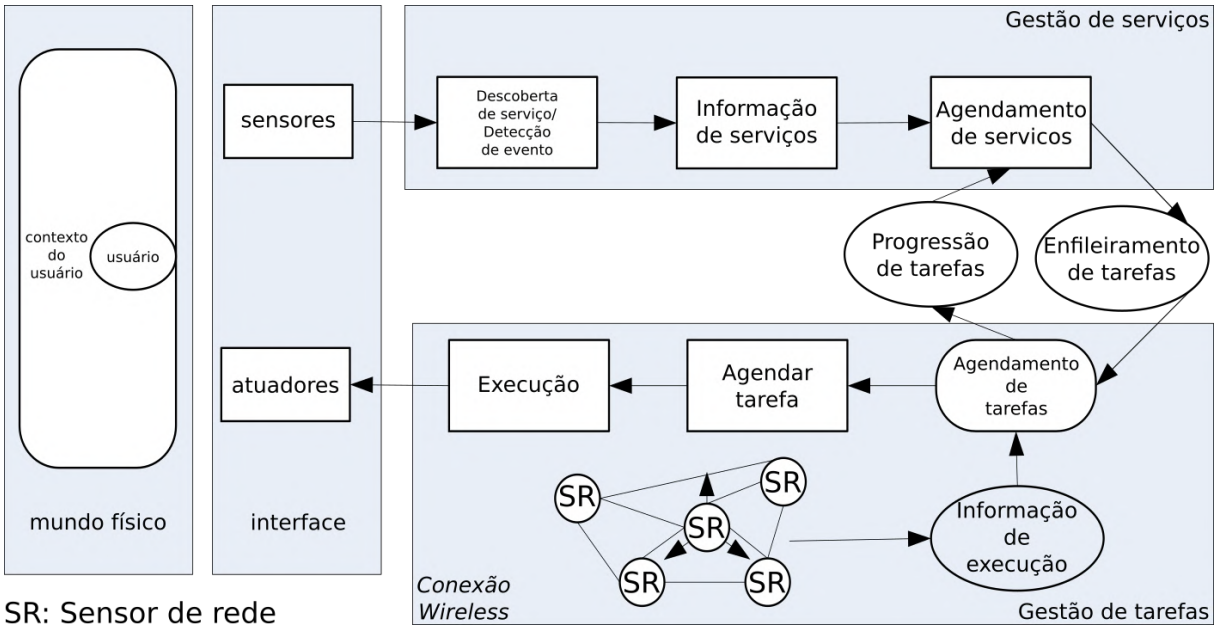
Quando se aplica esse conceito a um espaço, surge o que se pode chamar de espaços inteligentes, uma coleção de sensores, atuadores e unidades de computação comunicando uns com os outros para tornar-se uma parte importante de uma rede de informações e implementar algumas ações (LIU et al., 2007).

Para Fernandes (2014, p. 25) um espaço inteligente pode ser definido como:

> um ambiente equipado com uma rede de sensores (por exemplo câmeras, microfones e termômetros), que obtém informações sobre o mundo que observa, e uma rede de atuadores (robôs móveis, telas de informação, eletrodomésticos automatizados, entre outros), que permitem sua interação com os usuários e alteração do próprio ambiente. Tanto os sensores quanto os atuadores devem ser governados por um sistema capaz de coletar e analisar informações obtidas pelos sensores e tomar decisões.

A forma de se reproduzir um espaço inteligente é algo consolidado. Na Figura 1, pode-se observar um exemplo de arquitetura genérica de espaço inteligente que permite a um ambiente todas as capacidades citadas conferindo a abstração de inteligente a esse.

Figura 1 – Arquitetura genérica de espaços inteligentes.



Fonte: Liu et al. (2007).
Nota: Traduzido pelo autor.

Dissecando essa arquitetura, pode-se dividi-la em três grandes partes: o mundo físico; o mundo digital; e a interface que funciona como ponte entre esses mundos.

No mundo físico se encontra o usuário do espaço inteligente, o contexto desse usuário e do mundo. É de onde serão retiradas informações e realizadas ações.

O mundo digital possui a maior parte da computação e inteligência do espaço. Nele se encontram a gestão de serviços, que gerencia entradas dos usuário ao sistemas, e quais macroações serão tomadas. Já na gestão de tarefas, são coordenadas quais microações são necessárias para efetivar os serviços e as ações do mundo digital no ambiente. Esses dois se comunicam a partir da criação de filas de tarefas e o estado de cada tarefa.

A interface é o elo entre esses dois mundos. Composta majoritariamente de sensores e atuadores, a interface realiza a comunicação entre o mundo físico e o mundo digital além de poder possuir dispositivos inteligentes.

A aplicação de espaços inteligentes se mostrou muito diversificada abrangendo várias áreas de atuação humana (WRIGHT; STEVENTON, 2004). São encontradas muitas aplicações, como detecção de pessoas (ALMONFREY et al., 2018), reconhecimento de gestos (CASILLAS-PEREZ et al., 2016), casas inteligentes (HUANG; WU; LIU, 2013), ambientes industriais (SO; SUN, 2010) e controle de robôs (CARMO et al., 2020).

Pode-se considerar que espaços inteligentes são uma integração de muitos subsistemas com a capacidade de coletar informações discretamente para inferir e atender às necessidades contextualmente específicas de seres humanos ou do ambiente (WRIGHT; STEVENTON, 2004).

2.2 Interface entre Programas

Dois dos pilares de um espaço inteligente são a computação e a comunicação. Esta última não diz respeito apenas à troca de informações entre dois dispositivos ou entre o mundo físico e o mundo digital, mas também à comunicação entre os núcleos de computação, ou programas.

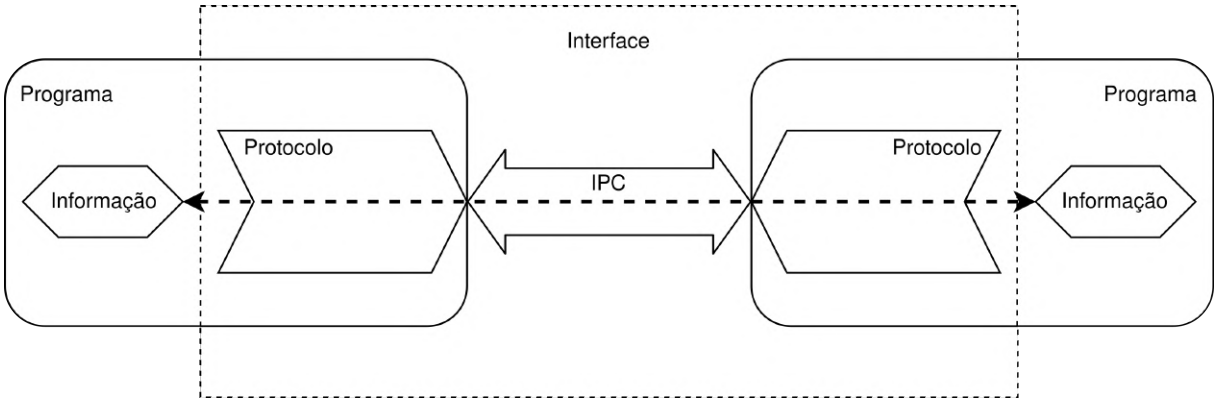
À medida que os programas se tornaram maiores e mais complexos, uma tendência natural foi a modularização dos mesmos, muitas vezes até separando-os em vários programas distintos. Esse comportamento levou, inicialmente, a um isolamento da informação, fazendo-se necessário o surgimento de formas de comunicação entre essas partes, as interfaces entre programas.

Tal troca de informação tomou diversas formas dependendo do contexto computacional e das preferências de quem desenvolve o sistema. A comunicação entre processos (IPC, do inglês *Inter-Process Communication*) é o termo que engloba as soluções para essa comunicação. Algumas das IPCs mais conhecidas são *pipes*, soquetes de domínio Unix, memória compartilhada (VENKATARAMAN; JAGADEESHA, 2015), soquetes de redes (BISHOP et al., 2005) e outros.

Escopos mais modernos possuem estruturas e protocolos baseados em outras IPCs para estabelecer ambientes com vantagens diversas como o TCP, UDP (BISHOP et al., 2005), Redis (REDIS LTD, 2021), AMQP, MQTT, HTTP (SELEZNEV; YAKOVLEV, 2019). Baseando-se nessas estruturas e protocolos, a maioria das interfaces entre programas foi concebida. Na Figura 2, pode-se visualizar um exemplo de interface onde a informação sai de um programa, passa por um protocolo, é transmitida através de uma IPC, interpretada e chega no outro programa.

Uma dessas IPCs que se popularizou muito devido à expansão da rede mundial de computadores foi a Interface de Programação de Aplicação (API, do inglês *Application Programming Interface*).

Figura 2 – Exemplo de interface entre programas.



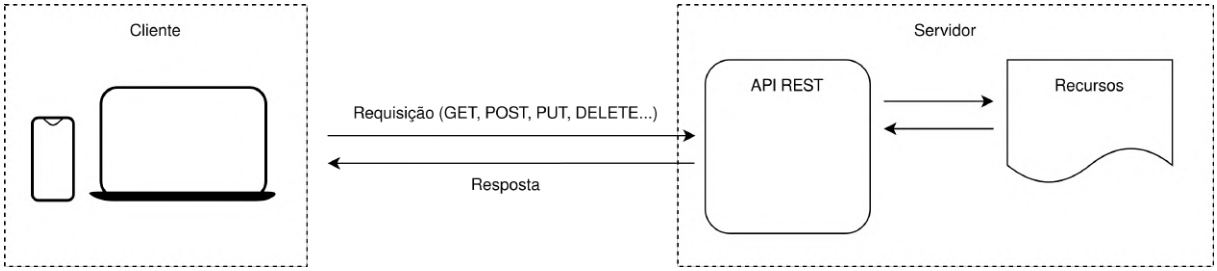
Fonte: Produção do próprio autor.

De forma geral, uma API expõe um conjunto de dados e funções para facilitar as interações entre programas de computador e permitir a troca de informações. O estilo de arquitetura Transferência Representacional de Estado (REST, do inglês *Representational State Transfer*) é comumente aplicado ao projeto de APIs para serviços modernos. Uma API em conformidade com a arquitetura REST é uma API REST. Uma aplicação que possui uma API REST pode ser denominada "RESTful" (MASSE, 2011).

Uma chamada à API REST é baseada em um localizador uniforme de recursos (URL, do inglês *Uniform Resource Locator*) simples que especifica os dados necessários e o formato retornado. Tais chamadas definem dados a serem inseridos, editados ou requisitados ao sistema, ou ações a serem tomadas (YATES et al., 2015).

Na Figura 3, encontra-se um exemplo de comunicação do tipo cliente-servidor utilizando-se uma API REST. O cliente requisita à API um recurso ou uma ação através do protocolo HTTP, a API interpreta tal requisição e tenta realizar o pedido retornando ao cliente o recurso desejado, sucesso da ação ou algum erro.

Figura 3 – Exemplo de comunicação utilizando API REST.



Fonte: Produção do próprio autor.

2.3 Interface Gráfica de Usuário

A UI é uma parte essencial de quase todas interações humano-máquina, importante para tomadas de decisões, controle ou muitas vezes apenas para *feedback* do estado da máquina.

Pode-se dividir as UIs de acordo com o seu propósito e sua implementação. Pelo propósito, tem-se que a UI pode ser de saída, de entrada ou ambos, determinando o sentido da comunicação dos dados, ou seja, se a informação vai da máquina para o humano, o contrário ou os dois.

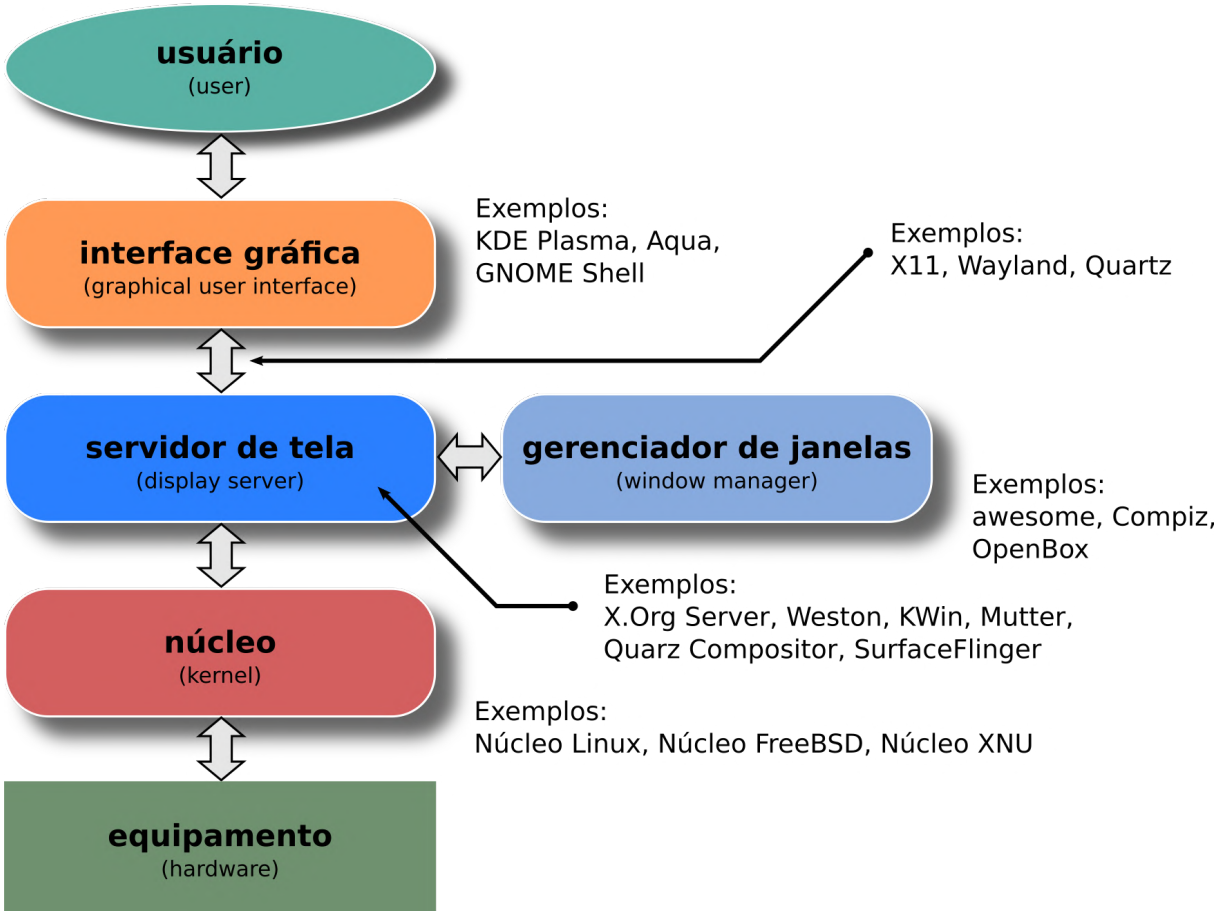
Já pela implementação, as UIs podem ser analógicas ou digitais, de acordo com os tipos de componentes que foram utilizados em sua constituição. Não podendo ser confundidas com a forma que a informação é apresentada por UIs de saída, que também pode ser analógica ou digital no sentido de informação contínua ou discreta. Por exemplo, um relógio de pulso analógico, ou seja, construído a partir de componentes analógicos, possui uma UI de saída analógica porém a informação nesta UI é apresentada ao utilizador de forma digital, os dados são discretos.

Uma forma comum que as UIs se apresentam, principalmente as de saída, é a gráfica, também conhecidas como interface gráfica de usuário (GUI, do inglês *graphical user interface*). A apresentação gráfica de informações utiliza a capacidade de processamento de uma pessoa com muito mais eficácia do que outros métodos de apresentação. Se elaborada de forma apropriada, ela reduz a necessidade de recodificação e reorganização de informações perceptivas e mentais, e também as cargas de memória (GALITZ, 2007).

Ela também permite uma transferência mais rápida de informações entre computadores e pessoas, proporcionando uma comparação mais visual de quantidades, tendências ou relações, representações mais compactas de informações e simplificação da percepção da estrutura. Os gráficos também adicionam apelo ou charme à interface e permitem uma maior personalização para criar um estilo corporativo ou organizacional original (GALITZ, 2007).

As GUIs se popularizaram muito, principalmente com a evolução dessas nos sistemas operacionais. Na Figura 4, pode-se observar o quão complexa e diversificada pode ser uma GUI de um sistema operacional para permitir que um equipamento e um usuário se comuniquem.

Figura 4 – Esquema das camadas da interface gráfica.



Fonte: Traian (2013).
Nota: Traduzido pelo autor.

Existem, também, exemplos práticos que demonstram como uma GUI viabiliza a usuários leigos interagirem e até mesmo usarem todos os aspectos de um sistema (BRUN et al., 2017), mesmo que não tenham conhecimento técnico sobre o mesmo, como conhecimento em linguagem de programação. Um caso é a programação em blocos como mostrado na Figura 5, que permite até mesmo crianças sem conhecimento prévio de programação programarem robôs (QUEIROZ; SAMPAIO, 2016).

Tais vantagens levam a expectativas otimistas na criação de uma GUI com o objetivo de facilitar o acesso de usuários a programas. Entretanto, é preciso ter atenção e cuidado quando se cria uma UI. Numerosos acidentes e desastres foram atribuídos a um *design* mal feito para uma UI. Todos os dias UIs ruins resultam em taxas de erro maiores, custos de treinamento mais altos e rendimento reduzido. Isso custa dinheiro às empresas e causa estresse para aqueles que interagem com as UIs, a saber, os usuários (STONE et al., 2005). Ainda assim, UIs bem feitas e planejadas trazem muito mais benefício aos sistemas, facilitando a sua utilização.

Figura 5 – Exemplo de GUI para programação em blocos.

Fonte: Batista et al. (2017).

3 PROPOSTA

Neste capítulo serão apresentadas as ideias propostas para se alcançar o objetivo especificado na Seção 1.2.

3.1 Contextualização

Espaços inteligentes tornaram-se fornecedores de inúmeras aplicações, que possuem apelos diversos como científicos, culturais, econômicos e comerciais, resultando em um grande interesse da comunidade científica e do mercado neste tópico (WRIGHT; STEVENTON, 2004).

Devido à diversidade de aplicações possíveis, a pluralidade de usuários em sistemas baseados em espaços inteligentes é enorme, tanto no âmbito de desenvolvimento quanto no de utilização final. Pensando em tal pluralidade, um sistema que seja de fácil reconhecimento e utilização é, inegavelmente, benéfico ao mesmo.

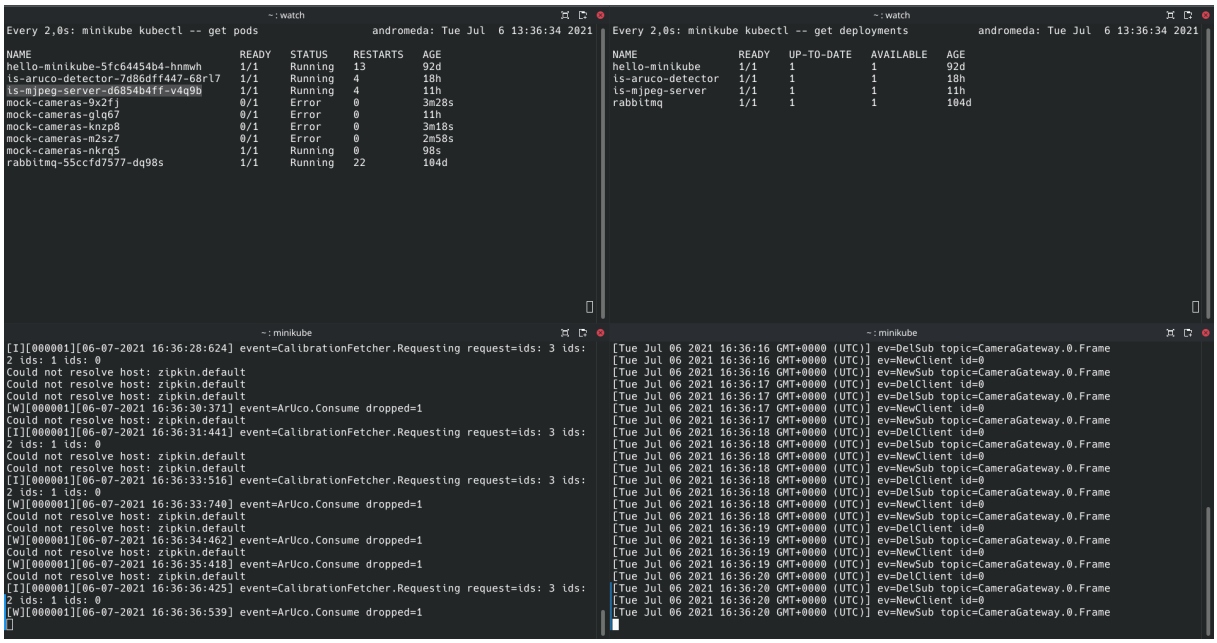
Atualmente o IS, desenvolvido no Lab VISIO da Ufes, é manuseado, gerenciado e observado através de telas de terminal, como pode ser visto na Figura 6, o que dificulta e complica muitos aspectos de utilização do sistema.

Com a intenção de melhorar a interface para gerenciamento e utilização do IS, foi proposta a criação de uma GUI para sanar alguns desses problemas e modernizar o sistema. Tendo em vista os benefícios que uma interface gráfica pode trazer e a facilidade de utilização que ela pode agregar a sistemas mais complexos, a GUI permitirá que o IS seja utilizado por pessoas com os mais diferentes graus de maturidade tecnológica facilitando a visualização de informações chave, descomplicando sua utilização e colaborando para com sua disseminação e o seu gerenciamento.

O IS atual exige um conhecimento técnico aprofundado para manipulá-lo. É preciso estar a par de alguns programas que são utilizados em sua arquitetura base e ter conhecimento com terminal e comandos de terminal, levando a uma grande curva de aprendizado para novos usuários e a inabilidade para usuário leigos.

A legibilidade de informações é outro fator que fica aquém dos padrões modernos. Está

Figura 6 – Exemplo de terminal ao observar alguns aspectos do IS.



Fonte: Produção do próprio autor.

claro na Figura 6 que as informações disponíveis não são fáceis de localizar e interpretar.

Também é proposto que a GUI permita aos usuários menos acostumados ao IS automatizar aplicações sem necessitar de conhecimentos mais aprofundados no IS ou de *script*, permitindo mais facilidade e rapidez.

Outro fator interessante em modernizar o IS é a publicidade do mesmo, permitindo uma maior visibilidade e apresentação.

Finalmente, uma vantagem a mais ao desenvolver uma GUI própria para o IS é a possibilidade de personalização. GUIs existentes para partes do IS não cobrem todos os seus aspectos e nem levam em conta o contexto em que o IS é aplicado.

3.2 Metodologia

Após o entendimento do problema atual que o IS possui em relação à sua UI, foram pesquisadas quais soluções existentes nesse contexto poderiam contribuir para a solução do problema. Apesar das inúmeras opções encontradas, nenhuma delas dava a liberdade necessária e possibilidade de inclusão de todas as aplicações que o cenário do IS exigia. Algumas não possuem uma tradução para português, outras por serem projetadas para o

caso genérico, não possuem soluções para ações que o IS exerce a mais que outros espaços inteligentes da área.

Por isso e por outras razões explicitadas na Sesão 3.1, a opção de se desenvolver uma GUI própria para o IS se mostrou a mais adequada. Na Figura 7 pode-se encontrar a ideia básica do trabalho que consiste em uma UI e uma forma de comunicação entre a UI e o IS.

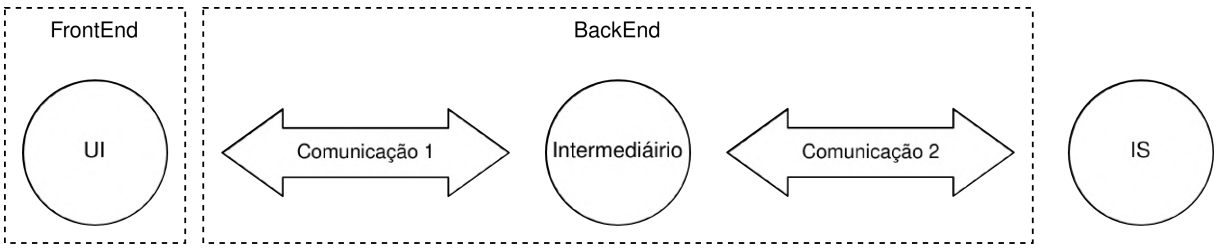
Figura 7 – Esquema básica do trabalho.

Fonte: Produção do próprio autor.

Para isso iniciou-se uma revisão bibliográfica da área de aquisição de dados e gerenciamento de sistemas distribuídos utilizando o Kubernetes. Além disso, foram analisados os principais *frameworks* existentes utilizados para criação de APIs REST, gerenciamento de usuários, UI e interação com o Kubernetes.

Baseado na revisão bibliográfica, foi constatado que uma melhor divisão do projeto poderia resultar numa solução mais robusta, segura e de mais fácil desenvolvimento e manutenção (Figura 8) (ZIRKELBACH; KRAUSE; HASSELBRING, 2019). Diferente da ideia básica de ter todo o trabalho num único lugar, o trabalho foi dividido em módulos com funções específicas, um focada no usuário (*frontend*) e um focado nas comunicações, gerenciamentos e processos necessários (*backend*).

Figura 8 – Esquema final do trabalho.



Fonte: Produção do próprio autor.

Devido à situação atual de pandemia, causada pelo COVID-19, além da realização de reformas no prédio do CT II (Centro Tecnológico II) na UFES, o desenvolvimento deste trabalho foi dificultado. Além do impedimento de acesso ao prédio, a infraestrutura do IS

teve de ser desativada por causa da obra, impedindo a realização de testes no laboratório. Mesmo assim, o projeto pôde ser desenvolvido e testado. Para a sua realização, foi adotada uma abordagem de emulação local do IS, o que permitiu a execução de serviços e aplicações como se o IS ainda estivesse em funcionamento normal. Quando necessário, vídeos de gravações realizadas no ambiente real foram utilizados para emular a captura de imagens. Mais detalhes serão descritos na Seção 3.3.

Definidas as principais diretrizes, foi desenvolvido um sistema com os seguintes módulos: um módulo de interface entre o IS e o meio exterior, por meio de uma ferramenta que interaje com o Kubernetes e uma API REST (*back-end*), o qual será discutido na Seção 3.4; e um módulo de GUI entre o módulo anterior e o usuário final, com proteções como autenticação de usuário (*front-end*) que será tratado na Seção 3.5.

Em cima deste conjunto, foram desenvolvidos exemplos de automatização de inicialização de aplicações, uma *dashboard* simples com dados sobre o IS como uso de memória e CPU, lista de aplicações e *pods* (que é a menor unidade computacional gerenciada pelo Kubernetes) e uma GUI para visualização das câmeras disponíveis no IS.

3.3 IS

Devido a obras no CT-II onde se encontra parte do IS e também ao contexto social contemporâneo de pandemia COVID-19, o acesso e consequentemente integração deste trabalho ao IS foi dificultado, o que resultou na adoção de uma abordagem de emulação local do IS para a realização dos testes e validação.

Para isso foram utilizados arquivos de configuração para se estruturar minimamente o IS em um ambiente Kubernetes e para executar alguns serviços e aplicações leves em um computador pessoal. Entre tais funcionalidades estão um servidor simples que abre uma porta de rede para acesso às imagens das câmeras do laboratório e uma ferramenta que detecta marcadores ArUcos (MUNOZ-SALINAS, 2012) através das câmeras.

Com relação às câmeras, foi produzido pelo Lab VISIO uma aplicação *mock* para o IS que emula câmeras reais através de filmagens e dados previamente gravados. Tal aplicação, que também roda na própria arquitetura do IS, foi disponibilizada para utilização neste trabalho.

Para a implantação do ambiente Kubernetes, a fim de embarcar o IS em uma máquina local, foi utilizado o Minikube que pode ser definido como uma implementação leve do

Kubernetes. O Minikube cria uma máquina virtual (VM, do inglês *Virtual Machine*) na máquina local e implanta um *cluster* simples contendo apenas um nó (CLOUD NATIVE COMPUTING FOUNDATION, 2021b).

Como o ambiente oficial que roda o IS está utilizando a versão 1.14.0 do Kubernetes, por motivos de compatibilidade, essa foi a versão escolhida para ser instalada localmente. Para facilitar a utilização do Kubernetes também foi instalada uma ferramenta de linha de comando chamada Kubectl na versão 1.14.0, utilizada principalmente para implantar aplicações, inspecionar e gerenciar recursos de *cluster* e visualizar registros (CLOUD NATIVE COMPUTING FOUNDATION, 2021a), os exemplos que aparecem na Figura 6 foram obtidos utilizando-se essa ferramenta.

3.4 Backend

Como pode ser visto na Figura 8, o *backend* é constituído de três elementos principais, a comunicação com o IS, a comunicação com o *frontend* e o intermediário que é a base para essas duas estruturas.

O Kubernetes possui uma API REST interna justamente para auxiliar na sua comunicação com outros programas. Além disso, existem bibliotecas para se utilizar como cliente para o Kubernetes mantidos pela organização Kubernetes Clients[1], estando dispovíveis em várias linguagens diferentes.

A linguagem escolhida para o *backend* foi Python por ser a mais popular entre os clientes Kubernetes e pelo fato do Lab VISIO já possuir um grande *portfolio* e experiência nessa linguagem.

Dada essas escolhas, o cliente selecionado foi o cliente Kubernetes Python versão 10.1.0[2] por compatibilidade de versão com o Kubernetes 1.14.0.

Para o desenvolvimento da API REST, que é usada pelo *frontend*, foi identificado que os *frameworks* mais populares para esta aplicação no GitHub[3] em Python são Django, Flask e Httpie. Ao se fazer uma pesquisa mais profunda, o Flask[4] foi selecionado devido à sua simplicidade e praticidade que se alinha melhor com o escopo deste trabalho por ser uma

[1] https://github.com/kubernetes-client
[2] https://github.com/kubernetes-client/python/tree/release-10.0
[3] https://github.com/EvanLi/Github-Ranking
[4] https://flask.palletsprojects.com/en/2.0.x/

ferramenta que exige menos configurações para se estruturar um ambiente básico.

Assim, utilizando-se as bibliotecas cliente Kubernetes e Flask, foi desenvolvido em Python o *backend* com comunicação com o IS e uma API REST para ser acessada pelo *frontend*. Além disso, foi desenvolvido um sistema de controle de acesso através de par de chaves usuário-senha e a concepção de um novo recurso chamado Serviço abstraindo inúmeros recursos do Kubernetes como *pods*, *services*, *jobs*, etc, e abrangendo alguns conceitos particulares do IS.

Para o gerenciamento de usuários foi feito um esquema simplificado com uma biblioteca de criptografia e três papéis fixos, de administrador, de operacional e de usuário comum. A persistência dos dados relacionados aos usuários e ao recurso Serviço foi feita apenas usando arquivos json. Tais abordagens simples foram tomadas pois a equipe de desenvolvimento do IS ainda não definiu quais tecnologias serão usadas para esses casos.

3.5 FrontEnd

O *frontend* faz a ponte do usuário com o resto do sistema. Como o usuário é a principal preocupação deste trabalho, esta parte se torna fundamental, definindo a GUI e a maioria das interações que serão estabelecidas.

Inicialmente foi levantada a ideia de se fazer essa parte em Python para manter um mesmo padrão com o *backend*. Porém, após pesquisas e como os módulos *frontend* e *backend* são totalmente independentes, passou-se a levar em consideração o uso da linguagem Javascript pelo seu maior número de exemplos, *framworks* para UI e comunidade nesta área.

Procurando pelos *frameworks* mais famosos para desenvolvimento de *frontend* no GitHub, encontrou-se o Vue e o React, ambos utilizando o Javascript como base, o que reforçou mais a escolha para esta linguagem.

Ao comparar os dois *frameworks*, observou-se que o React possui maior curva de aprendizado se comparado com o Vue (ATALAIA, 2020). Como este projeto será mantido posteriormente por pessoas com diferentes níveis de conhecimento técnico, a escolha do *framework* Vue se mostrou mais adequado.

Escolhido o Vue como ferramenta base de desenvolvimento do *frontend*, uma decisão que precisou ser tomada foi a versão do Vue. Atualmente existem no mercado duas versões oficiais, o Vue 2 e o Vue 3. Apesar de mais moderno, o Vue 3 ainda é novo e, portanto,

ainda não está estabilizado nem estabelecido na comunidade, contendo mais *bugs* que seu antecessor e muito menos exemplos e ferramentas, o que levou à escolha da utilização do Vue 2[5] para este trabalho.

Para auxiliar no desenvolvimento de uma melhor UI, foi utilizada também uma biblioteca de componentes chamada Element UI[6], encontrada num repositório que seleciona uma lista de coisas "incríveis" relacionadas a Vue, o awesome-vue[7]. Sua escolha se deve à simplicidade, facilidade e praticidade em relação a outras opções como o Quasar.

Durante o desenvolvimento do trabalho, foi tomada a decisão de alterar a linguagem de Javascript para Typescript. Tal decisão não alterou nenhuma ferramenta já escolhida, por ser compatível com todas, e foi tomada considerando-se o aspecto plural de futuros contribuidores a este trabalho. Sendo construído em cima do Javacript e, no final, compilado para o mesmo, o Typescript apresenta uma verão tipada do Javascript, levando a uma linguagem com toda a liberdade que o Javascript propõe, porém com mais ferramentas para auxiliar no desenvolvimento, descoberta de erros antes de compilar e executar, além da criação de um ambiente mais propício para a manutenção do projeto.

[5] https://vuejs.org/
[6] https://element.eleme.io/
[7] https://github.com/vuejs/awesome-vue

4 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos ao desenvolver e implementar o que foi exposto no Capítulo 3.

4.1 Alicerce

Para se obter o objetivo principal deste trabalho, que é a UI, foi necessário desenvolver uma base em cima da qual a UI foi erguida. Nesta seção serão apresentados os frutos obtidos no desenvolvimento dessa estrutura.

No geral, o desenvolvimento se sucedeu fluidamente após todos os passos planejados. O maior desafio foi a utilização da biblioteca cliente Kubernetes Python devido à sua documentação pobre em relação à explicação de como funciona e exemplos de como usar.

Seguindo a documentação oficial, foi instalado, na máquina local, o ambiente Kubernetes com auxílio do Minikube. Utilizando-se o Kubectl, os arquivos disponibilizados pelo Lab VISIO foram implantados nessa estrutura, permitindo a emulação do IS. Na Figura 9, se encontra uma captura de tela da *dashboard* padrão do Kubernetes já executando o IS e algumas aplicações como o "is-aruco-detector".

A aplicação *backend* foi estruturada de acordo com a documentação do Flask e do guia para Python do Hitchhiker[1]. A implementação inicial do servidor foi simples com a ajuda de inúmeros exemplos da documentação do Flask.

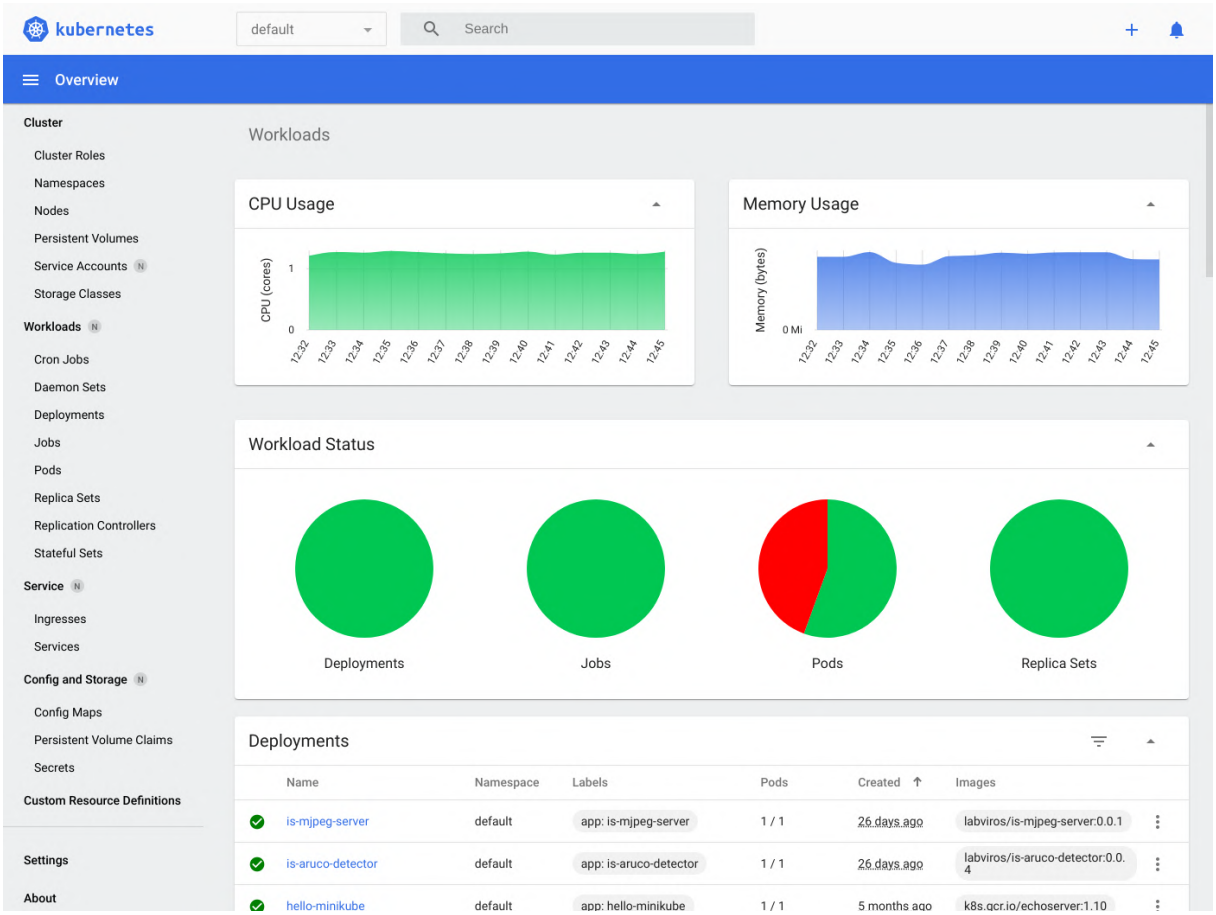
A implementação do cliente Kubernetes Python levou a várias dificuldades, algumas talvez relacionadas à versão escolhida por motivos de compatibilidade, mas ao estudar o código da biblioteca foi possível entender melhor como funciona e realizar o acesso do *backend* à API integrada do Kubernetes.

Baseado no código dessa biblioteca, foi produzida pelo autor uma ferramenta para acessar rotas personalizadas da API do Kubernetes que podem ser criadas por aplicações externas a esse como o IS. A exemplo foi desenvolvido um código que dá acesso do *backend* para a rota do Metric Server[2], que é um servidor de métricas, como uso de CPU e memória

[1] https://docs.python-guide.org/
[2] https://github.com/kubernetes-sigs/metrics-server

RAM, que cria uma rota personalizada e que não está disponível no cliente padrão do Kubernetes.

Figura 9 – Captura de tela da *dashboard* padrão do Kubernetes.



Fonte: Produção do próprio autor.

A API para comunicação entre o *backend* e o *frontend* foi desenvolvida criando rotas no servidor Flask à medida que foram sendo necessitadas pelo *frontend*. A seguir encontra-se a lista de rotas, com os métodos HTTP disponíveis, que foram desenvolvidos para a finalização deste trabalho:

a) /login:
   - GET/POST: Autenticação do usuário na aplicação retornando um *token* que é necessário para acessar as rotas que precisam de autenticação;

b) /me:
   - GET: Obtenção das informações de um usuário dado um *token*;

c) /services:
   - GET: Obtenção de uma lista de todos os serviços cadastrado no sistema;

  - POST: Cadastro um novo serviço;

d) /services/{id}:
  - GET: Obtenção de um serviço cujo identificador seja {id};
  - PATCH: Alteração do cadastro de um serviço cujo identificador seja {id};
  - DELETE: Exclusão do cadastro de um serviço cujo identificador seja {id};

e) /services/{id}/deployment:
  - POST: Cadastro do arquivo de configuração de em um serviço cujo identificador seja {id};
  - DELETE: Exclusão do arquivo de configuração de um serviço cujo identificador seja {id};

f) /services/{id}/start:
  - POST: Inicialização do serviço cujo identificador seja {id} no IS;

g) /services/{id}/stop:
  - POST: Suspensão do serviço cujo identificador seja {id} no IS;

h) /pods/:
  - GET: Obtenção de uma lista de *pods* do Kubernetes cujo *namespace* seja "default";

i) /pods/{name}:
  - GET: Obtenção de um *pod* do Kubernetes cujo *namespace* seja "default" e o nome {name};
  - DELETE: Exclusão de um *pod* do Kubernetes cujo *namespace* seja "default" e o nome {name};

j) /pods/{name}/log:
  - GET: Obtenção do *log* de um *pod* do Kubernetes cujo *namespace* seja "default" e o nome {name};

k) /metric/nodes:
  - GET: Obtenção das métricas dos nós do Kubernetes;

l) /metric/pods:
  - GET: Obtenção das métricas dos *pods* do Kubernetes.

A estrutura do *frontend* foi desenvolvida a partir de um modelo também desenvolvido utilizando Vue e Element UI, o vue-element-admin[3]. Para acessar a API do *backend* foi

[3] https://github.com/PanJiaChen/vue-element-admin

utilizada a biblioteca axios[4] e, para utilização de gráficos, a biblioteca vue-echarts[5]. As telas criadas serão expostas na Seção 4.2.

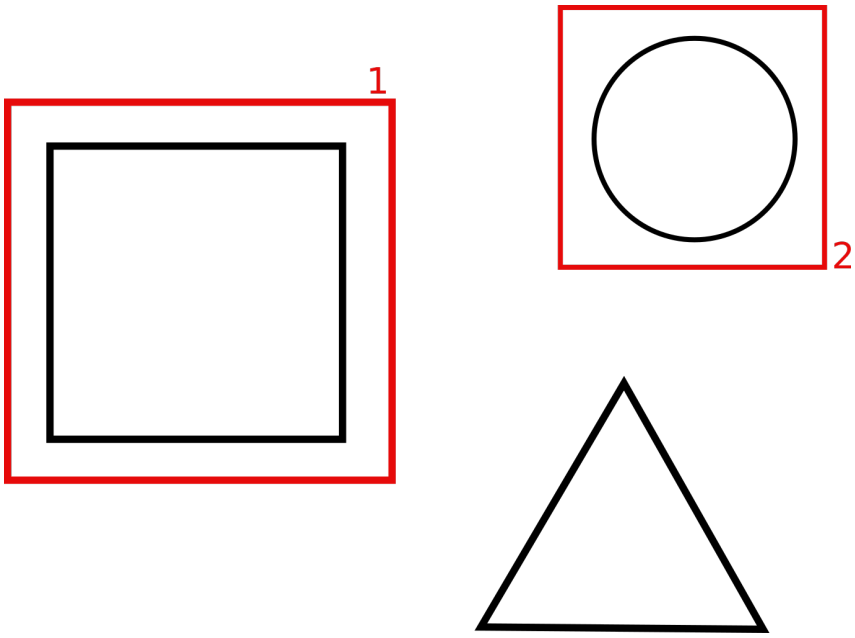
E por fim, para "empacotar" a aplicação, foi criada uma imagem Docker para o *backend* e o *fronend*. O Docker foi escolhido por ser utilizado pelo Lab VISIO e porque facilitará sua integração com o IS. O trabalho pode ser encontrado no GitHub do Lab VISIO[6].

4.2 UI

Nesta seção, serão apresentadas as telas da UI obtidas no desenvolvimento deste trabalho.

Para melhor explicar algumas das telas, será utilizado um sistema de marcações onde, quando necessário, as telas terão rótulos que estarão no título da figura entre parênteses e regiões da imagem serão contornadas de vermelho e numeradas. Para ilustrar, tem-se na Figura 10 a tela exemplo e as marcações exemplo-1 que é um quadrado e a marcação exemplo-2 que é um círculo.

Figura 10 – Exemplificação do sistema de marcação (exemplo).



Fonte: Produção do próprio autor.
Legenda: 1: exemplo-1; 2: exemplo-2.

[4] https://github.com/axios/axios
[5] https://github.com/ecomfe/vue-echarts
[6] https://github.com/labviros

De forma a prevenir o excesso de imagens e tornar a leitura desta seção mais simples, algumas figuras relacionadas a capturas de telas da UI foram editadas de forma a retirar espaços vazios e só mostrar a informação importante de cada tela.

4.2.1 Autenticação

Na Figura 11 encontra-se a tela de autenticação, única tela disponível para um usuário sem autenticação. Para acessar o resto da UI, o usuário deve entrar com as credencias corretas.

Figura 11 – Tela de autenticação.

Fonte: Produção do próprio autor.

4.2.2 Tela Inicial

Após o sucesso na autenticação, o usuário é levado à página inicial que pode ser vista na Figura 12. Na Figura 12a, encontra-se a tela inicial com o menu lateral retraído e, na Figura 12b, com o menu lateral expandido. Na marcação inicial-1 é encontrado o cabeçalho

da aplicação que também pode ser visto na Figura 14 e na marcação inicial-2 o menu lateral que pode ser visto na Figura 16.

Figura 12 – Tela inicial em seus estados de menu lateral retraído (a) e expandido (b) (inicial).

(a)

(b)

Fonte: Produção do próprio autor.
Legenda: 1: inicial-1; 2: inicial-2.

Na Figura 13 encontram-se as mesmas telas da Figura 12 em suas versões *mobile*.

Figura 13 – Tela inicial em sua versão *mobile* com o menu lateral retraído (a) e expandido (b).

(a) (b)

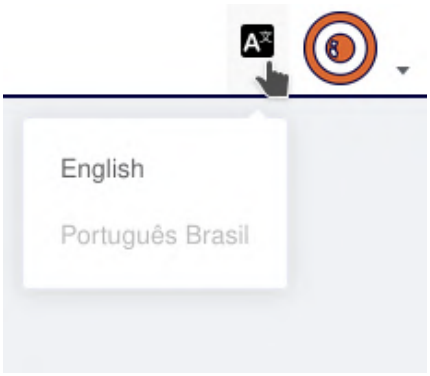
Fonte: Produção do próprio autor.

4.2.3 Cabeçalho

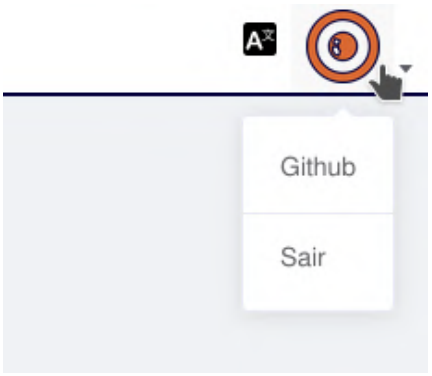
No cabeçalho (Figura 14) o usuário pode expandir e retrair o menu lateral através do ícone na marcação cabeçalho-1. De qualquer tela o usuário pode voltar à tela inicial através da logo do Lab VISIO (marcação cabeçalho-2). Na marcação cabeçalho-3 encontra-se a funcionalidade de alterar o idioma da aplicação. Outras opções podem ser vistas ao clicar na logo que está na marcação cabeçalho-4, como desconectar do usuário local e ir para o GitHub do Lab VISIO.

Figura 14 – Cabeçalho da UI (a) e as ações extras de trocar de idioma (b) e ver demais opções (c) (cabeçalho).

(a)



(b)



(c)

Fonte: Produção do próprio autor.
Legenda: 1: cabeçalho-1; 2: cabeçalho-2; 3: cabeçalho-3; 4: cabeçalho-4.

Na Figura 15 encontra-se um exemplo de tela em dois idiomas que foram implementados neste trabalho.

Figura 15 – Exemplo de uma tela em português (a) e inglês (b).

(a)

(b)

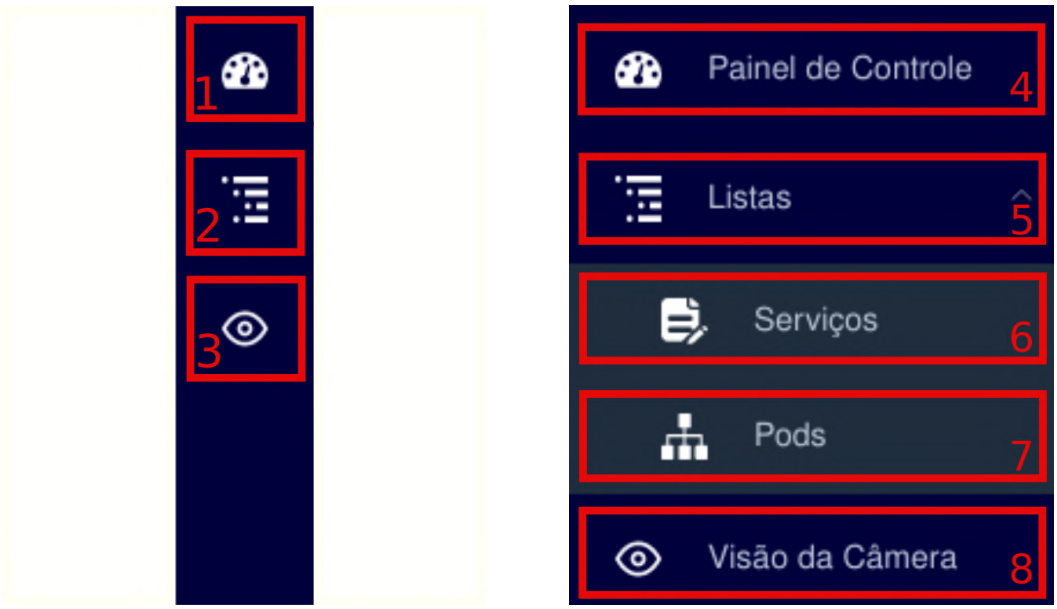
Fonte: Produção do próprio autor.

4.2.4 Menu Lateral

O menu lateral (Figura 16), possui duas formas, a retraída (Figura 16a) e a expandida (Figura 16b). Com ele é possível navegar pelos diferentes módulos da UI. Para este trabalho foram desenvolvidos quatro módulos. O primeiro é o painel de controle (Figura 17) que pode ser acessado pelas marcações menu-lateral-1 e menu-lateral-4. A visualização das câmeras (Figura 18) que pode ser acessado pelas marcações menu-lateral-3 e menu-lateral-8. E duas listas de recursos que podem ser acessadas pelas marcações menu-lateral-2 e

menu-lateral-5, abrindo, assim, as opções de lista de serviços que podem ser acessadas pela marcação menu-lateral-6 (Figura 21) e lista de *pods* (Figura 20) que pode ser acessada pela marcação menu-lateral-7.

Figura 16 – Menu lateral em suas formas retraída (a) e expandida (b) (menu-lateral).



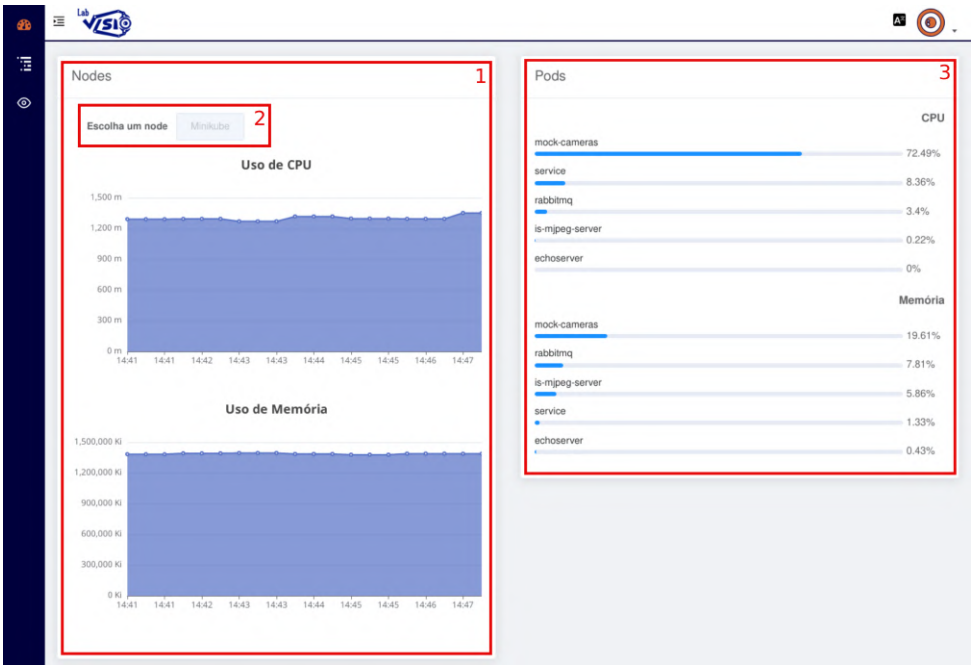
(a) (b)

Fonte: Produção do próprio autor.
Legenda: 1: menu-lateral-1; 2: menu-lateral-2; 3: menu-lateral-3; 4: menu-lateral-4; 5: menu-lateral-5; 6: menu-lateral-6; 7: menu-lateral-7; 8: menu-lateral-8.

4.2.5 Painel de Controle

No módulo de painel de controle (Figura 17) estão dois blocos de informações. A marcação dashboard-1 mostra gráficos de utilização de CPU e memória RAM por nó do Kubernetes, podendo ser escolhido o nó através das opções na marcação dashboard-2. Por motivos de implementação local do Kubernetes através do Minikube, só há um nó. E na marcação dashboard-3, uma lista de *pods*, com suas métricas relativas ao total consumido, é mostrada.

Figura 17 – Tela de painel de controle (dashboard).

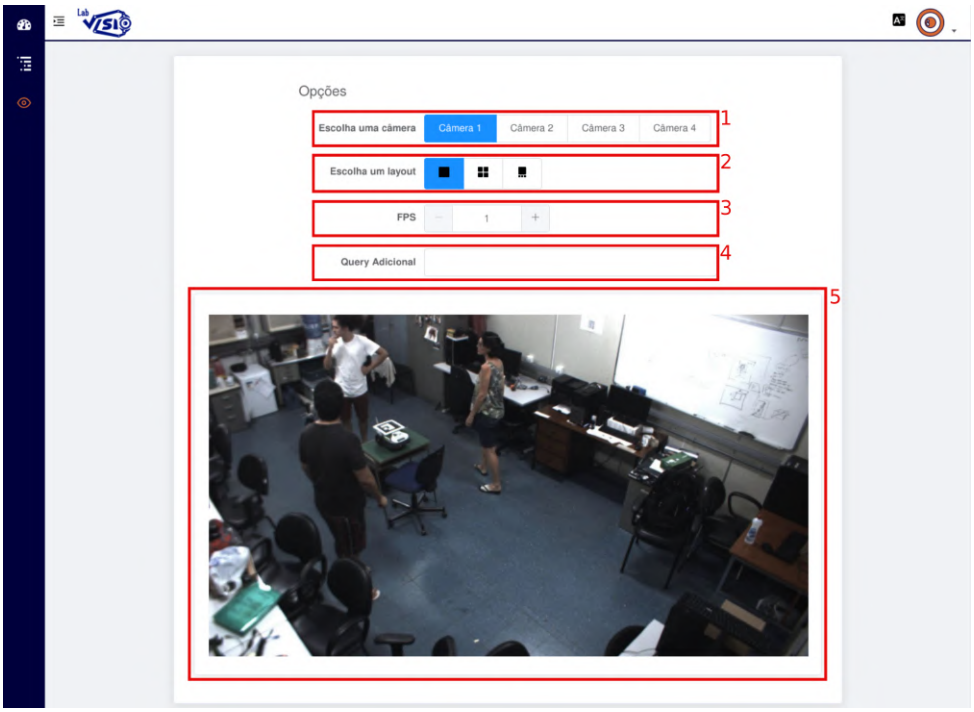


Fonte: Produção do próprio autor.
Legenda: 1: dashboard-1; 2: dashboard-2; 3: dashboard-3.

4.2.6 Visualização das Câmeras

No módulo de visualização das câmeras (Figura 18) é possível acompanhar em tempo real as imagens das câmeras localizadas no espaço físico do IS (marcação vis-cam-5). Na marcação vis-cam-1 é possível escolher qual câmera será a principal da visualização, quando esta opção é cabível. Na marcação vis-cam-2 é permitido ao usuário escolher como as câmeras serão apresentadas. Para este trabalho, foram desenvolvidas três configurações, da esquerda para a direita, apenas uma câmera (Figura 19a), as quatro câmeras apresentadas de forma igual (Figura 19b) e as quatro câmeras, sendo uma principal e maior e as outras três menores abaixo (Figura 19c). Na Figura 19 estão mostradas as três formas de visualização possíveis atualmente. Na marcação vis-cam-3 é possível definir o FPS da visualização apresentada. Na marcação vis-cam-4 encontra-se uma configuração avançada que está relacionada às funcionalidades disponíveis pelo IS.

Figura 18 – Tela de visualização de câmeras (vis-cam).



Fonte: Produção do próprio autor.
Legenda: 1: vis-cam-1; 2: vis-cam-2; 3: vis-cam-3; 4: vis-cam-4; 5: vis-cam-5.

Figura 19 – As várias formas que a UI pode mostrar as imagens das câmeras. Da Esquerda para direita, apenas uma câmera (a), quatro câmeras (b) e quatro câmeras com destaque para uma (c).

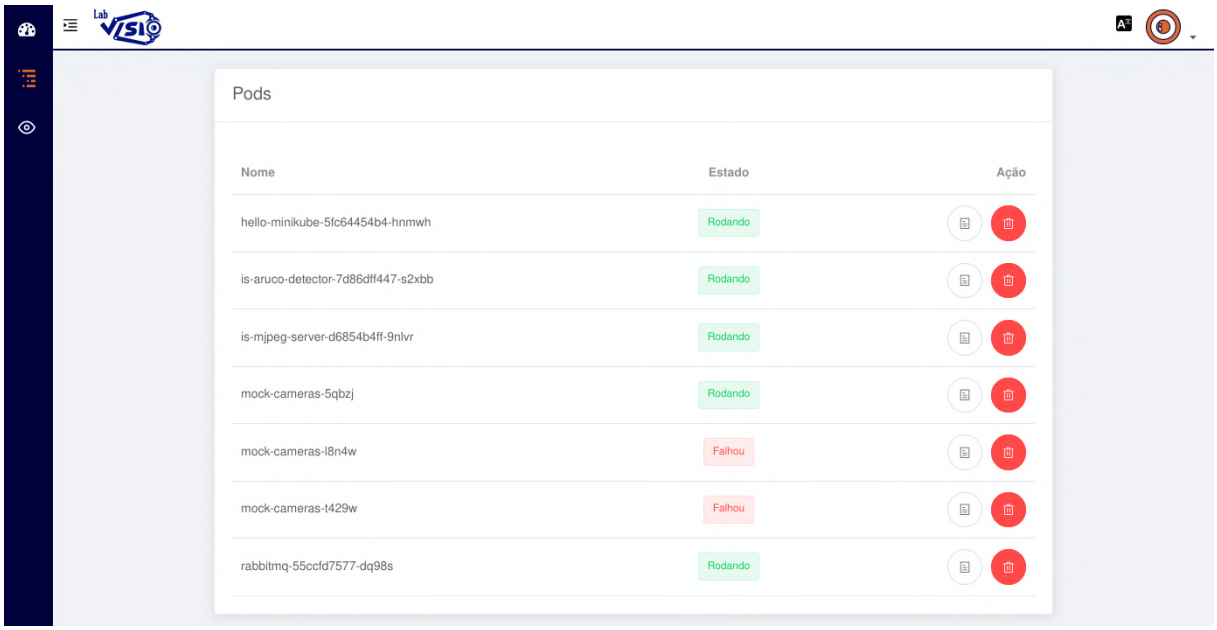
(a) (b) (c)

Fonte: Produção do próprio autor.

4.2.7 Lista de *Pods*

No módulo de lista de *pods* (Figura 20), encontra-se um bloco com uma lista de todos os *pods* do Kubernetes cujo *namespace* seja "default". Os detalhes de cada item desta lista será explicado mais a frente em conjunto com os detalhes dos itens da lista de serviços, pois na lista de serviços também possui itens da lista de *pods*.

Figura 20 – Tela de lista de *pods*.



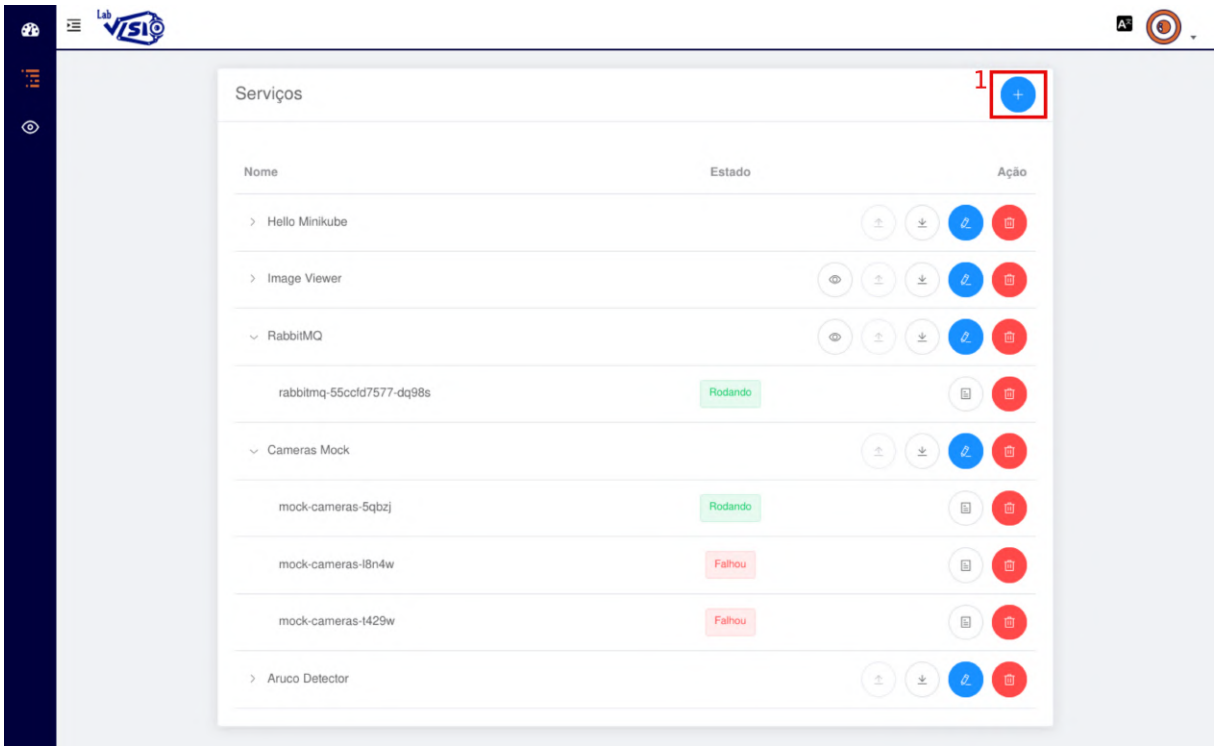
Fonte: Produção do próprio autor.

4.2.8 Lista de Serviços

No módulo de lista de serviços (Figura 21) está um bloco com uma lista em "árvore", onde os itens principais são serviços cadastrados no sistemas que podem ser expandidos para mostrar os *pods* relacionados a esses serviços. Na Figura 22 encontra-se um exemplo de um item dessa lista expandido. Na marcação lista-serviço-1 está o botão para a ação de cadastrar um novo serviço no sistema.

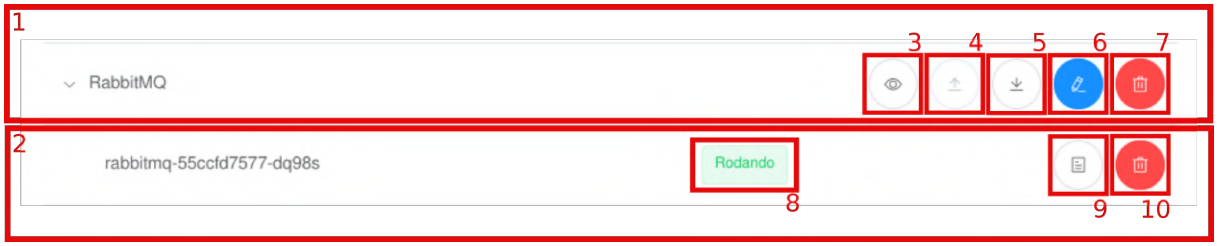
Cada item da lista de serviços e *pods* pode possuir ações relacionadas ao item listado. O item na marcação lista-item-1 é um serviço e representa o item básico da lista de serviços, podendo possuir até cinco ações relacionadas a ele. Pela ação na marcação lista-item-3 o usuário pode visualizar diretamente a saída do serviço quando disponível (Figura 24). Nas marcações lista-item-4 e lista-item-5 o usuário pode iniciar ou parar um serviço no IS. Com a ação na marcação lista-item-6 é possível alterar o cadastro do serviço e, com a ação na marcação lista-item-7, excluir o cadastro. O item na marcação lista-item-2 é um *pod* e representa o item item básico da lista de *pods* que pode ser encontrado expandido o item de serviço, quando cabível, na lista de serviços. O *pod* listado pode possuir até duas ações relacionadas a si mesmo. A primeira (marcação lista-item-9) mostra os *logs* do *pod* (Figura 25) e a segunda (marcação lista-item-10) possibilita ao usuário excluir o *pod* diretamente no Kubernetes. Na marcação lista-item-8 é mostrado o estado do *pod* dentro do ambiente Kubernetes.

Figura 21 – Tela de lista de serviços (lista-serviço).



Fonte: Produção do próprio autor.
Legenda: 1: lista-serviço-1.

Figura 22 – Detalhamento de um item expandido da lista de serviço (lista-item).

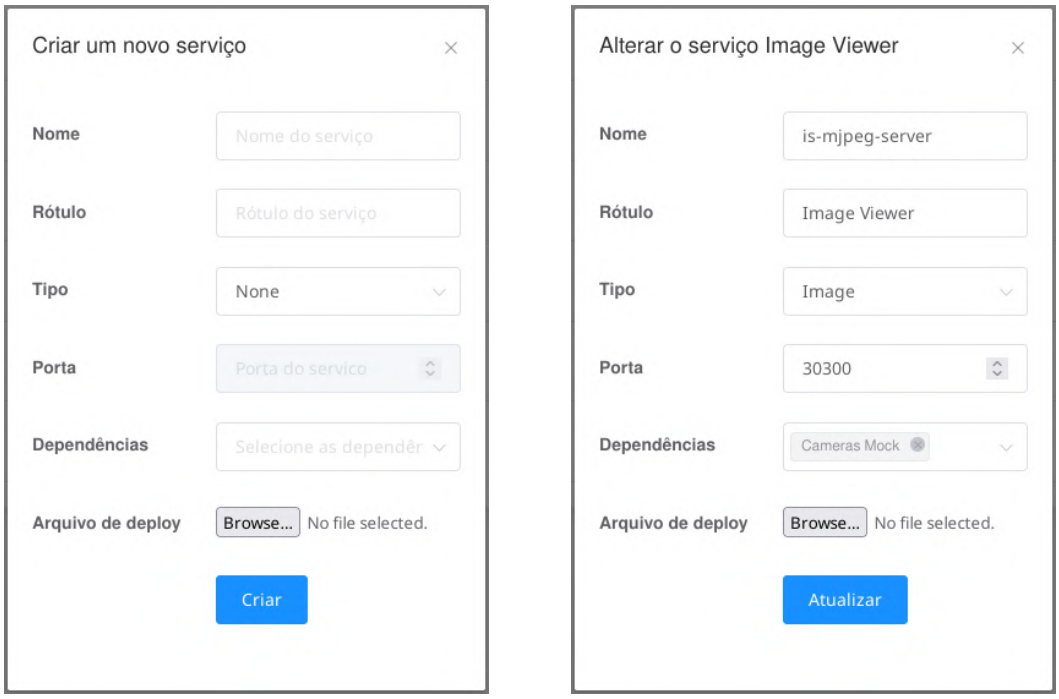


Fonte: Produção do próprio autor.
Legenda: 1: lista-item-1; 2: lista-item-2; 3: lista-item-3; 4: lista-item-4; 5: lista-item-5; 6: lista-item-6; 7: lista-item-7; 8: lista-item-8; 9: lista-item-9; 10: lista-item-10.

4.2.9 Cadastro de Serviços

De forma a abstrair alguns conceitos do Kubernetes e sustentar, ao mesmo tempo, alguns conceitos do IS foi criado o recurso "serviço". Para o correto funcionamento, esse recurso deve ser cadastrado no sistema através da ação na marcação lista-serviço-1, ou, por qualquer motivo, alterar as propriedades de qualquer serviço através da ação na marcação lista-item-6. Essas ações expõem ao usuário os formulários de criação (Figura 23a) e edição (Figura 23b) de serviços no sistema que podem ser encontrados na Figura 23.

Figura 23 – Formulários de criação (a) e edição (b) de cadastro de serviços.



(a) (b)

Fonte: Produção do próprio autor.

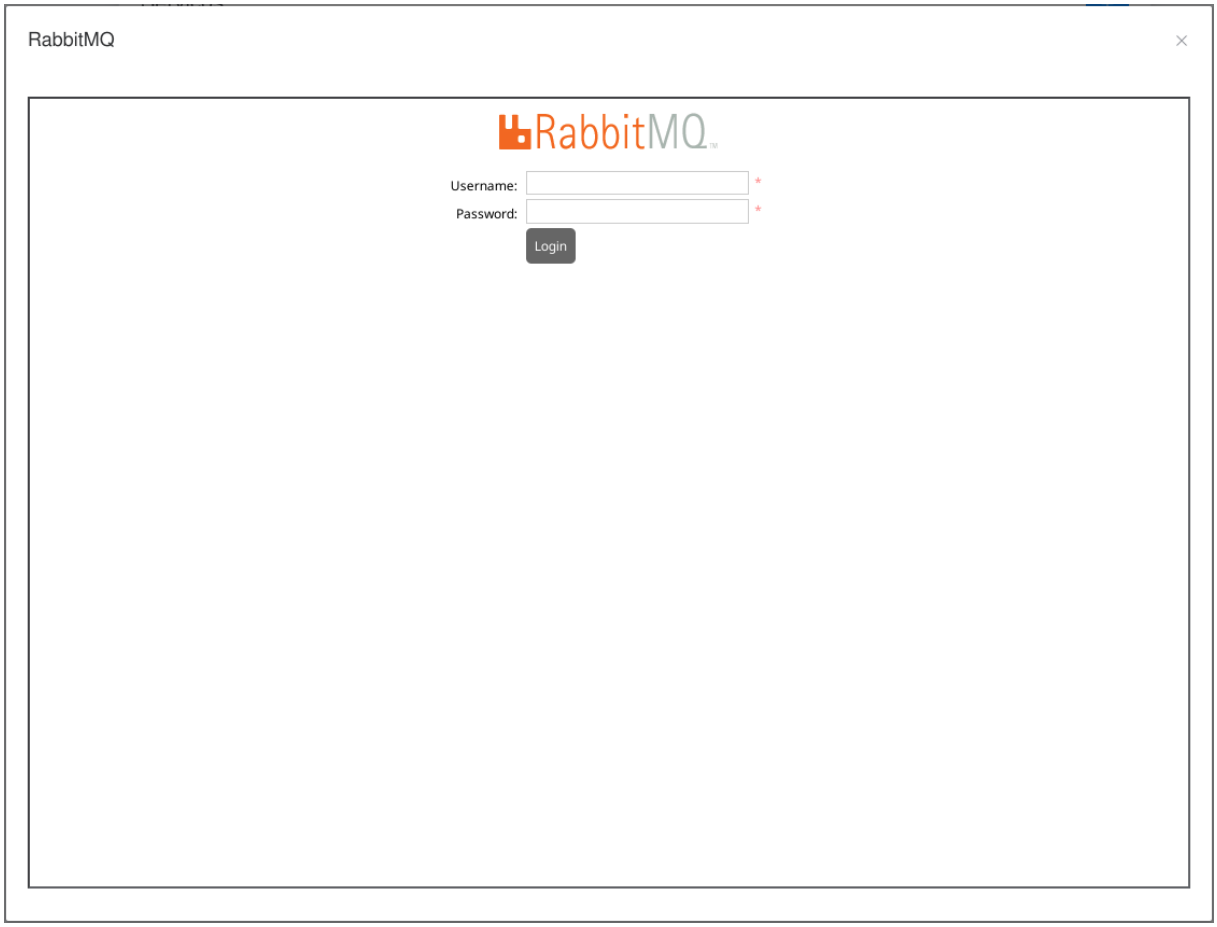
Este cadastro contém informações utilizadas por várias partes do sistemas, entre elas: o arquivo de *deploy* (*deployment*) em formato yaml que serve como aquivo de configuração dos recursos Kubernetes que o serviço estará ligado; o nome que tem que ser o mesmo *name* do recurso Kubernetes que está no arquivo de *deploy* para o sistema encontrar os *pods* relacionados; a porta de acesso, caso seu tipo seja "IMAGEM" ou "PAGE" que possibilita o acesso direto ao serviço através da ação na marcação lista-item-3; um rótulo para uma visualização mais "humana"; e uma lista de dependências onde é possível selecionar múltiplos serviços previamente cadastrados, dos quais o serviço sendo cadastrado atualmente depende, e que serão inicializados em conjunto na automatização.

O arquivo de *deploy* é utilizado pelas ações lista-item-4 e lista-item-5 para a correta inicialização e suspensão dos recursos Kubernetes relacionados ao serviço cadastrado. Esse arquivo fica persistido para que a informação dos recursos Kubernetes relacionados ao serviço seja independente do Kubernetes.

4.2.10 Ações Extras

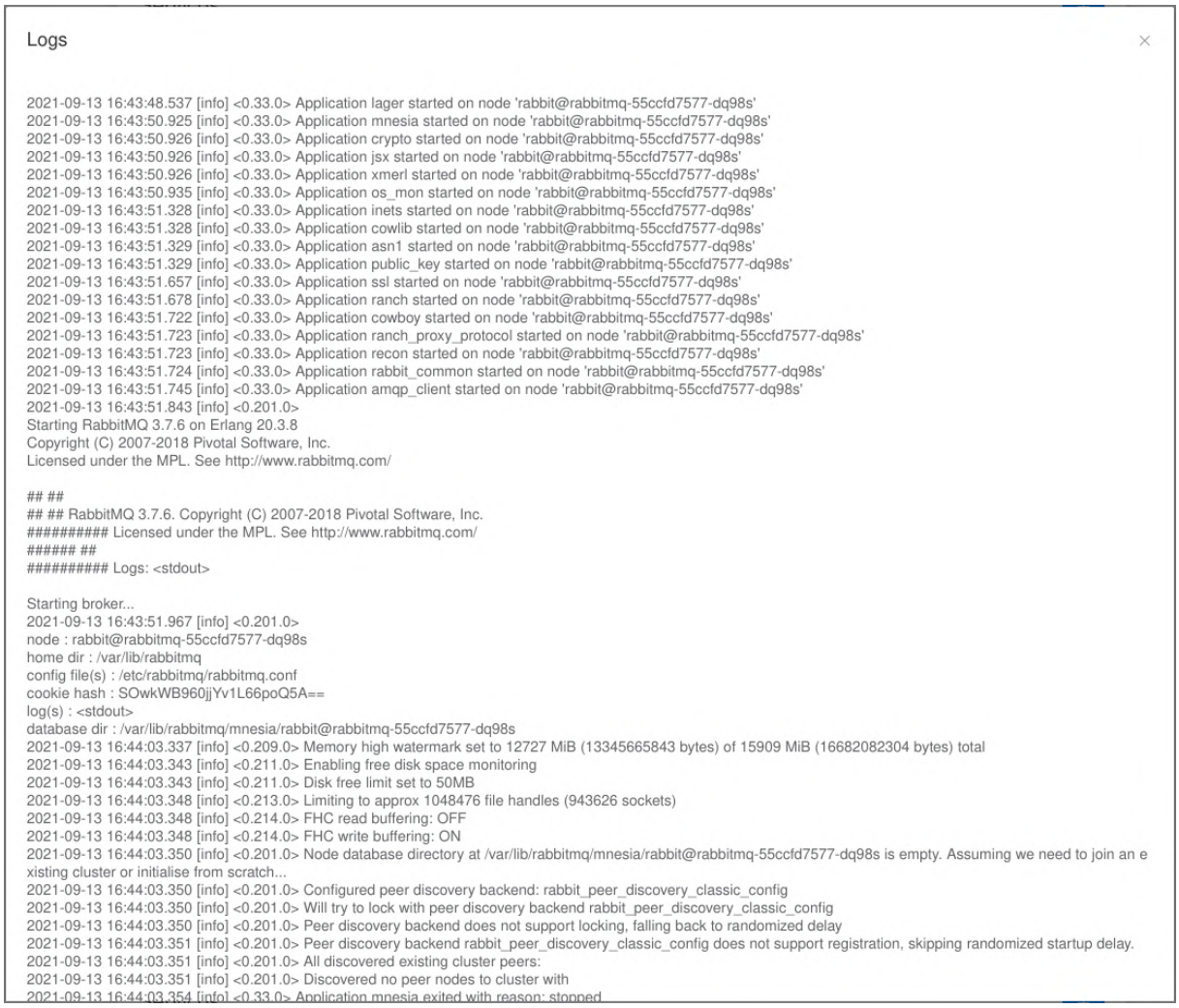
Nas Figuras 24 e 25 encontram-se, respectivamente, a visualização direta de um serviço, através da ação na marcação lista-item-3, e a visualização de *logs* de um *pod*, disponíveis pela ação na marcação lista-item-9.

Figura 24 – Visualização direta de um serviço.



Fonte: Produção do próprio autor.

Figura 25 – Visualização dos *logs* de um pod.

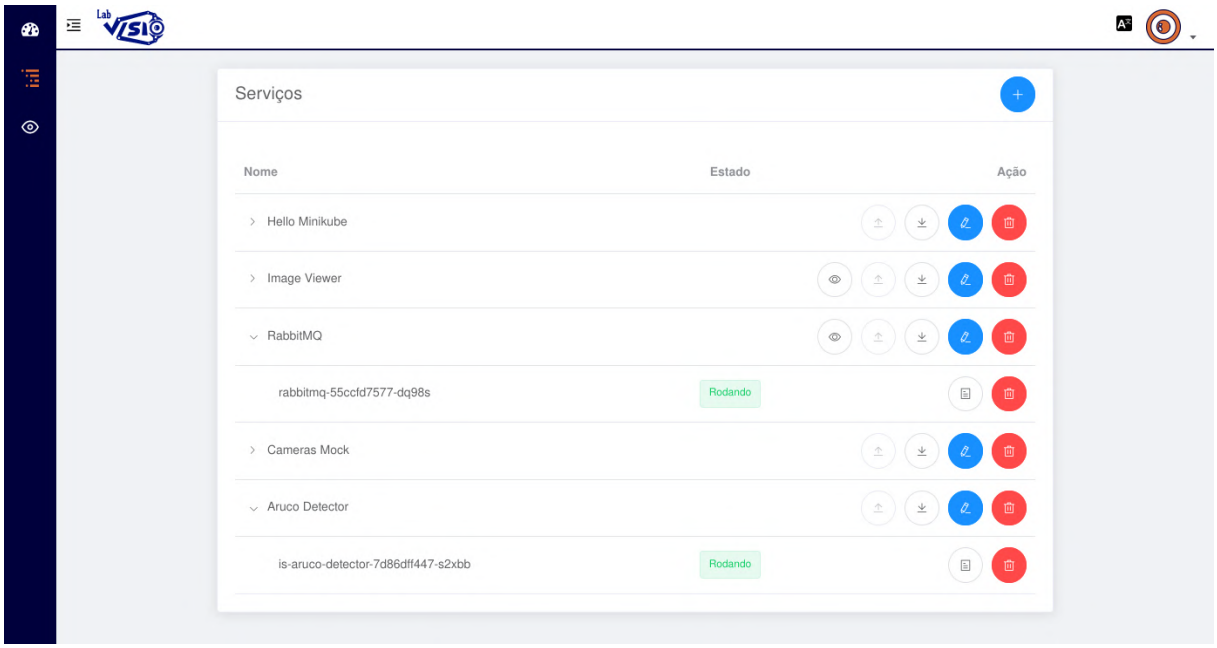


Fonte: Produção do próprio autor.

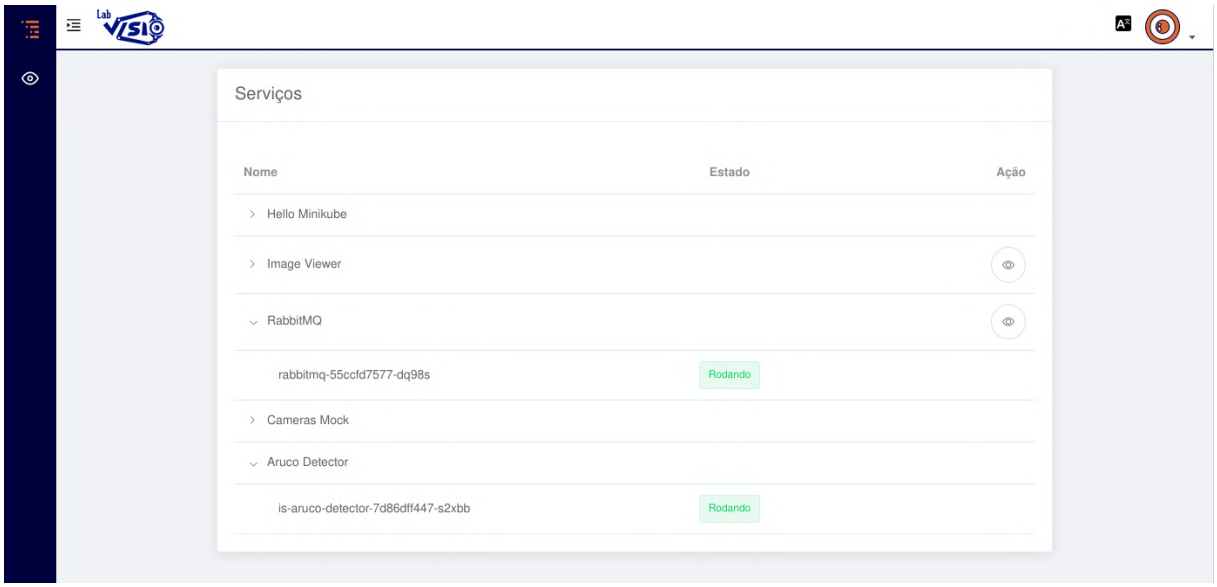
4.2.11 Papéis de Usuários

Ao desenvolver a autenticação de usuários também foi definido o controle de acesso por papéis. A Figura 26 mostra a tela de lista de serviços vista por dois tipos de usuários diferentes, um com papel de administrador (Figura 26a) e o outro com papel de usuário comum (Figura 26b).

Figura 26 – Exemplo de uma tela para usuários com diferentes papéis. Em (a) um usuário administrador e em (b) um usuário comum.



(a)



(b)

Fonte: Produção do próprio autor.

5 CONCLUSÃO E TRABALHOS FUTUROS

O objetivo principal deste trabalho foi desenvolver uma UI para a fácil utilização (visualização e gerenciamento) do IS. Tal objetivo se originou do contato com o sistema e sua interface atual que se mostrou complexa e confusa, principalmente para usuários leigos. Baseado em *frameworks* consolidados na comunidade, foi desenvolvido uma estrutura de base para se construir a UI, um módulo *backend* e um *frontend*.

As telas desenvolvidas para a UI, ao serem apresentadas para o Lab VISIO, mostraram-se satisfatórias, abrangendo aspectos gerais do Kubernetes, como o painel de controle, e aspectos específicos do IS, como a visualização de câmeras, além de uma interface mais limpa, simples e personalizada para o Lab VISIO.

Quando comparado a outros trabalhos na área de UI para Kubernetes, este trabalho mostra que ainda há muito espaço para melhorar em relação a funcionalidades (Kubernator) e a interatividade (Kosntellate), mas proporciona personalização profunda como a automatização, módulos para aplicações do IS e a aparência.

O próximo passo imediato é integrar o trabalho desenvolvido com o IS do Lab VISIO. Como futuro aperfeiçoamento, há a possibilidade de se aprimorar os módulos existentes, desenvolver outros módulos para a UI, de acordo com as aplicações do IS, e melhorar a UI de acordo com o *feedback* dos usuários.

Outro passo de interesse é a criação de *scripts* de testes, tanto para o *backend* quanto para o *frontend*, com o intuito de se identificar e aferir possíveis problemas no desenvolvimento do trabalho.

Como forma de aprofundar as funcionalidades deste trabalho em relação ao IS, pode-se realizar a automatização da desativação de serviços, levando em conta os serviços de que depende e outros serviços que também dependam destes. Além disso, seria interessante considerar a criação de um novo recurso, a "aplicação" que consiste num conjunto de serviços necessários para se ter um resultado final específico.

Finalmente, no intuito de manter o trabalho continuamente alinhado com as inovações, seria importante que, assim que o projeto Vue 3 estabilizar, a atual implementação fosse migrada do Vue 2 para o Vue 3 e, consequentemente, também do Element UI para o Element Plus ou outro *framework* em conformidade com o Vue 3.

REFERÊNCIAS

ALMONFREY, D.; CARMO, A. P. do; QUEIROZ, F. M. de; PICORETI, R.; VASSALLO, R. F.; SALLES, E. O. T. A flexible human detection service suitable for intelligent spaces based on a multi-camera network. International Journal of Distributed Sensor Networks, v. 14, n. 3, 2018. Disponível em: https://journals.sagepub.com/doi/full/10.1177/1550147718763550. Acesso em: 7 fev. 2020. Citado na página 14.

ATALAIA, A. VUE.js vs REACT: We Built an App on Both Frameworks. 2020. Disponível em: https://www.imaginarycloud.com/blog/vue-js-vs-react-an-app-on-both-frameworks/. Acesso em: 15 out. 2021. Citado na página 25.

BATISTA, E. J. S.; SILVA, L.; LEITE, C.; LIMA, A. Poredu: um ambiente de programação em blocos. In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação. [S.l.: s.n.], 2017. v. 6, n. 1, p. 144. Citado na página 19.

BISHOP, S.; FAIRBAIRN, M.; NORRISH, M.; SEWELL, P.; SMITH, M.; WANSBROUGH, K. Rigorous specification and conformance testing techniques for network protocols, as applied to tcp, udp, and sockets. In: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications. [S.l.: s.n.], 2005. p. 265–276. Citado na página 15.

BRUN, F.; MASSIMI, L.; FRATINI, M.; DREOSSI, D.; BILLÉ, F.; ACCARDO, A.; PUGLIESE, R.; CEDOLA, A. Syrmep tomo project: a graphical user interface for customizing ct reconstruction workflows. Advanced structural and chemical imaging, v. 3, n. 1, p. 1–9, 2017. Citado na página 18.

CARMO, A. P. do; QUEIROZ, F. M. de; SANTOS, C. C. dos; SILVA, L. de A.; VASSALLO, R. F. Uso de um espaço inteligente baseado em visão computacional para o controle de formação de robôs móveis. In: SBC. Anais do XII Simpósio Brasileiro de Computação Ubíqua e Pervasiva. [S.l.], 2020. p. 171–180. Citado na página 14.

CARMO, A. P. do; VASSALLO, R. F.; QUEIROZ, F. M. de; PICORETI, R.; FERNANDES, M. R.; GOMES, R. L.; MARTINELLO, M.; DOMINICINI, C. K.; GUIMARÃES, R.; GARCIA, A. S. et al. Programmable intelligent spaces for industry 4.0: Indoor visual localization driving attocell networks. Transactions on Emerging Telecommunications Technologies, Wiley Online Library, v. 30, n. 11, 2019. Citado na página 10.

CASILLAS-PEREZ, D.; MACIAS-GUARASA, J.; MARRON-ROMERA, M.; FUENTES-JIMENEZ, D.; FERNANDEZ-RINCON, A. Full body gesture recognition for human-machine interaction in intelligent spaces. In: SPRINGER. International Conference on Bioinformatics and Biomedical Engineering. [S.l.], 2016. p. 664–676. Citado na página 14.

CLOUD NATIVE COMPUTING FOUNDATION. Kubernetes. 2020. Disponível em: https://kubernetes.io/pt/. Acesso em: 15 out. 2020. Citado na página 11.

CLOUD NATIVE COMPUTING FOUNDATION. Install Tools. 2021. Disponível em: https://kubernetes.io/docs/tasks/tools/. Acesso em: 17 ago. 2021. Citado na página 24.

CLOUD NATIVE COMPUTING FOUNDATION. Usando Minikube para criar um cluster. 2021. Disponível em: https://kubernetes.io/pt-br/docs/tutorials/kubernetes-basics/create-cluster/cluster-intro/. Acesso em: 17 ago. 2021. Citado na página 24.

FERNANDES, M. R. Calibração de uma rede de câmeras e controle de um robô móvel em um Espaço Inteligente. Tese (Doutorado) — Universidade Federal do Espírito Santo, Centro Tecnológico, Vitória, 2014. Citado na página 13.

GALITZ, W. O. The essential guide to user interface design: an introduction to gui design principles and techniques. [S.l.]: John Wiley & Sons, 2007. Citado na página 17.

GITHUB INC. GitHub. 2020. Disponível em: https://github.com/. Acesso em: 15 out. 2020. Citado na página 10.

HUANG, Y.-C.; WU, K.-Y.; LIU, Y.-T. Future home design: an emotional communication channel approach to smart space. Personal and ubiquitous computing, Springer, v. 17, n. 6, p. 1281–1293, 2013. Citado na página 14.

LIU, B.; WANG, F.-Y.; GENG, J.; YAO, Q.; GAO, H.; ZHANG, B. Intelligent spaces: An overview. In: IEEE. 2007 IEEE International Conference on Vehicular Electronics and Safety. [S.l.], 2007. p. 1–6. Citado 2 vezes nas páginas 13 e 14.

MARTINEZ, W. L. Graphical user interfaces. Wiley Interdisciplinary Reviews: Computational Statistics, Wiley Online Library, v. 3, n. 2, p. 119–133, 2011. Citado 2 vezes nas páginas 9 e 10.

MASSE, M. REST API Design Rulebook: Designing consistent restful web service interfaces. [S.l.]: O'Reilly Media, Inc., 2011. Citado na página 16.

MUNOZ-SALINAS, R. Aruco: a minimal library for augmented reality applications based on opencv. Universidad de Córdoba, v. 386, 2012. Citado na página 23.

QUEIROZ, R. L.; SAMPAIO, F. F. Duinoblocks for kids: Um ambiente de programação em blocos para o ensino de conceitos básicos de programação a crianças do ensino fundamental i por meio da robótica educacional. In: SBC. Anais do XXIV Workshop sobre Educação em Computação. [S.l.], 2016. p. 91–100. Citado na página 18.

REDIS LTD. Redis. 2021. Disponível em: https://redis.io/topics/introduction. Acesso em: 15 ago. 2021. Citado na página 15.

SELEZNEV, S.; YAKOVLEV, V. Industrial application architecture iot and protocols amqp, mqtt, jms, rest, coap, xmpp, dds. International Journal of Open Information Technologies, v. 7, n. 5, p. 17–28, 2019. Citado na página 15.

SHELKE, S.; HARBOUR, J.; AKSANLI, B. Building an intelligent and efficient smart space to detect human behavior in common areas. In: IEEE. 2018 International Symposium on Networks, Computers and Communications (ISNCC). [S.l.], 2018. p. 1–6. Citado 2 vezes nas páginas 9 e 10.

SO, S. C.; SUN, H. Creating ambient intelligent space in downstream apparel supply chain with radio frequency identification technology from lean services perspective. International Journal of Services Sciences, Inderscience Publishers, v. 3, n. 2-3, p. 133–157, 2010. Citado na página 14.

STONE, D.; JARRETT, C.; WOODROFFE, M.; MINOCHA, S. User interface design and evaluation. [S.l.]: Elsevier, 2005. Citado na página 18.

TRAIAN, C. B.-S. S. C. O. Esquema das camadas da interface gráfica. 2013. Disponível em: https://commons.wikimedia.org/wiki/File:Esquema_das_camadas_da_interface_gr%C3%A1fica.svg. Acesso em: 15 ago. 2021. Citado na página 18.

VENKATARAMAN, A.; JAGADEESHA, K. K. Evaluation of inter-process communication mechanisms. Architecture, v. 86, p. 64, 2015. Citado na página 15.

WRIGHT, S.; STEVENTON, A. Intelligent spaces—the vision, the opportunities and the barriers. BT technology journal, Springer, v. 22, n. 3, p. 15–26, 2004. Citado 3 vezes nas páginas 14, 15 e 20.

WU, X.; YU, C.; SHI, Y. Multi-depth-camera sensing and interaction in smart space. In: IEEE. 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI). [S.l.], 2018. p. 718–725. Citado na página 9.

YATES, A.; BEAL, K.; KEENAN, S.; MCLAREN, W.; PIGNATELLI, M.; RITCHIE, G. R.; RUFFIER, M.; TAYLOR, K.; VULLO, A.; FLICEK, P. The ensembl rest api: Ensembl data for any language. Bioinformatics, Oxford University Press, v. 31, n. 1, p. 143–145, 2015. Citado na página 16.

ZIRKELBACH, C.; KRAUSE, A.; HASSELBRING, W. Modularization of research software for collaborative open source development. arXiv preprint arXiv:1907.05663, 2019. Citado na página 22.