UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

OTÁVIO BAZIEWICZ FILHO

SISTEMA BASEADO EM INFRAESTRUTURA DE DESKTOPS VIRTUAIS EM NUVEM PÚBLICA PARA ENSINO E PESQUISA

CURITIBA

2024

OTÁVIO BAZIEWICZ FILHO

SISTEMA BASEADO EM INFRAESTRUTURA DE DESKTOPS VIRTUAIS EM NUVEM PÚBLICA PARA ENSINO E PESQUISA

A system based on public cloud virtual desktop infrastructure for education and research

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Sistemas De Informação do Curso de Bacharelado em Sistemas De Informação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Me. Wilson Horstmeyer Bogado

CURITIBA

2024

OTÁVIO BAZIEWICZ FILHO

SISTEMA BASEADO EM INFRAESTRUTURA DE DESKTOPS VIRTUAIS EM NUVEM PÚBLICA PARA ENSINO E PESQUISA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Sistemas De Informação do Curso de Bacharelado em Sistemas De Informação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 27/junho/2024

Gustavo Alberto Giménez Lugo
Doutorado
Universidade Tecnológica Federal do Paraná

Mauro Sergio Pereira Fonseca
Doutorado
Universidade Tecnológica Federal do Paraná

Wilson Horstmeyer Bogado
Mestrado
Universidade Tecnológica Federal do Paraná

CURITIBA

2024

RESUMO

O acesso a recursos computacionais de alto desempenho é essencial para a realização de pesquisas acadêmicas e científicas, porém, a aquisição e manutenção de *desktops* de alto desempenho é um desafio para instituições públicas de ensino superior devido a existência de processos licitatórios complexos e demorados. Por outro lado, universitários advindos de famílias de baixa renda, que não possuem condições financeiras para adquirir um computador pessoal, podem ter dificuldades para realizar atividades acadêmicas dentro e fora da universidade. Sendo assim, o objetivo deste trabalho é desenvolver um sistema de Infraestrutura de Desktops Virtuais, do inglês *Virtual Desktop Infrastructure* (VDI) capaz de possibilitar que alunos e pesquisadores de instituições de ensino superior acessem *desktops* virtuais alocados em nuvem pública através de qualquer dispositivo com acesso à internet e com suporte a um navegador web, aprimorando a gestão dos recursos computacionais mais potentes, uma vez que são consumidos como serviço ao invés de serem adquiridos através de processos licitatórios complexos e demorados.

Palavras-chave: computação em núvem; infraestrutura virtual de desktops; virtualização; conexão remota; ensino e pesquisa.

ABSTRACT

Access to high-performance computing resources is essential for conducting academic and scientific research. However, acquiring and maintaining high-performance *desktops* is a challenge for public higher education institutions due to the existence of complex and lengthy bidding processes. On the other hand, university students from low-income families, who do not have the financial means to purchase a personal computer, may have difficulties carrying out academic activities both inside and outside the university. Therefore, the objective of this work is to develop a VDI system capable of enabling students and researchers from higher education institutions to access virtual *desktops* allocated in the public cloud through any device with internet access and web browser support, improving the management of the most powerful computing resources, as they are consumed as a service instead of being acquired through complex and lengthy bidding processes.

Keywords: cloud computing; infraestrutura virtual de desktops; virtualization; remote connection; learning and research.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTAGEM DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	Interface de Programação de Aplicações, do inglês *Application Programming Interface*
AWS	*Amazon Web Services*
EC2	*Elastic Compute Cloud*
IaC	Infraestrutura como Código, do inglês *Infrastructure as Code*
N/A	Não se Aplica
RAM	*Random Access Memory*
RDP	*Remote Desktop Protocol*
SDK	Kit de Desenvolvimento de Software, do inglês *Software Development Kit*
SSH	*Secure Shell*
UADM	Usuário Administrador
UCOM	Usuário Comum
UPEN	Usuário Pendente
UTFPR	Universidade Tecnológica Federal do Paraná
VDI	Infraestrutura de Desktops Virtuais, do inglês *Virtual Desktop Infrastructure*
VNC	*Virtual Network Computing*

SUMÁRIO

1 INTRODUÇÃO

As origens do termo Computação em nuvem datam do início dos anos 1990, quando componentes desconhecidos em diagramas de redes de computadores eram representados por nuvens. Mesmo não significando exatamente o que é hoje, a ideia de que a nuvem representava algo que estava fora das fronteiras de uma rede local já estava presente. Antes de ganhar o nome mais atual, a tecnologia chegou a ser chamada de *Utility Computing*, fazendo alusão ao acesso compartilhado e de forma simultânea a computadores *mainframe*, que ocupavam muitas vezes salas completas e custavam caro. O termo só chegou formalmente ao mercado em 2006, quando a Amazon lançou o serviço de computação em nuvem *Elastic Compute Cloud* (EC2). (BHOWMIK, 2017)

A computação em nuvem atualmente já aparece como pilar de soluções tecnológicas em situações de escalas variadas. Por um lado, uma grande empresa pode utilizá-la para prover a infraestrutura de grandes aplicações de operação crítica, e por outro lado, um desenvolvedor pode executar cargas de trabalhos em uma instância remota, desativá-la após concluir e pagar apenas pelo tempo de uso do recurso. (TAURION, 2009)

Infraestrutura de Desktops Virtuais, do inglês *Virtual Desktop Infrastructure* é o termo usado para referenciar um ambiente de infraestrutura que hospeda *desktops* virtuais e provê para usuários finais a medida que são requisitados. Tanto ambientes *On-Premise*, onde os servidores estão localizados na infraestrutura local, quanto em nuvem utilizam a mesma base conceitual de virtualização para tal arquitetura, a grande diferença está na forma em que os recursos são precificados. Em servidores locais, o recurso deve ser adquirido previamente, gerenciado e protegido, já em ambientes de nuvem, onde o modelo de responsabilidade compartilhada exime o usuário de responsabilidades de segurança e gerenciamento físicos dos recursos, o preço é pago pela utilização dos recursos. (BHOWMIK, 2017)

Embora tenham sido feitos progressos na utilização de VDI em nuvem e no estudo da influência dos fatores de qualidade de serviço em sistemas de ensino baseados em VDI (NAKAZAWA; KOIZUMI; HIRASAWA, 2012), poucos estudos foram realizados a fim de apresentar uma infraestrutura completa de aplicação que sustente *desktops* virtuais em nuvem pública para utilização em ensino e pesquisa no contexto de instituições públicas de ensino superior.

O objetivo deste trabalho é o desenvolvimento de um sistema de VDI em nuvem pública que permita o acesso a *desktops* virtuais a partir de qualquer dispositivo com um navegador de internet instalado, promovendo o acesso a recursos computacionais adequadamente robustos para a execução de atividades acadêmicas e de pesquisa, a partir de qualquer lugar, dentro ou fora da universidade. Além disso, viabilizar o consumo dos mesmos recursos computacionais como serviço, reduzindo custos com aquisição e manutenção de equipamentos e garantindo a disponibilidade de recursos computacionais para alunos e pesquisadores.

O acesso ao sistema baseado em papéis, permitirá que diferentes combinações de sistemas operacionais, *softwares* e especificações de *hardware* sejam disponibilizados para públicos

com necessidades específicas, como alunos de diferentes cursos e pesquisadores de diferentes áreas de atuação.

Através de uma interface *web* principal, os professores poderão criar configurações de *desktops* virtuais contendo todos os requisitos necessários para a execução de atividades acadêmicas e de pesquisa e disponibilizá-los para os alunos. Estes, por sua vez, poderão utilizar as configurações disponibilizadas para criar suas instâncias e escolher entre diferentes tipos de *hardware* previamente habilitados pelos professores.

O sistema também lidará com a persistência de dados entre sessões, desligamento preemptivo de recursos ociosos e também com todo o ciclo de vida dos recursos de infraestrutura, garantindo que os recursos sejam desalocados após o uso e que novos recursos sejam alocados conforme a demanda, de forma rápida e escalável.

1.1 Objetivos

A seguir, o objetivo geral e os objetivos específicos do trabalho são apresentados. O objetivo geral é o propósito principal do trabalho, que através dos objetivos específicos é desdobrado em partes menores que enriquecem o escopo do trabalho.

1.1.1 Objetivo geral

Desenvolver um sistema de VDI em nuvem pública capaz de prover acesso a *desktops* virtuais para ensino e pesquisa a partir de qualquer dispositivo com um navegador de internet instalado.

1.1.2 Objetivos específicos

- Modelar a infraestrutura do sistema de VDI, utilizando serviços de computação em nuvem pública, garantindo escalabilidade, alta disponibilidade e segurança dos dados.
- Modelar o sistema utilizando práticas de desenvolvimento de *software* que garantam a extensibilidade do sistema validada através da cobertura de testes unitários dos casos de uso.
- Implementar a Infraestrutura como Código, do inglês *Infrastructure as Code* (IaC) para todos os componentes do sistema, permitindo a implantação completa em novos ambientes de forma automatizada.
- Implementar um sistema de configuração para que sejam possível conectar um provedor de identidade e autenticação ao sistema no momento da implantação.

- Apresentar uma estimativa de custo de utilização do sistema contemplando diferentes cenários de utilização.
- Desenvolver uma documentação externa, que aborde tanto a configuração do sistema em novos ambientes quanto o uso do sistema através da interface *web* principal ou da API que o sistema disponibiliza.

1.2 Justificativa

Promover o acesso a recursos computacionais de forma remota, a partir de qualquer dispositivo com acesso à internet e que tenha um navegador *web*, de forma que os alunos e pesquisadores possam acessar recursos computacionais de qualquer lugar, através de dispositivos com capacidade computacional limitada.

1.3 Motivação

Em 2021, o censo de educação superior mostrou que cerca de 3,9 milhões de estudantes ingressaram em instituições de ensino superior e que 77,7% destes concluíram o ensino médio na rede pública de ensino. (INEP, 2021) Para aqueles que não possuem computadores de uso próprio, ainda é um desafio lidar com demandas acadêmicas que exigem o acesso a esse tipo de equipamento dentro e fora da universidade.

No contexto da Universidade Tecnológica Federal do Paraná (UTFPR), os alunos têm a possibilidade de utilizar os equipamentos em aulas de laboratório ou na biblioteca. Porém a pandemia de COVID-19 evidenciou que essas alternativas ainda não são suficientes para promover o acesso igualitário entre os estudantes. Este cenário é ainda mais restrito para pesquisadores que necessitam de recursos computacionais mais robustos para executar suas atividades.

Não menos importante, o trabalho também visa possibilidade de redução de custos com aquisição de equipamentos para universidades públicas, como também elucida uma abordagem de consumo de recursos computacionais como serviço, onde os recursos robustos são utilizados pelo tempo necessário, sem a necessidade de aquisição, manutenção e tratativas relacionadas à depreciação de equipamentos.

1.4 Estrutura do trabalho

Este trabalho está organizado em cinco capítulos, sendo este o primeiro deles, a introdução. Seguido do Capítulo 2 que apresenta a revisão da literatura, o Capítulo 3 que apresenta a metodologia, o Capítulo 4 que apresenta os resultados obtidos e por fim o Capítulo 5 que apresenta as conclusões obtidas com o desenvolvimento do trabalho e discute sobre perspectivas futuras para o trabalho.

2 REVISÃO DA LITERATURA

Esse capítulo apresenta os principais conceitos envolvidos na construção desse trabalho, a fim de contextualizar as bases teóricas utilizadas como referência para o desenvolvimento. O capítulo está organizado em duas seções, na primeira com aspectos do referencial teórico e na segunda com a apresentação de trabalhos relacionados.

2.1 Computação em nuvem

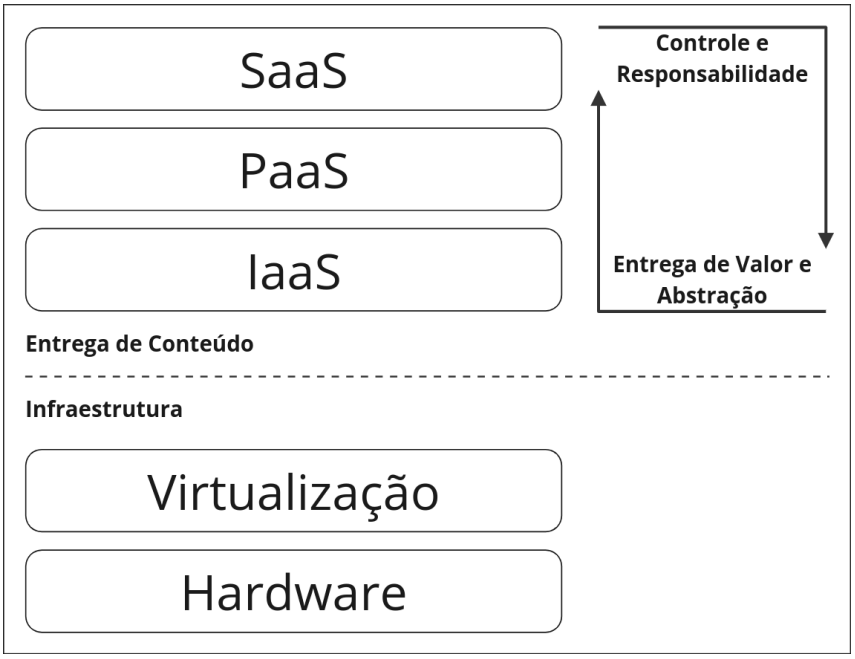
A computação em nuvem é um termo que se refere ao modelo de de entrega de recursos computacionais, baseado em uma grande rede de servidores físicos ou virtuais. Recursos como capacidade computacional ou capacidade de armazenamento são virtualizados em *data centers* e consumidos sob demanda remotamente. Segundo Taurion (2009), as principais características são:

- A computação em nuvem transmite uma sensação de disponibilidade infinita de recursos.
- Com esse tipo de modelo de serviços computacionais, não é necessário adquirir equipamentos físicos para poder utilizá-los. Os mesmo são acessáveis como serviços.
- Os recursos podem ser utilizados na proporção que são necessários para determinado cenário, podendo redimensionar a capacidade computacional de forma dinâmica.
- O modelo de cobrança em ambientes de nuvem é baseado na quantidade de uso dos recursos.

Os serviços de computação em nuvem podem ser classificados de acordo com o modelo de entrega de recursos, bem como o nível de responsabilidade que o usuário tem sobre a infraestrutura que suporta os serviços. Entre os modelos de entrega de recursos, os mais comuns são: *Software as a Service* (SaaS), *Platform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS). (BHOWMIK, 2017)

A Figura 1 apresenta um diagrama que ilustra a hierarquia de responsabilidades entre os modelos de entrega de recursos. O diagrama mostra que o *SaaS* é o modelo que demanda menos responsabilidade do cliente, mas também entrega mais valor e abstração. O *IaaS* é o modelo que demanda mais responsabilidade do cliente, mas também entrega mais controle sobre a infraestrutura (BHOWMIK, 2017).

Figura 1 – Modelo de entrega de conteúdo em computação em nuvem



Fonte: Adaptado de Bhowmik (2017).

Figura 2 – Responsabilidades nos modelos de entrega de conteúdo em computação em nuvem

Fonte: Adaptado de Bhowmik (2017).

A Figura 2 apresenta um diagrama que ilustra as responsabilidades de cada modelo de entrega de recursos. É possível observar que no modelo de entrega *SaaS* o cliente acessa diretamente uma aplicação e o provedor é inteiramente responsável pelo gerenciamento e configuração da infraestrutura que o suporta. No *PaaS* o cliente é capaz de executar e gerenciar aplicações sem a necessidade de se preocupar com a infraestrutura que suporta as aplicações e no *IaaS* o cliente tem controle total sobre: Dados, Aplicações, Runtimes, Middleware e Sistema Operacional. (BHOWMIK, 2017)

Um exemplo de serviço gerenciável é o *Elastic Beanstalk*, que é categorizado como *Paas*. Com ele é possível executar aplicações sem a necessidade direta de configurar a maior parte dos serviços de infraestrutura disponíveis. O desenvolvedor precisa se preocupar somente com o código da aplicação.

Esse tipo de serviço é útil para soluções comuns como APIs que necessitam na maioria dos casos dos mesmos componentes de infraestrutura como: Um servidor de aplicação, configuração de rede, balanceador de carga, banco de dados e um local para armazenamento de objetos.

A *AWS* opera seus recursos a partir de um modelo de responsabilidade compartilhada que visa reduzir a responsabilidade operacional dos clientes sobre recursos. (AWS, 2023) E seguindo o modelo, a sua responsabilidade é garantir a segurança **DA** nuvem. Isso significa que a provedora deve proteger todos os aspectos de infraestrutura que executa os serviços disponibilizados.

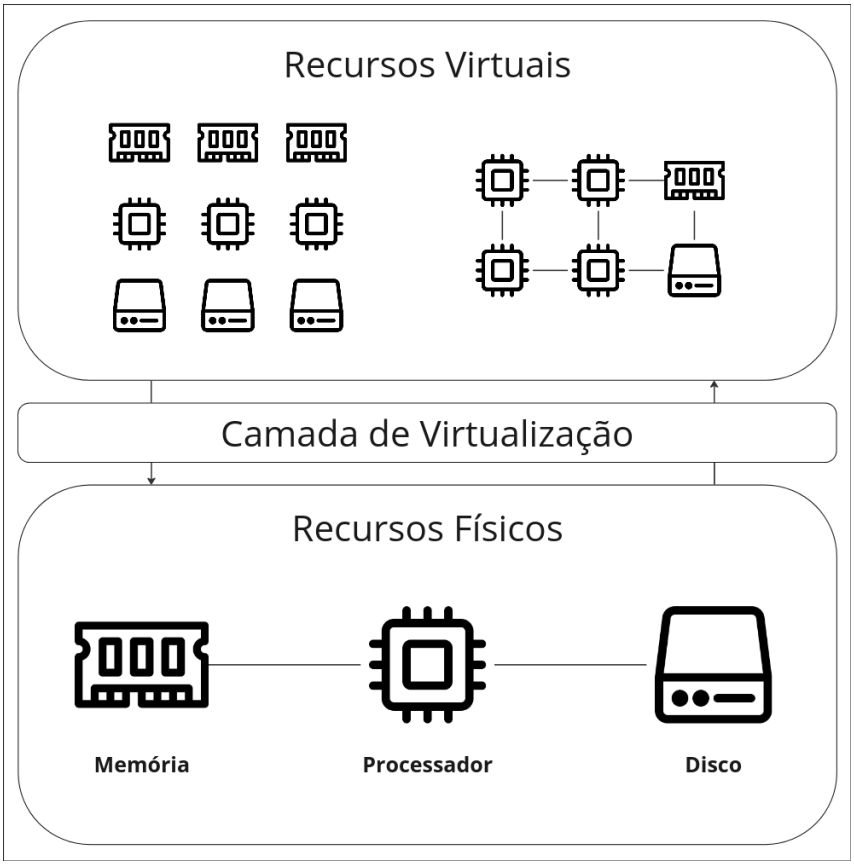
Para o cliente, a responsabilidade é de garantir a segurança **NA** nuvem, levando em consideração todas as boas praticas de desenvolvimento para que os recursos sejam utilizados de maneira responsável e segura.

2.2 Virtualização

A virtualização é uma técnica que permite a representação de recursos computacionais físicos através de uma camada de software que permite a utilização de recursos de maneira customizada e independente para cada necessidade, diferente da abordagem tradicional onde os recursos físicos são acessados diretamente. (BHOWMIK, 2017)

Qualquer tipo de recurso computacional pode ser virtualizado, como processadores, memória, armazenamento e redes. Partindo de um modelo estrutural onde os recursos físicos são interligados e compartilhados, a virtualização permite a criação de ambientes isolados e independentes, onde cada recurso é acessado de maneira exclusiva. Ainda, a técnica de virtualização permite que um recurso físico seja representado através de múltiplos recursos virtuais, como mostra a Figura 3, viabilizando assim o modelo de Computação em Nuvem. (BHOWMIK, 2017)

Figura 3 – Transformação de recursos físicos em recursos virtuais através da virtualização



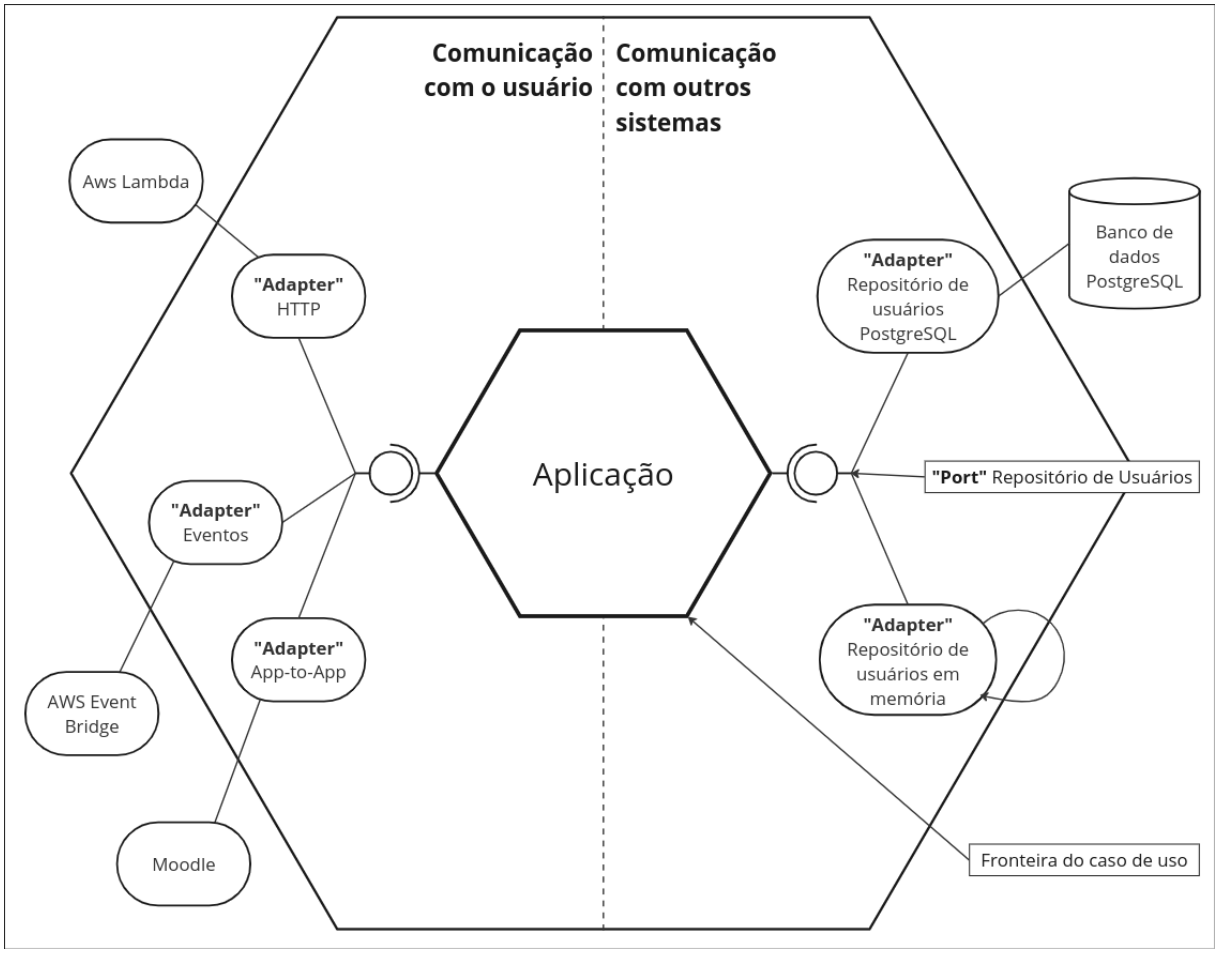
Fonte: Adaptado de Bhowmik (2017).

2.3 Arquitetura hexagonal

Arquitetura hexagonal é um padrão de arquitetura de *software* proposto por Alistair Cockburn em 2005, que ajuda a solucionar problemas de testabilidade e acoplamento de componentes em sistemas de *software*. Quando um *software* é projetado, muitas vezes é difícil projetar componentes genéricos o suficiente para que não dependam de um determinado *framework* ou tecnologia. Isso é problemático porque a manutenção do *software* se torna mais difícil e a escolha de novas tecnologias pode ser limitada, além de tornar o processo de testes complexo e custoso. Para resolver esses problemas, a arquitetura hexagonal propõe uma abordagem onde o núcleo da aplicação é responsável por orquestrar todas as operações de entrada e saída de dados através de adaptadores. (COCKBURN, 2017).

Como pode ser visto na Figura 4, o padrão é representado por um hexágono com a aplicação no centro. As arestas do hexágono delimitam as fronteiras dos casos de uso da aplicação e também representam os *Ports*, que são as interfaces as interfaces de comunicação entre a camada de aplicação e a camada de interfaces (COCKBURN, 2017).

Figura 4 – Arquitetura hexagonal



Fonte: Adaptado de Cockburn (2017) e Amazon Web Services (2024s).

Os *adapters* são classes que implementam as interfaces definidas pelos *Ports*, neles a lógica de comunicação com quaisquer componente externo é implementada. Dessa forma, a aplicação não precisa saber de detalhes técnicos de cada componente externo, como por exemplo, a forma de comunicação com um banco de dados ou a forma de comunicação com um serviço de mensageria. (COCKBURN, 2017).

No momento da criação de testes unitários para os casos de uso da aplicação, os *Adapters* que utilizam comunicação externa são substituídos por outros *Adapters* que simulam a comunicação externa. Isso permite que os testes sejam executados de maneira independente e sem a necessidade de outros serviços externos. Na imagem Figura 4, existe um exemplo de *Port* para armazenamento de informações de usuários, bem como dois *Adapters* que implementam a interface do *Port*. Um dos *Adapters* é responsável por armazenar as informações em um banco de dados e o outro é responsável por armazenar as informações em memória. Ambos são utilizados dentro do ciclo de desenvolvimento, mas em momentos diferentes, sendo um para testes e outro para a execução real da aplicação.

No desenvolvimento prático utilizando *Ports and Adapters*, padrões como o *Domain Driven Design* o *SOLID* e a injeção de dependências são comumente utilizados em conjunto (COCKBURN, 2017).

2.4 Trabalhos relacionados

O trabalho desenvolvido por Nakazawa, Koizumi e Hirasawa (2012) realiza uma análise quantitativa do impacto de parâmetros de conexão de rede em ambientes de ensino remoto sustentados por uma estrutura de VDI. A análise é feita a partir de um experimento com o objetivo de clarificar a relação entre a qualidade de conexão com a internet e a usabilidade de sistemas de ensino remoto.

O resultado do experimento mostrou que é possível realizar atividades como escrita, desenhos e consumo de mídias, sem grandes prejuízos para a usabilidade, mas é necessário que a conexão com a internet tenha qualidade. Outro ponto importante é que fora atividades de visualização de vídeos, a largura de banda não é um fator de grande consumo nesses tipos de ambientes.

Outro trabalho de grande relevância para essa produção, desenvolvido por Scandura (2022), apresenta um sistema de VDI para educação remota utilizando as mesmas tecnologias propostas no presente trabalho.

A proposta da produção é de criar uma VDI de baixo custo com *Elastic Compute Cloud* (EC2) *Spot Instances* da AWS e com o Apache Guacamole™ para gerenciamento das conexões, e permitindo o acesso de qualquer dispositivo com um navegador de internet, até mesmo através televisões com acesso à internet, como apresentado no trabalho.

A solução proposta por Scandura (2022) aborda um cenário de aula de laboratório, onde um professor pode agendar um evento de criação das máquinas virtuais para os alunos e no horário da aula, todos poderiam acessar o recurso reservado para a aula.

Em termos de custos operacionais, o trabalho apresenta uma estimativa de custo por usuário de USD$ 0.87 mensais com a utilização dos seguintes parâmetros:

- Tempo de conexão diária: 18 horas
- Categoria da instância: t3.micro
- Sistema operacional: *Microsoft Windows Server 2019*
- Memória: 4 GB de *Random Access Memory* (RAM)
- Espaço em disco: 90 GB
- Região: São Paulo

Em termos de preço pela transferência de dados, o trabalho relata um custo de USD$ 0.06 por hora em picos onde o uso de banda chaga a 2 MB/s. O custo dos outros serviços que suportam a infraestrutura, foi estimado em USD$ 200.00 mensais.

O trabalho feito por Scandura (2022) apresenta um panorama muito promissor e consoante aos objetivos do presente trabalho, demostrando que é possível executar a solução em um cenário de laboratório. Mas ainda não apresenta resultados referentes ao gerenciamento de *desktops* que tenham os dados persistidos, para outros cenários de uso, como o de pesquisa por exemplo.

3 METODOLOGIA

Este capítulo aborda as metodologias de modelagem e desenvolvimento utilizadas no trabalho.

3.1 Escopo do sistema

O sistema *Virtual Lab* possibilita a criação e gerenciamento de instâncias de máquinas virtuais para uso em laboratórios remotos de ensino e pesquisa. O acesso é feito por meio de um navegador de internet, permitindo a utilização de qualquer dispositivo com um navegador de internet com acesso à rede.

O sistema acomoda dois tipos de usuários: usuários comuns e administradores. Os usuários comuns têm permissão de criar instâncias de máquinas virtuais com base em templates pré-definidos, gerenciar suas próprias instâncias, bem como o conteúdo instalado nas mesmas. Os administradores têm as mesmas permissões dos usuários comuns, além de gerenciar os templates de instância, controlar o acesso dos usuários comuns ao sistema e aos recursos de *hardware*.

O sistema permite o acesso aos usuários através de autenticação por meio de usuário e senha, bem como através de um provedor de identidade externo, caso o ambiente de uso do sistema já possua um provedor de identidade.

Inicialmente, um usuário administrador cria os templates de instância indicando o sistema operacional e a quantidade de armazenamento disponível. Os usuários comuns então criam instâncias com base nesses templates, escolhendo entre os tipos de *hardware* permitidos.

Ao ser criada, a instância passa por um processo de configuração inicial para garantir que o sistema operacional esteja pronto para uso, e então ser disponibilizada para o usuário, que pode acessá-la através da página de conexões do sistema. As instâncias são encerradas automaticamente após um período de inatividade.

3.2 Modelagem

O processo de modelagem do sistema partiu de alguns princípios elementares para a construção de um sistema de *software*:

- **Simplicidade**: Criação apenas de modelos que sejam necessários para a compreensão do sistema. Modelos em excesso tornam o processo de desenvolvimento mais complexo e custoso.
- **Extensibilidade**: Os modelos levam em consideração a evolução constante do sistema, evitando restrições que impliquem em grandes refatorações estruturais.

- **Coesão entre os modelos e as tecnologias existentes**: Na concepção dos componentes, é importante considerar a arquitetura do sistema como um todo, de forma que os componentes sejam coesos e acoplados de forma apropriada. Esse princípio também é relevante quando pensamos na utilização de serviços de computação em nuvem, que possuem características próprias de arquitetura, infraestrutura, segurança e modelo de cobrança.
- **Consistência das informações em todos os níveis**: As informações que são transmitidas entre os componentes do sistema são verificadas e validadas em todos os níveis, garantindo que a integridade do sistema seja mantida.

O processo de modelagem do sistema foi dividido nas seguintes etapas: levantamento de requisitos, projeto de arquitetura e projeto de componentes (PRESSMAN ROGER S.; MAXIM, 2016).

3.2.1 Levantamento de requisitos

Inicialmente, os requisitos do sistema foram identificados a partir de discussões recorrentes com o orientador do trabalho, que possui experiência em desenvolvimento de sistemas de *software* e em ambientes de ensino e pesquisa. Além disso, é um dos responsáveis pelo gerenciamento dos recursos de infraestrutura da UTFPR servidos em nuvem publica através da AWS.

Os requisitos foram divididos em funcionais e não funcionais, e como uma forma de orientar o desenvolvimento, foram criados casos de uso para representar as interações entre os usuários e o sistema.

As Tabelas 1 e 2 apresentam os requisitos funcionais e não funcionais do sistema, respectivamente.

Tabela 1 – Requisitos funcionais do sistema

Identificador	Descrição
RF01	Deve haver três tipos de permissões de usuário no sistema: Usuário Pendente (UPEN), Usuário Comum (UCOM) e Usuário Administrador (UADM)
RF02	O usuário deve ser capaz de registrar-se no sistema utilizando um e-mail válido, senha e nome de usuário
RF03	O usuário deve ser capaz de autenticar-se no sistema utilizando um e-mail válido e senha
RF04	O usuário deve ser capaz de autenticar-se no sistema utilizando um provedor de identidade externo
RF05	O usuário deve ser capaz de encerrar a sessão no sistema
RF06	O usuário deve ser capaz de confirmar seu endereço de e-mail
RF07	O usuário deve ser capaz de ver suas informações de perfil
RF08	O usuário deve ser capaz de visualizar as suas cotas de uso no sistema

(continua)

Tabela 1 – Requisitos funcionais do sistema

(continuação)

Identificador	Descrição
RF09	Tanto o UCOM quanto o UADM devem ser capazes de criar instâncias com base em templates de instância e suas cotas de uso disponíveis
RF10	Tanto o UCOM quanto o UADM devem ser capazes de listar de forma paginada as instâncias que possuem
RF11	Tanto o UCOM quanto o UADM devem ser capazes de visualizar detalhes, como estado e recursos, das instâncias que possuem
RF12	Tanto o UCOM quanto o UADM devem ser capazes de excluir instâncias que possuem
RF13	Tanto o UCOM quanto o UADM devem ser capazes de ligar instâncias desligadas que possuem
RF14	Tanto o UCOM quanto o UADM devem ser capazes de desligar instâncias ligadas que possuem
RF15	Tanto o UCOM quanto o UADM devem ser capazes de reiniciar instâncias ligadas que possuem
RF16	Tanto o UCOM quanto o UADM devem ser capazes de acessar as instâncias ligadas que possuem
RF17	O UADM deve ser capaz de listar de forma paginada todos os templates de instância disponíveis
RF18	O UADM deve ser capaz de criar templates de instância
RF19	O UADM deve ser capaz de editar templates de instância
RF20	O UADM deve ser capaz de excluir templates de instância
RF21	O UADM deve ser capaz de criar templates de instância a partir de suas instâncias
RF22	O UADM deve ser capaz de listar de forma paginada todos os usuários do sistema
RF23	O UADM deve ser capaz de editar a permissão de usuários do sistema
RF24	O UADM deve ser capaz de alterar a cota de uso de usuários do sistema
RF25	O usuário deve ser capaz de visualizar as notificações enviadas pelo sistema
RF26	O sistema deve ser capaz de enviar notificações para os usuários alertando sobre a mudança de estado de suas instâncias
RF27	O sistema permitir o uso de filtros pré-definidos e busca textual para todos os recursos de listagem
RF28	O sistema deve ser capaz de desligar automaticamente instâncias ociosas
RF29	O sistema deve fornecer uma lista de imagens de sistema operacional recomendadas para a criação de templates de instância
RF30	O usuário deve ser capaz de acessar a documentação do sistema

Fonte: Autoria própria (2024)

Tabela 2 – Requisitos não funcionais do sistema

Identificador	Descrição
RNF01	O sistema deve ser responsivo e acessível em dispositivos móveis, mesmo que algumas funcionalidades sejam limitadas
RNF02	O sistema deve ser capaz de lidar com um grande número de usuários simultâneos
RNF03	O sistema deve ser capaz de lidar com um grande número de instâncias simultâneas

(continua)

Tabela 2 – Requisitos não funcionais do sistema

(continuação)

Identificador	Descrição
RNF04	O sistema deve atualizar as informações do estado das instâncias em tempo real, sem técnicas de *polling*[1].
RNF05	O sistema deve ser capaz de lidar com a criação de instâncias em menos de 5 minutos
RNF06	O sistema deve apresentar eventuais erros de forma clara e objetiva
RNF07	O sistema deve sinalizar ao usuário enquanto realiza operações assíncronas
RNF08	O sistema deve tratar uma conexão como ativa com uma instância se a mesma estiver sendo feita a partir do sistema. Caso contrário, a instância será considerada ociosa
RNF09	A documentação do sistema deve ser clara e objetiva, contendo explicações tanto sobre o funcionamento do sistema quanto sobre a utilização do sistema, além de apresentar todos os *endpoints* da API

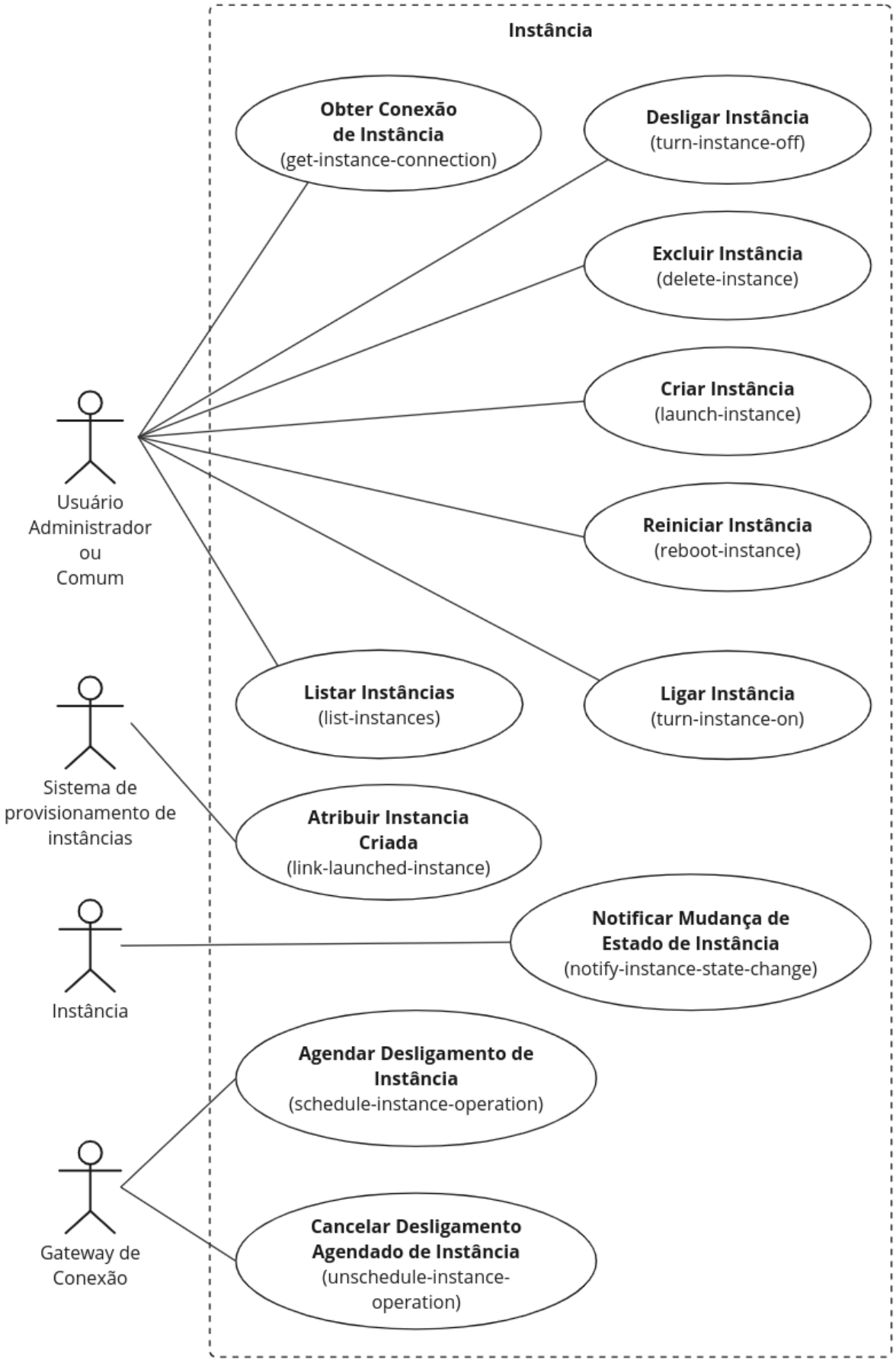
Fonte: Autoria própria (2024)

Os casos de uso do sistema foram modelados a partir dos requisitos funcionais identificados. De forma a dividir as responsabilidades do sistema em módulos, os casos de uso foram agrupados em quatro módulos principais: instância, template de instância, usuário e diversos, representados nas Figuras 5, 6, 7 e 8, respectivamente.

Como pode ser visto no diagrama da Figura 5, o módulo de instância necessitou de quatro interfaces para contextos distintos: Para usuários, para o sistema de provisionamento, para o sistema de envio de eventos das instâncias e para o *gateway* de conexão.

[1] O *polling* é uma técnica utilizada para acompanhar o status de uma operação baseada na execução de requisições espaçadas por um intervalo de tempo pré-definido. Essa técnica pode ser custosa e esgotar os recursos do sistema

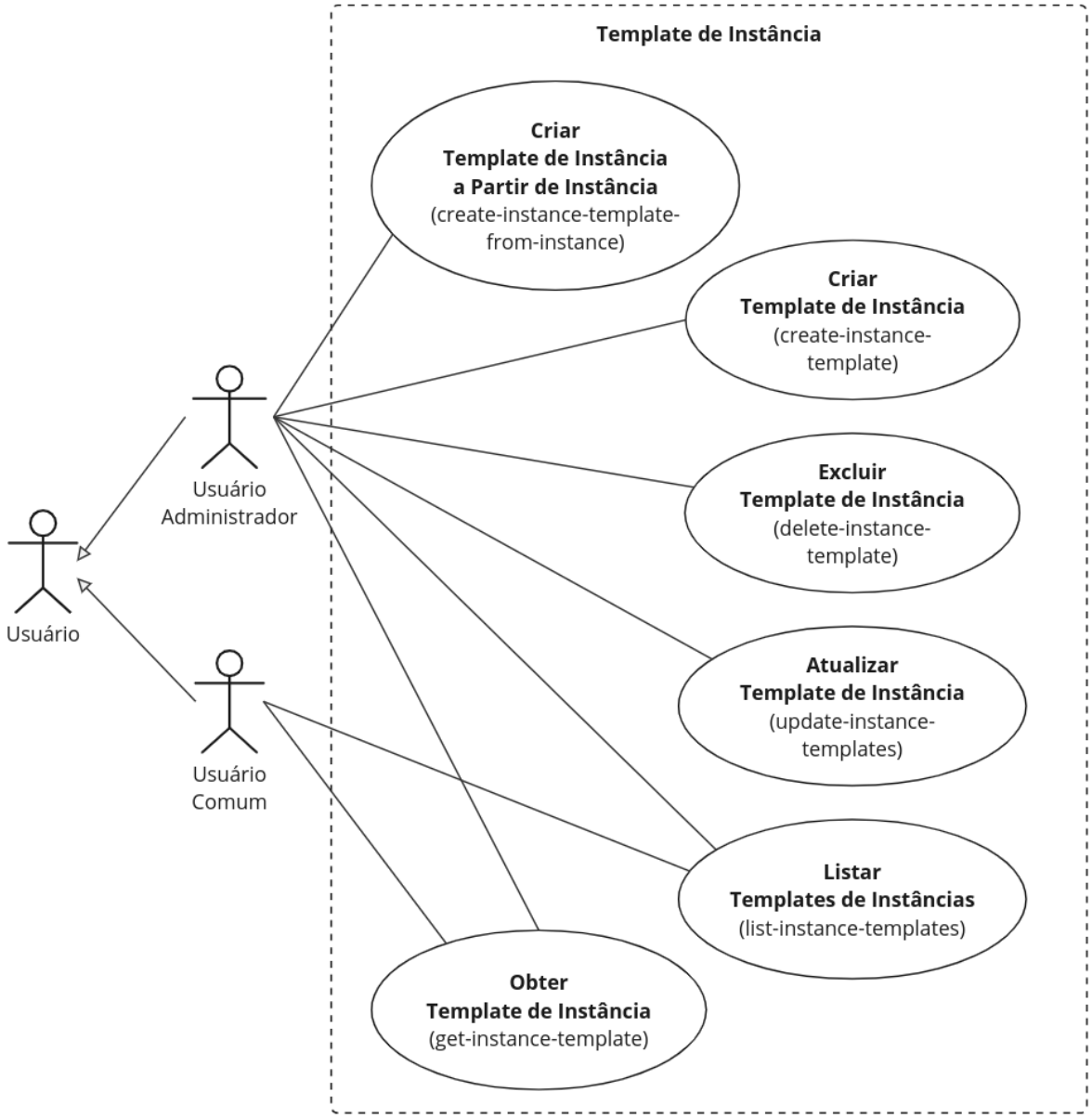
Figura 5 – Diagrama de caso de uso: Instância



Fonte: Autoria Própria (2024).

O diagrama da Figura 6 apresenta o módulo de template de instância, que é essencial para a criação de instâncias no sistema, isso porque as instâncias são criadas a partir de templates que definem o sistema operacional e a quantidade de armazenamento disponível. Tanto o UCOM quanto o UADM podem interagir com esse módulo, porém apenas o UADM pode gerenciar os templates e criá-los a partir de instâncias existentes, está que é uma funcionalidade importante para a disponibilização de configurações personalizadas para diferentes cenários de aplicação do sistema.

Figura 6 – Diagrama de caso de uso: Template de instância

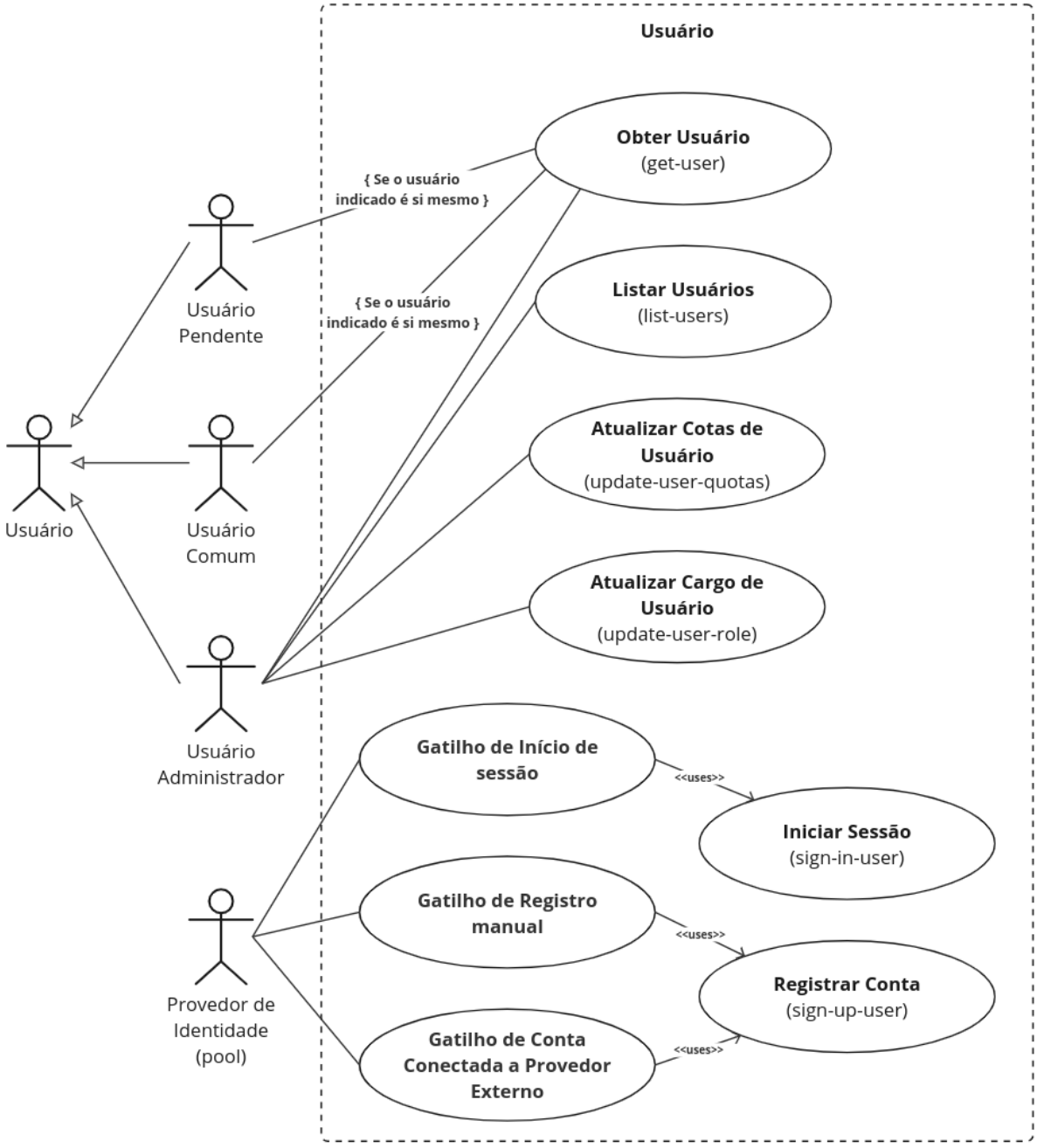


Fonte: Autoria Própria (2024).

No diagrama da Figura 7, é possível notar que o atores que representam o usuário final do sistema não tem acesso aos casos de uso de criação de conta nem de de início de sessão.

Isso porque o acesso ao sistema foi modelado para ser gerenciado por um serviço fora do contexto da implementação do sistema. Nesse caso, o sistema recebe os dados dos usuários através de gatilhos e então executa as operações para lidar apenas com dados não sensíveis e puramente operacionais.

Figura 7 – Diagrama de caso de uso: Usuário

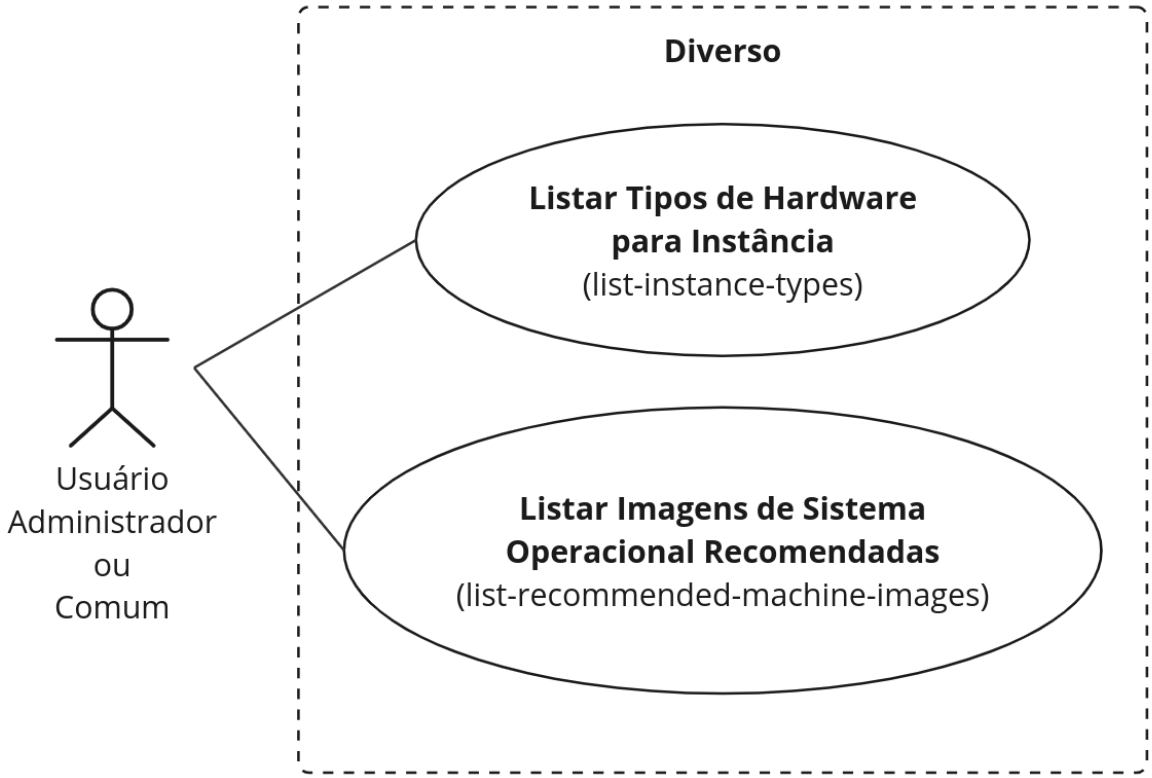


Fonte: Autoria Própria (2024).

No caso do diagrama da Figura 8, são apresentados apenas dois casos de uso auxiliares que servem para manter a consistência do sistema, permitindo que tanto a lista de tipos de

instância quanto a lista de imagens de sistema operacional recomendadas sejam sugeridas de forma dinâmica, evitando assim a necessidade de sair do sistema para buscar informações ou até mesmo que o usuário tenha que memorizar informações que não são de seu interesse direto.

Figura 8 – Diagrama de caso de uso: Diversos



Fonte: Autoria Própria (2024).

3.2.2 Projeto de arquitetura

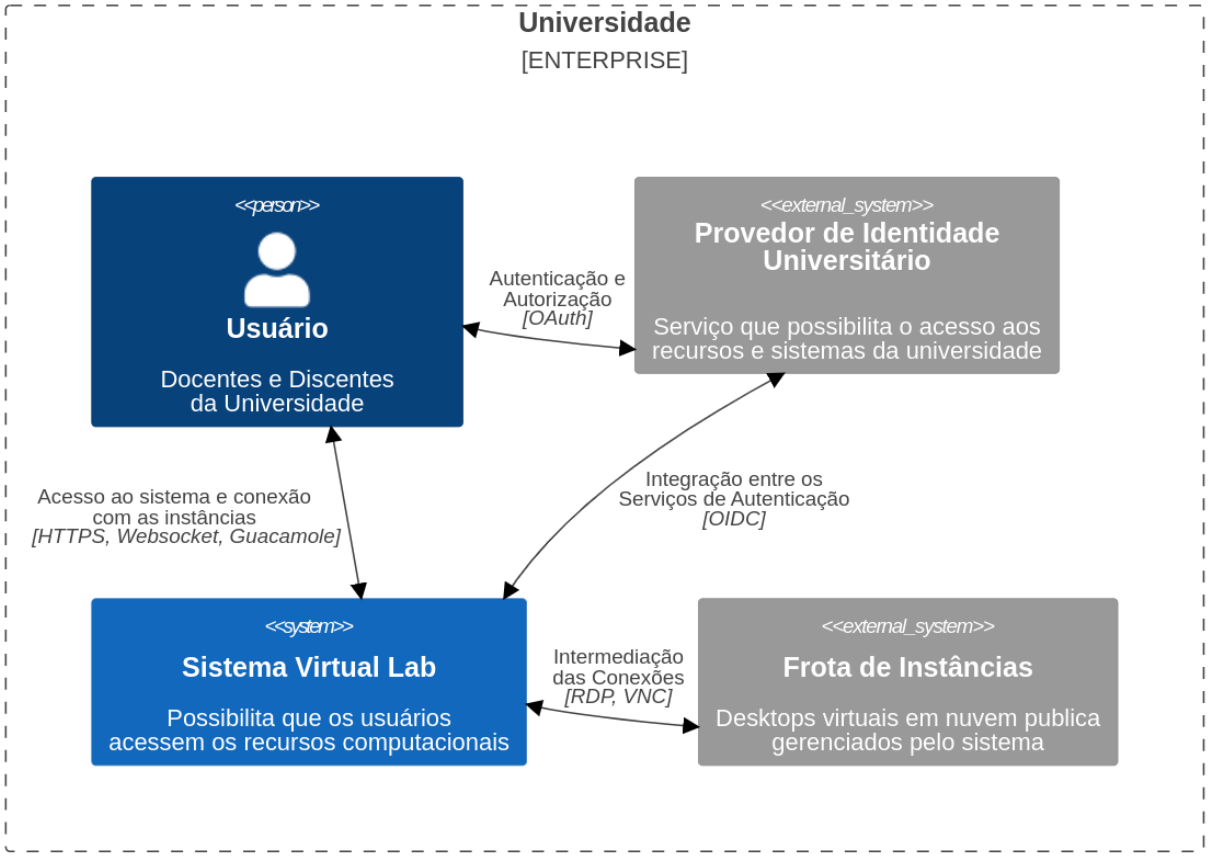
Em seguida, o projeto de arquitetura do sistema foi elaborado a fim de produzir representações de alto nível dos componentes do sistema e das camadas de dado e serviço (PRESSMAN ROGER S.; MAXIM, 2016).

A apresentação do projeto de arquitetura foi construída utilizando o modelo de visualização de arquitetura de sistemas *C4*, que promove uma abordagem partindo de uma visão abstrata até do sistema, passando por visões de contexto, *containers* e componentes, de forma a facilitar a compreensão do sistema como um todo (BROWN, 2018).

A figura Figura 9 apresenta o diagrama de contexto do sistema, que mostra as interações do sistema com os atores externos, como os usuários e o provedor de identidade externo. Esse

diagrama é a ponto de partida inicial para uma visão geral do contexto, onde o sistema em si é representado como um bloco central.

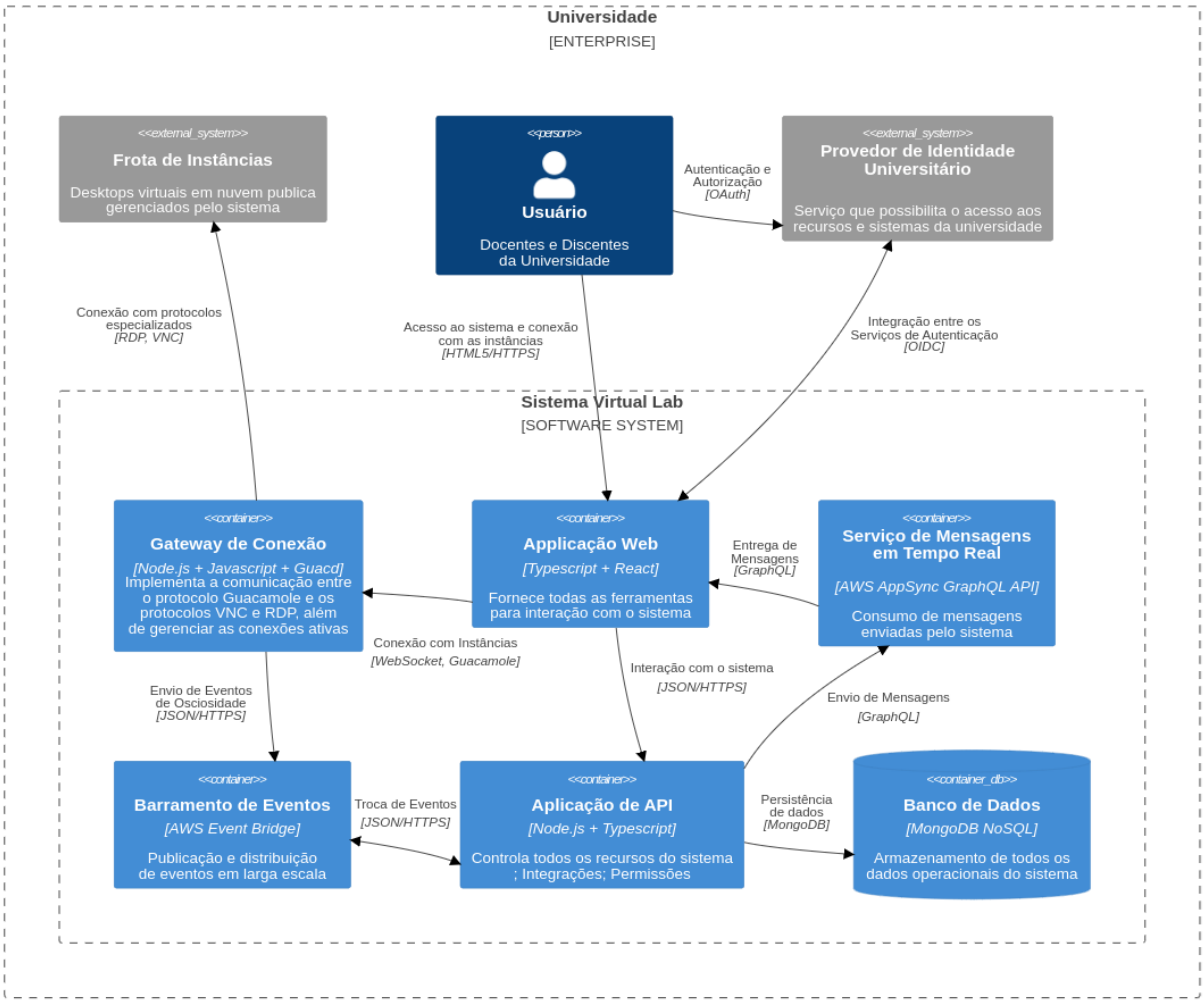
Figura 9 – Diagrama C4: Contexto do sistema



Fonte: Autoria Própria (2024).

Com uma ideia bem definida de como o sistema interage com o ambiente externo, o próximo passo foi a definição dos *containers*, que são os componentes de alto nível que compõem o sistema, além de explicitar os protocolos de comunicação. O diagrama da Figura 10 representa o resultado dessa etapa.

Figura 10 – Diagrama C4: *Containers* do sistema



Fonte: Autoria Própria (2024).

A partir do diagrama de *containers* do sistema, foi possível a estruturação interna de cada *container* em componentes, que são as partes fundamentais para a implementação do sistema. Nessa etapa, foram criados dois diagramas de componentes, um para a aplicação de API e outro para o *gateway* de conexão, que são as partes modeladas e implementadas por completo, enquanto os demais possuem apenas a definição de alto nível.

A Figura 11 apresenta o diagrama de componentes da aplicação de API, que é responsável por orquestrar todas as operações do sistema que estão dispostas em quatro módulos de funcionalidades:

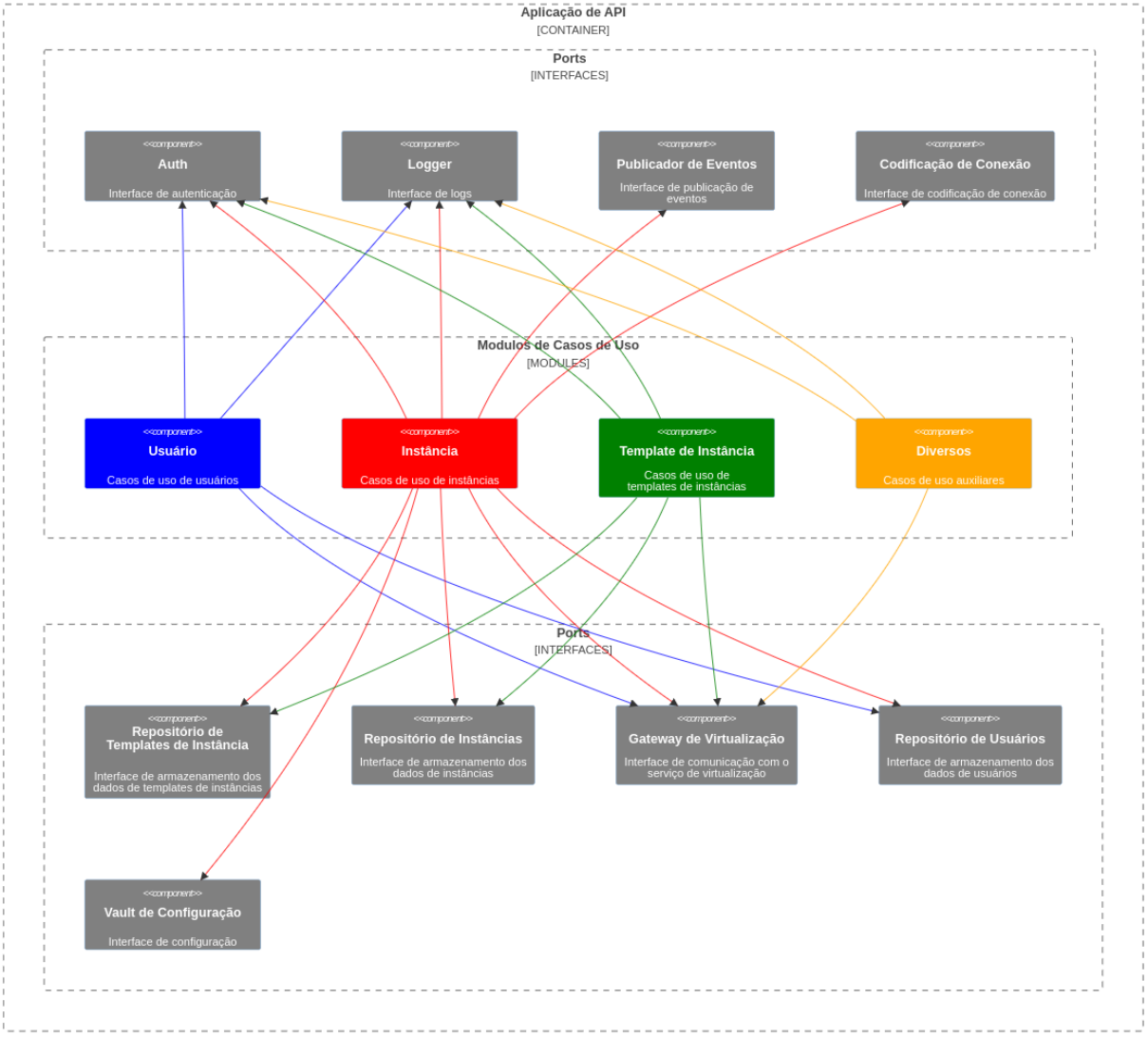
- **Usuário**: Módulo responsável por gerenciar as informações e permissões dos usuários do sistema.
- **Instância**: Módulo responsável por gerenciar as instâncias do sistema e as operações que podem ser realizadas sobre elas.

- **Template de Instância**: Módulo responsável por gerenciar os templates de instância disponíveis no sistema e as operações que podem ser realizadas sobre eles.
- **Diversos**: Módulo responsável por gerenciar os recursos do sistema que servem de apoio para a aplicação *web*, como a listagem de recursos recomendados utilizados na criação de templates de instância.

Nesse diagrama, a relação entre os módulos e os *ports*, que são as interfaces da camada de aplicação, é explicitada. Porém, a explicação detalhada das comunicação não está presente pois cada módulo é composto por diversos casos de uso que são tratados de forma independente. Portanto, uma seta originada em um módulo e direcionada a um *port* significa que pelo menos um caso de uso do módulo necessita de um *adapter* que implementa a interface do *port* para ser executado.

A discussão sobre *ports* e *adapters* é mais aprofundada na seção 3.5, onde os padrões de arquitetura de código utilizados são apresentados.

Figura 11 – Diagrama C4: Componentes da aplicação de API



Fonte: Autoria Própria (2024).

O *gateway* de conexão é o componente responsável por intermediar a conexão entre o cliente *web* e as instâncias de máquinas virtuais. O diagrama de componentes do *gateway* de conexão é apresentado na Figura 12.

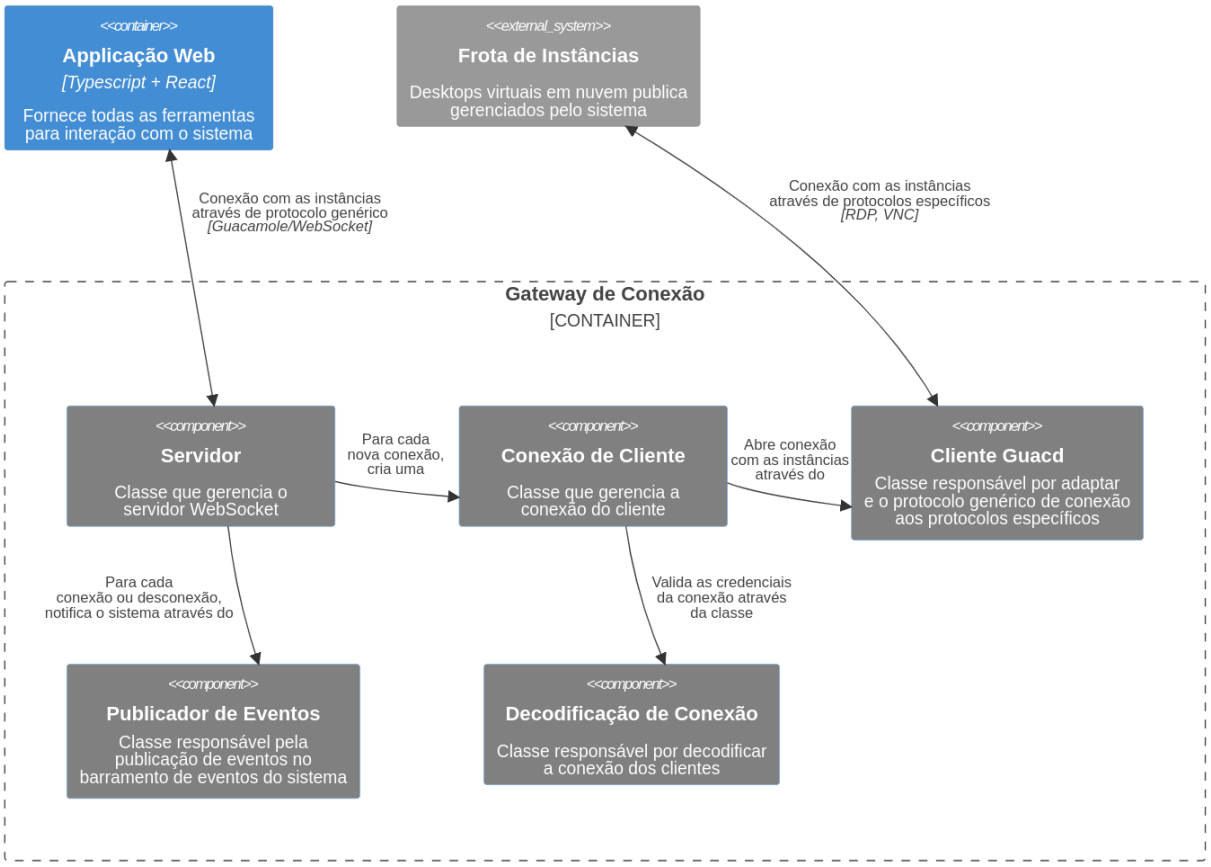
No *gateway* de conexão, além do gerenciamento das conexões com as instâncias de máquinas virtuais, o processo de identificação de conexões ativas e ociosas é realizado. Isso é de suma importância para o fluxo do sistema, já que seria difícil identificar a ociosidade através de alertas sensíveis às métricas de consumo dos recursos das instâncias.

Esse elo do sistema foi projetado para ser escalável horizontalmente, isto é, caso a demanda de conexões aumente, é possível adicionar mais instâncias do *gateway* de conexão para atender a demanda, sem a necessidade de alterar a arquitetura do sistema. Esse processo de escalabilidade é transparente para o usuário e também não afeta a disponibilidade do sistema.

Essa decisão foi tomada como uma medida de aprimoramento das perspectivas utilizadas no trabalho de Scandura (2022), que é discutido na seção 2.4. A escalabilidade horizontal

é uma característica importante para sistemas que possuem uma demanda variável, como é o caso de sistemas de ensino e pesquisa, que podem ter picos de uso em determinados períodos do ano. Dessa forma o sistema pode "crescer"para atender a demanda e "encolher"quando a demanda diminuir para economizar custos.

Figura 12 – Diagrama C4: Componentes do *gateway* de conexão



Fonte: Autoria Própria (2024).

3.3 Ferramentas e tecnologias

A Tabela 3 apresenta uma lista de ferramentas, tecnologias e serviços utilizados na implementação do sistema. As versões utilizadas estão escritas de acordo com o padrão de versionamento semântico. (PRESTON-WERNER, 2023)

Tabela 3 – Ferramentas e tecnologias utilizadas

Nome	Versão	Descrição
Node.js (OPENJS FOUNDATION, 2024c)	^18.19.0	Ambiente de execução de javascript utilizado em todos os serviços de *backend*
ECMAScript (ECMA INTERNATIONAL, 2022)	ES2020	Especificação formal da linguagem javascript a qual o código escrito é compatível

(continua)

Tabela 3 – Ferramentas e tecnologias utilizadas

(continuação)

Nome	Versão	Descrição
Typescript (MICROSOFT, 2024b)	$\leq$5.4.0	Extensão do javascript que adiciona suporte para tipagem, utilizada na implementação de todos os serviços de *backend* e do cliente *web* do sistema
NPM (NPM INC., 2024)	10.2.3	Gerenciador de dependências do Node.js
Shell Script (THE OPEN GROUP, 2018)	Não se Aplica (N/A)	Linguagem de scripts utilizada para implementar a configuração automatizada das instâncias baseadas em LINUX
Windows PowerShell Script (MICROSOFT, 2024a)	5	Linguagem de scripts utilizada para implementar a configuração automatizada das instâncias baseadas em WINDOWS
Apache Velocity Template Language (APACHE SOFTWARE FOUNDATION, 2020)	2.3	Engine de templates utilizada para definir as permissões de conexão dos usuários ao serviços de notificações do servidor ao cliente
Markdown (COMMONMARK, 2024)	N/A	Linguagem de marcação de texto utilizada na documentação do sistema
OpenAPI (THE LINUX FOUNDATION, 2020)	3.0.0	Especificação para documentação de API.
Zod (ZOD, 2024)	^3.23.8	Biblioteca utilizada para validar o conteúdo das requisições dos serviços
Prettier (PRETTIER, 2024)	^3.3.1	Utilitário utilizado para garantir a estilização do código.
Jest (OPENJS FOUNDATION, 2024b)	^29.7.0	*Framework* de teste para javascript
Eslint (OPENJS FOUNDATION, 2024a)	^8.56.0	Utilitário de análise estática de código.
AWS CDK (AMAZON WEB SERVICES, 2024m)	2.142.1	*Framework* utilizado para definição dos componentes de infraestrutura como código
SST (ANOMALY INNOVATIONS, 2024)	2.43.0	*Framework* para construção de aplicações *full-stack* na AWS utilizando infraestrutura como código
Docusaurus (META PLATFORMS, 2024a)	^3.4.0	Gerador de documentação em formato de site estático.
React (META PLATFORMS, 2024b)	^18.0.0	Biblioteca utilizada no desenvolvimento do cliente *web* do sistema
Stoplight Elements (SMARTBEAR SOFTWARE, 2024)	^8.3.1	Biblioteca utilizada para renderização da documentação da API.
Docker (DOCKER INC., 2024)	24.0.5	Ferramenta utilizada para a criação do ambiente isolado do *gateway* de conexão
Chakra UI (ADEBAYO, 2024)	^2.8.2	Biblioteca de componentes de interface utilizada no cliente *web*.
Apache Guacamole (APACHE SOFTWARE FOUNDATION, 2024)	1.5.3	Utilizado para intermediar a conexão entre o cliente *web* e *gateway* de conexão.
Vite (EVAN YOU, 2024)	^5.0.12	Ferramenta responsável por gerenciar a implementação e o empacotamento do cliente *web*

(continua)

Tabela 3 – Ferramentas e tecnologias utilizadas

(continuação)

Nome	Versão	Descrição
MongoDB Atlas (MONGODB INC., 2024)	Kit de Desenvolvimento de Software, do inglês *Software Development Kit* (SDK) ^6.0.3	Serviço de banco de dados não relacional utilizado para armazenar todos os dados do sistema.
AWS Amplify (AMAZON WEB SERVICES, 2024k)	SDK ^6.0.16	Biblioteca com integrações facilitadas, uitilizada no cliente *web* para autenticação e conexão com AWS AppSync
AWS Lambda PowerTools (AMAZON WEB SERVICES, 2023)	^2.1.1	Biblioteca utilizada para integrar a geração de logs dos serviços com o AWS CloudWatch
AWS AppSync (AMAZON WEB SERVICES, 2024l)	SDK ^3.592.0	Serviço utilizado para a comunicação em tempo real entre os serviços de *backend* e o cliente *web*
AWS Systems Manager (AMAZON WEB SERVICES, 2024u)	SDK ^3.592.0	Serviço utilizado para armazenar as configurações do sistema.
AWS Event Bridge (AMAZON WEB SERVICES, 2024h)	SDK ^3.592.0	Serviço de gerenciamento de eventos utilizado como barramento de comunicação entre os serviços de *backend*. Oferece também a funcionalidade de agendamento de eventos, utilizado no sistema de desligamento automático de instâncias ociosas.
AWS CloudFormation (AMAZON WEB SERVICES, 2024o)	SDK ^3.592.0	Serviço utilizado para armazenar a definição dos componentes de infraestrutura como código
AWS Cognito (AMAZON WEB SERVICES, 2024d)	SDK ^3.592.0	Serviço de gerenciamento de usuários e credenciais. Ele foi utilizado para gerenciar dados sensíveis, mantendo somente dados operacionais na integração com o banco de dados.
AWS Service Catalog (AMAZON WEB SERVICES, 2024t)	SDK ^3.592.0	Serviço de gerenciamento de modelos de IaC utilizado para o controle do ciclo de vida da instância e seus recursos dependentes
AWS CloudWatch (AMAZON WEB SERVICES, 2024c)	SDK ^3.592.0	Serviço de monitoramento utilizado para agregar os logs dos sistemas.
AWS Lambda (AMAZON WEB SERVICES, 2024r)	SDK ^3.592.0	Serviço de computação *serverless* utilizado para executar o serviço de API e tratamento de eventos do sistema
AWS Api Gateway (AMAZON WEB SERVICES, 2024a)	SDK ^3.592.0	Serviço gerenciado para criação de APIs *serverless*.
AWS Identity and Access Management (AMAZON WEB SERVICES, 2024q)	SDK ^3.592.0	Serivço utilizado para delimitação de todas as permissões atreladas aos componentes de infraestrutura do sistema

(continua)

Tabela 3 – Ferramentas e tecnologias utilizadas

(continuação)

Nome	Versão	Descrição
AWS S3 (AMAZON WEB SERVICES, 2024i)	SDK ^3.592.0	Serviço de armazenamento de objetos utilizado para servir o cliente *web* e armazenar todos os arquivos imutáveis do sistema
AWS CloudFront (AMAZON WEB SERVICES, 2024b)	SDK ^3.592.0	Serviço de distribuição de conteúdo utilizado como camada distribuída de cache para os serviços
AWS Certificate Manager (AMAZON WEB SERVICES, 2024n)	SDK ^3.592.0	Serviços utilizado para o gerenciamento dos certificados de domínio atrelados ao cliente *web* e à documentação
AWS Elastic Load Balancing (AMAZON WEB SERVICES, 2024v)	SDK ^3.592.0	Balanceador de carga gerenciado utilizado para gerenciar as conexões entre o cliente *web* e o *cluster* de *containers* rodando o *Gateway* de conexão.
AWS Elastic Container Service (AMAZON WEB SERVICES, 2024g)	SDK ^3.592.0	Serviço utilizado para a implantação do *gateway* de conexão, no formato de *cluster* de *containers* com escalabilidade previsível
AWS Elastic Container Registry (AMAZON WEB SERVICES, 2024f)	SDK ^3.592.0	Serviço utilizado para armazenar as imagens de *container* do *gateway* de conexão
AWS Simple Notification Service (AMAZON WEB SERVICES, 2024j)	SDK ^3.592.0	Serviço de Publisher-Subscriber utilizado para a transacionar as mensagens emitidas na criação de instâncias
AWS EC2 (AMAZON WEB SERVICES, 2024e)	SDK ^3.592.0	Serviço central utilizado para gerenciar as instâncias, regras de conexão e as imagens de sistema operacional

Fonte: Autoria própria (2024)

O ecossistema de tecnologias utilizado foi escolhido a partir de alguns critérios, como a familiaridade com o ambiente de desenvolvimento, a facilidade de integração entre as ferramentas e a disponibilidade de documentação e comunidade de suporte.

Visto que a UTFPR já possui integração com a AWS, a escolha da provedor de nuvem pública foi trivial, já que a infraestrutura de serviços da AWS é também utilizada em projetos a nível empresarial e acadêmico a nível global, oferecendo um portfólio extenso de serviços que atendem as necessidades do sistema.

O SST foi escolhido como um facilitador tanto para o fluxo de desenvolvimento local quanto para a definição de todos os componentes de arquitetura como código. A ferramenta disponibiliza componentes de alto nível que já implementam as melhores práticas de segurança e controle de custos. Além disso, o SST estabelece uma conexão entre os serviços da AWS, como as funções Lambda e o código escrito localmente, fazendo com que seja possível testar e depurar o código no seu ambiente próprio de implantação.

O Apache Guacamole™ foi escolhido como intermediador da conexão entre o cliente *web* e as instâncias de máquinas virtuais, porque trata-se de uma ferramenta de código aberto bem estabelecida que fornece os componentes necessários para a conexão remota a partir de

um navegador de internet. Além disso, ele é compatível com os protocolos de conexão remota utilizados no trabalho. No contexto do desenvolvimento do sistema, o Apache Guacamole™ foi utilizado de forma customizada. Apenas o *daemon* que implementa a conexão remota, o componente *web* que renderiza a interface de conexão e o protocolo de comunicação em si foram utilizados.

Essa customização foi necessária para que o sistema pudesse controlar todos os aspectos de interface e conexão, caso contrário, seria necessário utilizar a aplicação *web* já existente do Apache Guacamole™. Se esse fosse o caso, a customização do sistema dependeria de um conjunto de regras de negócio implementadas diretamente no Apache Guacamole™, o que dificultaria a manutenção e a evolução do sistema.

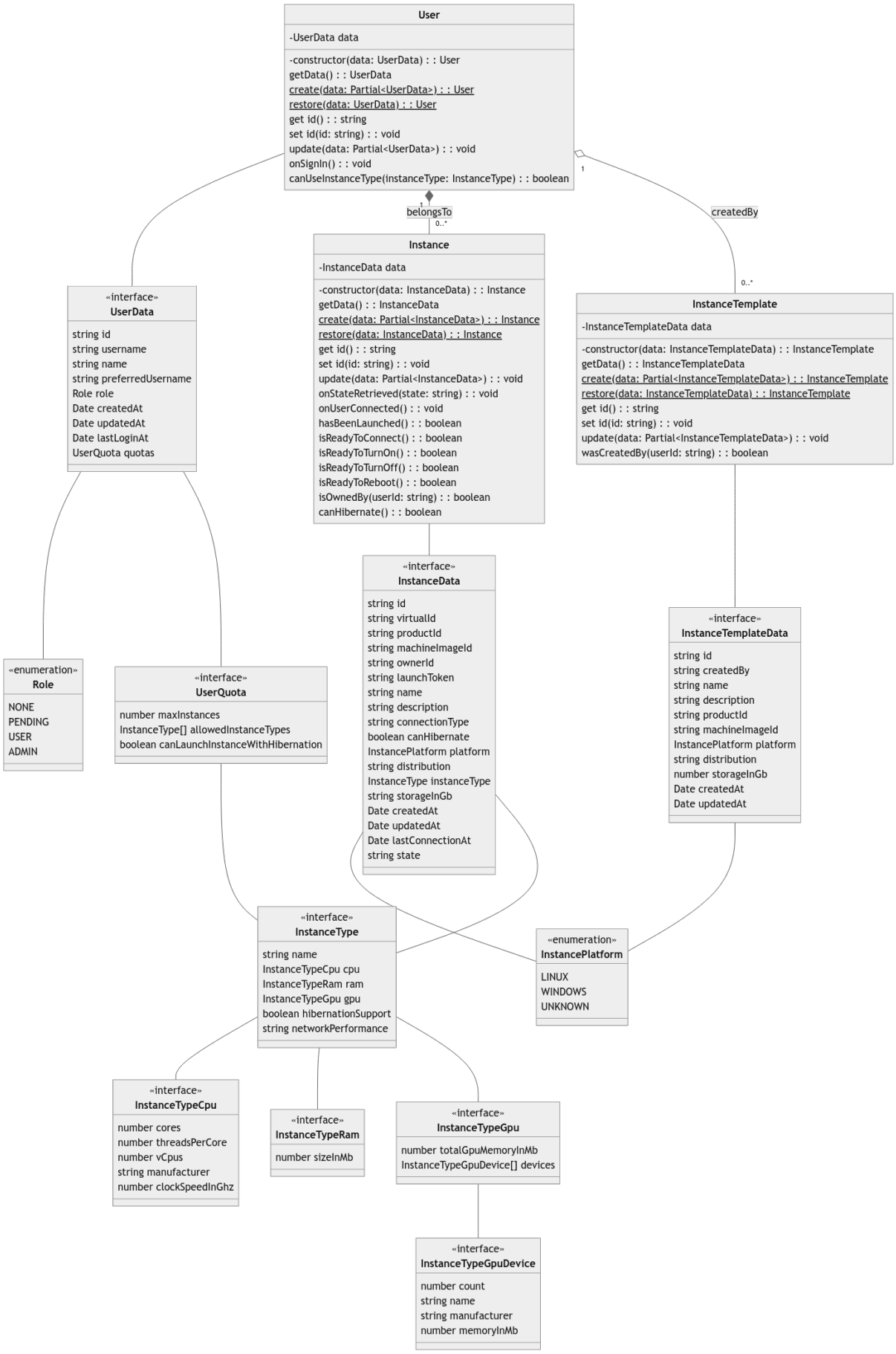
3.4 Modelo de dados

Aqui foi realizado o projeto do modelo de dados do sistema, que é a representação das entidades e relacionamentos centrais do sistema. O modelo de dados foi construído a partir dos requisitos funcionais do sistema, casos de uso e desdobramentos da arquitetura do sistema, de forma a garantir que todas as informações necessárias para o funcionamento do sistema estejam presentes.

No contexto de sistemas de *software* altamente dependentes da infraestrutura de nuvem, o modelo de dados precisa levar em consideração as características de funcionamento e dados disponibilizados pelos serviços de nuvem, como a AWS. Esse é o motivo pelo qual essa seção é apresentada após a discussão sobre a arquitetura do sistema e decisão das tecnologias utilizadas.

O modelo de dados foi elaborado a partir da especificação das entidades funcionais do sistema. A Figura 13 apresenta o diagrama de classes, onde cada classe é uma entidade funcional do sistema, exceto pelas interfaces representadas que apenas indicam modelos de dados mais complexos.

Figura 13 – Diagrama de classes do sistema



Fonte: Autoria Própria (2024).

3.5 Implementação

Essa seção apresenta os aspectos pertinentes ao processo de desenvolvimento do sistema, passando pela organização do repositório de código, dos serviços e componentes, dos componentes de infraestrutura e dos testes automatizados. É importante ressaltar que as implementações seguem as ferramentas e tecnologias apresentadas na seção 3.3.

3.5.1 Repositório de código

O repositório de código foi organizado através de uma estrutura de *monorepo*, onde o código fonte de todos os serviços e componentes de infraestrutura como código são versionados dentro de um único repositório de código fonte.

A utilização dessa abordagem com o uso do *framework* SST facilitou o desenvolvimento integrado de todas as partes do sistema desde o início, diminuindo as chances de problema no processo de integração entre partes separadas. Além disso, foi possível desenvolver conjuntos bem definidos de funcionalidades em pacotes envolvendo interfaces de usuário, regras de negócio e componentes de infraestrutura.

A estrutura de alto nível do repositório de código foi organizada da seguinte forma:

```
/
├── __tests__ ... Configurações do ciclo de vida dos testes
├── .github
│   └── workflows
│       └── ci.yml ... Implementação do fluxo de integração
│                      contínua
├── packages ... Implementação dos serviços do sistema
│   ├── api ... Aplicação de API
│   ├── app-sync-api ... Serviço de Mensagens em Tempo Real
│   ├── client ... Cliente web
│   ├── connection-gateway ... Gateway de Conexão
│   └── docs ... Documentação
├── stacks ... Definição de infraestrutura como código
│   ├── config ... Configurações de implantação
│   ├── products ... Base de stacks para cada sistema
│   │                operacional
│   └── scripts ... Scripts de preparação para cada sistema
│                   operacional
├── .env.example ... Exemplo das variáveis de ambiente
└── package.json ... Gerenciamento das dependências
                     compartilhadas e de infraestrutura
```

└── **sst.config.ts** ... Configuração do *framework* SST e
injeção das *stacks* de infraestrutura

A cada *commit* no repositório, um fluxo de integração contínua foi acionado executando uma sequência de operações que auxiliaram na validação do código entregue. O fluxo de integração contínua foi implementado utilizando o *GitHub Actions*, que é uma ferramenta de automação de *workflows* disponibilizada pelo GitHub. O Apêndice B apresenta o código do fluxo de integração contínua.

3.5.2 Banco de dados

O banco de dados utilizado no sistema foi o *MongoDB Atlas*, que é um serviço de banco de dados não relacional orientado a documentos, especificado também na Tabela 3.

O modelo dos dados foi implementado utilizando a biblioteca *mongodb*, que é um *Object Data Modeling* (ODM) para o MongoDB e Node.js. A biblioteca *mongodb* fornece uma interface de alto nível para a interação com o banco de dados, permitindo realizar operações tipadas.

Esse tipo de banco de dados não necessita de um configuração prévia do esquema dos dados, e por isso não foi necessário a implementação de migrações de banco de dados. Por outro lado a consistência dos dados deve ser garantida através de boas práticas de programação e validações, bem como mencionado nos princípios de modelagem no início da seção 3.2.

Não foram adicionados índices adicionais ao banco de dados. O índice padrão do *mongodb*, que faz do campo *id* um valor único e incremental, foi suficiente para garantir a performance das operações de leitura e escrita.

3.5.3 Serviço de mensagens em tempo real

Este serviço foi implementado utilizando o *AWS AppSync*, que é um serviço de comunicação em tempo real da AWS que permite a criação de APIs GraphQL com suporte para padrões de comunicação em tempo real executando sobre o protocolo *WebSocket*.

A implementação do serviço foi realizada em três frentes. A primeira foi a definição do esquema *GraphQL* com as operação de *Subscription* que são utilizadas pelos clientes para se inscreverem em canais de comunicação específicos, além dos *resolvers* que são funções escritas na linguagem *VTL* e que foram utilizadas para o processo de autorização das requisições. A segunda etapa foi a implementação das interfaces utilizada pelo servidor para enviar as mensagens para os clientes e pelos clientes para se inscreverem nos canais de comunicação. A terceira e última etapa consistiu na implementação dos recursos de infraestrutura necessários para o funcionamento do serviço, como a definição de permissões, conexão do esquema e dos *resolvers*.

Para garantir o isolamento dos canais de comunicação e evitar que usuários mal intencionados possam acessar informações de outros usuários, os usuários foram habilitados de realizar a inscrição apenas no canal com o seu próprio identificador. Esse identificador é gerado no momento da criação da conta do usuário e é utilizado para garantir a unicidade do canal de comunicação.

A implementação da interface feita no cliente *web* foi intermediada pela da biblioteca *AWS Amplify*, que é uma biblioteca de integração com os serviços da AWS para aplicações *web* e móveis.

3.5.4 Barramento de eventos

O barramento de eventos foi implementado utilizando o *AWS Event Bridge*, que é um serviço de gerenciamento de eventos da AWS que permite a comunicação entre os serviços de forma assíncrona e escalável. A implementação é baseada na criação de regras de eventos que são acionadas quando um evento específico é publicado no barramento. Ele foi escolhido para ser o principal meio de comunicação entre os serviços de *backend*, já que ele oferece integração com diversos serviços da AWS e também funcionalidade de agendamento de eventos, que foi utilizada no sistema de desligamento automático de instâncias ociosas.

A implementação do barramento de eventos foi realizada em duas frentes. A primeira foi a definição das regras de eventos que são acionadas quando um evento específico é publicado no barramento. A segunda etapa foi a implementação dos recursos de infraestrutura necessários para o funcionamento do serviço, juntamente com a conexão entre os serviços de *backend* e o barramento de eventos.

Para publicar eventos no barramento, os serviços de *backend* utilizaram o próprio SDK da AWS.

3.5.5 Gateway de conexão

Esse serviço foi modelado para intermediar a conexão entre o cliente *web* e as instâncias de máquinas virtuais, e sua implementação foi baseada na estrutura de um servidor *websocket*, onde as conexão são persistidas e mantidas abertas para permitir a comunicação.

A implementação do código foi baseada no diagrama da Figura 12, onde o serviço foi dividido em cinco classes de responsabilidades específicas.

A primeira classe, *Server* ficou responsável por gerenciar o ciclo de vida do servidor *websocket*, aceitando novas conexões e mantendo as conexões ativas. A segunda classe, *ClientConnection* ficou responsável por implementar as operações pertinentes a troca de dados entre o cliente que iniciou a conexão e a classe *guacdClient* que é a terceira classe, responsável especificamente por interagir com o binário *guacd* do Apache Guacamole™, que é nativamente

implementado na linguagem C. A quarta classe, *Crypt* ficou responsável por implementar as operações de descriptografia do *token* de conexão, que é gerado pela aplicação de API e encaminhado através do cliente *web*. A quinta e última classe é o *EventPublisher*, que é utilizado pela classe *Server* para publicar eventos de conexão e desconexão no barramento de eventos. Eventos estes que são consumidos por funções *Lambda* dentro do domínio da aplicação de API que agendam o desligamento automático de instâncias ociosas.

O algoritmo de criptografia utilizado tanto pela classe *Crypt* para descriptografar o *token* de conexão, quanto pela aplicação de API para criptografar o *token* de conexão, foi o algoritmo de criptografia simétrica *AES-256*. Esse algoritmo foi utilizado em *Cipher Block Chaining (CBC)*, que é um modo de funcionamento que evita que padrões repetidos no texto claro sejam visíveis no texto cifrado.

Como o *guacd*, binário do Apache Guacamole™, precisa ser executado no mesmo ambiente que as classes do serviços, foi necessário a criação de um ambiente completo para a execução do serviço, através de um *container Docker*. O *container* utilizou a imagem oficial fornecida pelo Apache Guacamole™ somente com o binário, como também a imagem oficial do *Node.js* para a execução do código do serviço, dessa forma foi possível garantir que o serviço se comportaria da mesma maneira em qualquer ambiente. Quando o *container* é iniciado, a ferramenta *supervisord* gerencia a execução dos processos do *guacd* e do serviço, permitindo o controle e monitoramento dos processos através de um único ponto de entrada.

3.5.6 Aplicação de API

Esse serviço é o ponto central de comunicação entre o cliente *web* e os demais serviços de *backend*. Ele foi implementado utilizando *typescript* e *Node.js* e padrões de arquitetura de código e de projeto para que fosse possível expandir o sistema de maneira incremental, sem que a complexidade do código aumentasse de forma descontrolada.

Para isso, o padrão de arquitetura de código *Arquitetura Hexagonal*, conhecida também como *Ports and Adapters*, foi utilizado. Esse padrão de arquitetura de código é baseado na separação das responsabilidades do código em camadas, onde a camada de aplicação é o núcleo do sistema e as camadas de infraestrutura e de interface são adaptadores que permitem a comunicação com o núcleo. Além disso, essa arquitetura permitiu que o conceito de inversão de dependência fosse aplicado, criando uma separação clara entre as regras de negócio e a implementação dos serviços.

A estrutura do serviço foi organizada da seguinte forma:

```
/packages/api
├── application ... Camada que contém as regras de negócio
│                   do sistema, implementadas em classes de
│                   casos de uso que orquestram as entidades
│                   do domínio do sistema e os ports de
│                   comunicação
│   ├── use-cases
│   │   ├── instance
│   │   │   ├── __tests__
│   │   │   └── launch-instance.ts
│   │   ├── instance-template
│   │   ├── misc
│   │   └── user
│   └── auth.ts ... Exemplo de adapter de autenticação que
│                   define as operações de autenticação do
│                   sistema, sem implementar a lógica de
│                   autenticação
├── domain ... Camada de domínio da aplicação, que contém
│              a implementação das classes modeladas e dos
│              protocolos de dados utilizados em todas as
│              outras camadas do sistema
│   ├── dtos
│   └── entities
│       ├── instance-template.ts
│       ├── instance.ts
│       └── user.ts
└── infrastructure ... Camada de infraestrutura, que implementa
                       as interfaces definidas na camada
                       de aplicação de modo a realizar a
                       comunicação com os serviços externos
    └── auth
        └── cognito-auth.ts ... Exemplo de implementação do
                                port de autenticação utilizando
                                o serviço de autenticação da
                                AWS
```

```
    │   └── in-memory-auth.ts ... Exemplo de implementação do
    │                                port de autenticação utilizando
    │                                lógica isolada em memória, com
    │                                o objetivo de ser utilizado em
    │                                testes unitários
    └── interfaces ... Camada de interfaces, que utiliza todas as
        │                  outras camadas para implementar os pontos de
        │                  acesso do sistema
        ├── api ... Implementação de todos os endpoits da API,
        │           compatíveis para serem executados em funções
        │           Lambda
        ├── events ... Implementação dos pontos de entrada que
        │              consomem os eventos do barramento de eventos
        └── jobs ... Implementação de pontos de entrada que realizam
                     operações assíncronas, como as rotinas de
                     atualização de alguns recursos no momento de
                     implantação do código
```

3.5.7 Cliente *web*

O cliente *web* foi implementado utilizando *typescript* e *React* como a base tecnológica principal. Para a construção da interface a biblioteca de componentes *Chakra UI* foi utilizada, pois ela oferece uma série de componentes customizáveis que implementam boas práticas de acessibilidade e permitem um controle sobre a apresentação dos componentes em diferentes formatos de tela.

As páginas e contexto do cliente *web* foram estruturados em componentes reutilizáveis, que são utilizados em diferentes partes do sistema. Bem como a camada que lida com a troca de dados com a aplicação de API, que foi implementada utilizando a biblioteca *ReactQuery*, que é uma biblioteca de gerenciamento de estado que fornece uma abstração sobre a chamadas dos *endpoints* da API, permitindo o cache local com tempo de expiração pré-definido e também o consumo dos recursos da API de forma equilibrada, evitando requisições redundantes ou excessivas.

O desenvolvimento levou em consideração todos os casos de uso, requisitos de sistema e perspectivas de modelagem para proporcionar uma experiência de usuário simplificada e intuitiva. Embora alguns aspectos técnicos sejam difíceis de serem apresentados através de abstrações simplificadas, como é o caso dos tipos de *hardware* de instância e pontos relacionados ao sistema operacional das instâncias, foi possível delegar a responsabilidade do mínimo de conhecimento técnico aos usuários com papel de UADM no sistema.

A interface desenvolvida e os fluxos que o usuário pode realizar dentro da Aplicação *Web* serão apresentados no Capítulo 4.

3.5.8 Documentação

A documentação foi desenvolvida utilizando o *Docusaurus*, que é um gerador de documentação em formato de site estático. Com ele foi possível escrever a documentação em formato de *Markdown* com suporte a componentes de *React*, a mesma tecnologia utilizada no cliente *web*, para então gerar um site estático com a documentação do sistema.

Ao utilizar essa ferramenta, o processo de documentação ficou focado na produção dos conteúdos de ajuda ao usuário e também na documentação técnica do sistema, sem a preocupação da estrutura do sistema de documentação, já que é uma parte acoplada ao sistema que não precisa de novas funcionalidades, precisa apenas de complementos textuais.

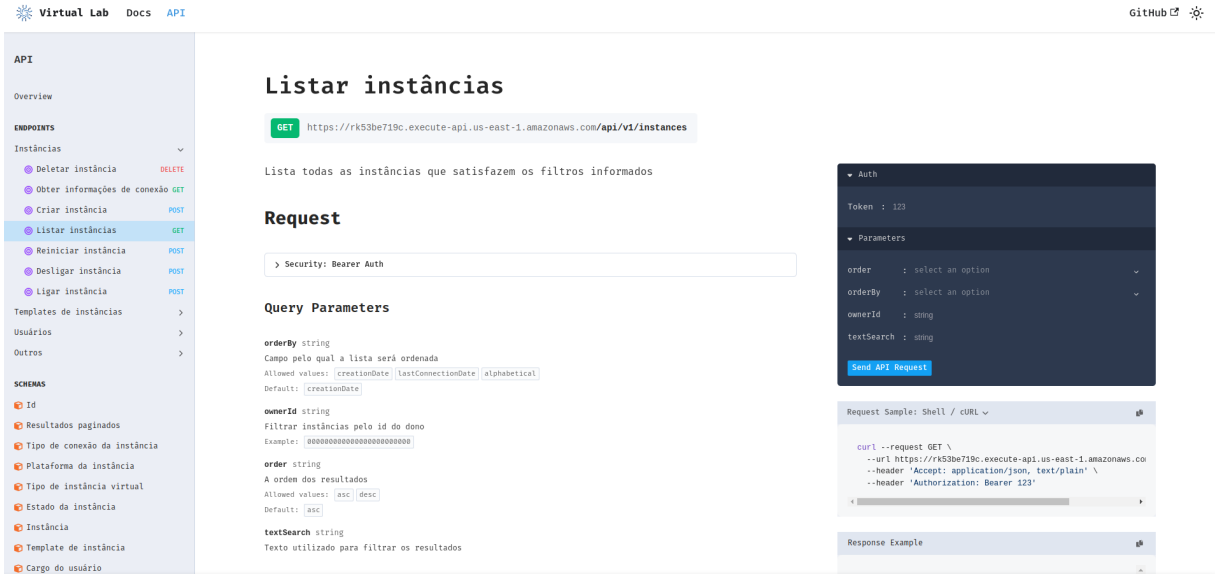
Para que a documentação das rotas de *api* pudessem ser listadas de forma automática, foi utilizado o *Stoplight Elements*, que é uma biblioteca que permite a renderização de documentação de API em formato de componente *React* a partir de um arquivo de especificação *OpenAPI* também gerado automaticamente dentro do processo de sintetização da *stacks* da aplicação de API, onde as rotas são definidas.

A Figura 14 apresenta a página inicial da documentação do sistema e a Figura 15 apresenta um exemplo de rota documentada.

Figura 14 – Página inicial da documentação do sistema

Fonte: Autoria Própria (2024).

Figura 15 – Documentação da rota de API: Listar instâncias



Fonte: Autoria Própria (2024).

3.5.9 Testes automatizados

Testes exaustivos não garantem a ausência de falhas, mas podem reduzir a probabilidade de falhas críticas. (PRESSMAN ROGER S.; MAXIM, 2016)

Os testes do sistema foram realizados de duas formas, a primeira através de testes unitários automatizados e a segunda através da execução manual de um teste *end-to-end* percorrendo os caminhos críticos da aplicação.

Os testes unitários foram implementados através do *framework Jest* a fim validar de forma satisfatória todos os casos de uso dos serviços de *backend*. Como visto anteriormente, a estrutura do projeto já foi planejada para facilitar a escrita de testes utilizando *adapters* implementados especificamente para reproduzir o comportamento dos serviços reais da camada de infraestrutura.

A seguir os testes unitários dos casos de usos mais importantes implementados serão listados, com suas condições iniciais e resultados esperados. O Apêndice A apresenta o relatório de cobertura dos testes por arquivo com os resultados obtidos.

Tabela 4 – Testes unitários do caso de uso: Desligar instância ociosa

Condição inicial	Resultado esperado
Quando a entrada do caso de uso é inválida	Deve retornar um erro de validação
Quando a instância indicada não existe	Não deve realizar nenhuma ação
Quando a instância indicada existe e pertence a um usuário	Deve desligar a instância

Fonte: Autoria própria (2024).

Os casos de teste da Tabela 4 são importantes para garantir que quando o agendamento para desligamento é acionado, a instância de fato será desligada, evitando assim que recursos sejam desperdiçados e que o usuário consiga acessar a instância de outro local diferente do cliente *web* do sistema.

Tabela 5 – Testes unitários do caso de uso: Deletar instância

Condição inicial	Resultado esperado
Quando a entrada do caso de uso é inválida	Deve retornar um erro de validação
Quando o usuário não tem permissão para deletar a instância	Deve retornar um erro de permissão
Quando a instância indicada não existe	Não retornar um erro de instância não encontrada
Quando o papel do usuário não é administrador e a instância pertence a outro usuário	Deve retornar um erro de permissão
Quando o usuário tem permissão para deletar a instância	Deve deletar a instância e todos os recursos associados

Fonte: Autoria própria (2024).

Na Tabela 5 os testes também são importantes para o gerenciamento correto dos recursos da nuvem, já que é preciso deletar o registro do objeto no banco de dados e também todos os recursos que foram criados na nuvem, como a instância, o grupo de segurança, volumes de armazenamento, estes que se ficarem foram de uso, podem gerar custos desnecessários e difíceis de rastrear.

Tabela 6 – Testes unitários do caso de uso: Obter informações de conexão com instância

Condição inicial	Resultado esperado
Quando a entrada do caso de uso é inválida	Deve retornar um erro de validação
Quando o usuário não está habilitado para acessar o sistema	Deve retornar um erro de permissão
Quando a instância indicada não existe	Deve retornar um erro de instância não encontrada
Quando o sistema não consegue obter a senha da instância	Deve retornar um erro interno
Quando o usuário não é o dono da instância e não é administrador	Deve retornar um erro de permissão
Quando o usuário tem acesso à instância e a mesma ainda não está pronta para conexão	Deve retornar um erro de violação do caso de uso
Quando o usuário tem acesso à instância e a mesma não está ligada	Deve retornar um erro de violação do caso de uso
Quando o usuário tem acesso à instância e o protocolo interno de conexão é RDP	Deve retornar as informações de conexão com a instância
Quando o usuário tem acesso à instância e o protocolo interno de conexão é VNC	Deve retornar as informações de conexão com a instância

Fonte: Autoria própria (2024).

Os testes da Tabela 6 garantem dentre outras coisas,que só será possível iniciar uma conexão com a instância se a mesma estiver de fato preparada para receber conexões. Para

chegar nesse estado, a instância precisa passar por várias etapas que sem a validação correta, ficariam praticamente impossíveis de serem rastreadas e corrigidas.

Tabela 7 – Testes unitários do caso de uso: Criar instância

Condição inicial	Resultado esperado
Quando a entrada do caso de uso é inválida	Deve retornar um erro de validação
Quando o usuário não está habilitado para acessar o sistema	Deve retornar um erro de permissão
Quando o usuário não é encontrado no sistema	Deve retornar um erro de usuário não encontrado
Quando o template de instância indicado não existe	Deve retornar um erro de template não encontrado
Quando o tipo de *hardware* indicado não existe	Deve retornar um erro de tipo de *hardware* não encontrado
Quando a imagem de sistema operacional indicada não existe	Deve retornar um erro de imagem não encontrada
Quando o papel do usuário não é administrador e o dono indicado é diferente do usuário	Deve retornar um erro de permissão
Quando o usuário não é administrador e o limite de instâncias do usuário foi atingido	Deve retornar um erro de violação do caso de uso
Quando o usuário não é administrador e o tipo de *hardware* indicado não é permitido para o usuário	Deve retornar um erro de violação do caso de uso
Quando o usuário não é administrador e a opção de hibernação não é permitida para o usuário	Deve retornar um erro de violação do caso de uso
Quando o usuário tem permissão para criar a instância	Deve criar a instância e retornar as informações da instância

Fonte: Autoria própria (2024).

Os testes da Tabela 7 são importantes para garantir que a criação de instâncias seja feita de forma segura e controlada, evitando que usuários mal intencionados possam criar instâncias sem permissão ou que usuários comuns possam criar instâncias com configurações que não são permitidas para o seu papel de usuário.

Tabela 8 – Testes unitários do caso de uso: Notificar mudança de estado de instância

Condição inicial	Resultado esperado
Quando a entrada do caso de uso é inválida	Deve retornar um erro de validação
Quando a instância indicada não existe	Não deve realizar nenhuma ação
Quando o usuário dono da instância não existe	Não deve realizar nenhuma ação
Quando a instância recebe o estado de ligada	Deve sinalizar o sistema de desligamento automático e notificar o usuário dono da instância sobre a mudança de estado
Quando a instância recebe qualquer outro estado	Deve notificar o usuário dono da instância sobre a mudança de estado

Fonte: Autoria própria (2024).

Os testes da Tabela 8 são importantes para garantir que o sistema notifique os usuários sobre as mudanças de estado das instâncias, permitindo a atualização da interface do cliente *web* em tempo real evitando que o usuário tenha que recarregar a página para visualizar as mudanças.

Tabela 9 – Testes unitários do caso de uso: Criar template de instância a partir de instância

Condição inicial	Resultado esperado
Quando a entrada do caso de uso é inválida	Deve retornar um erro de validação
Quando o usuário não é administrador	Deve retornar um erro de permissão
Quando a instância indicada não existe	Deve retornar um erro de instância não encontrada
Quando a instância utilizada como base ainda não terminou de ser configurada	Deve retornar um erro de violação do caso de uso
Quando a quantidade de armazenamento indicada é menor que o armazenamento mínimo requerido pela imagem de sistema operacional	Deve retornar um erro de violação do caso de uso
Quando todas as condições são atendidas	Deve criar o template de instância e retornar as informações do template

Fonte: Autoria própria (2024).

Os testes da Tabela 9 são importantes para garantir que os administradores possam criar templates de instâncias a partir de instâncias já criadas. As verificações nesse caso de uso garantem que os usuários terão templates válidos para criar instâncias que funcionem corretamente.

Tabela 10 – Testes unitários do caso de uso: Cadastrar usuário

Condição inicial	Resultado esperado
Quando a entrada do caso de uso é inválida	Deve retornar um erro de validação
Quando já existe um usuário no banco de dados com o *username* recebido na entrada	Deve retornar o usuário já cadastrado
Quando não existir um registro no banco de dados com o *username* recebido na entrada	Deve criar o usuário e retornar as informações do usuário
Quando o tipo de *hardware* de instância padrão indicado não existe	Deve retornar um erro interno

Fonte: Autoria própria (2024).

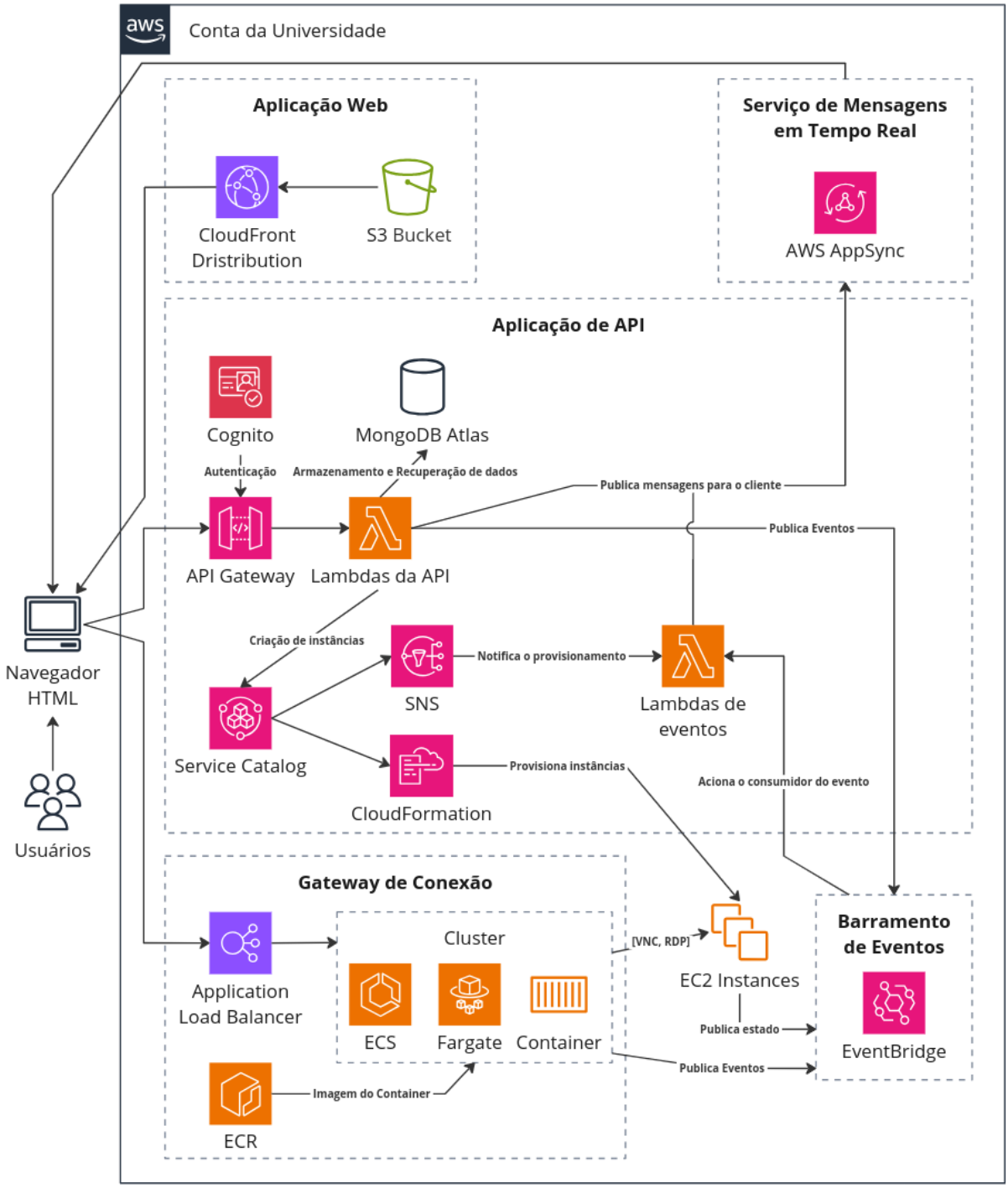
Os testes da Tabela 10 validam se a sincronia entre o *AWS Congito* e o banco de dados do sistema prevê casos de borda para que não haja duplicidade de usuários no sistema e os mesmo possam ser criados de forma controlada, tanto através do cliente *web*, quanto através do *console* ou *cli* da AWS.

3.6 Implantação do sistema

Essa seção descreve o processo de implantação do sistema em um ambiente de produção. Isso significa que após a execução das etapas a seguir, o sistema estará disponível para ser acessado pelos usuários finais.

Como resultado do processo de modelagem aplicado no decorrer deste capítulo, obtendo-se a arquitetura do sistema, a implementação dos serviços de *backend* e do cliente *web*, então foi possível estabelecer o diagrama de infraestrutura de nuvem do sistema, que relaciona todos os serviços e recursos da AWS especificados na Tabela 3 e como eles são conectados para a execução do *Virtual Lab* em um ambiente de produção. Esse diagrama é apresentado na Figura 16.

Figura 16 – Diagrama de infraestrutura dos serviços de nuvem do *Virtual Lab* na AWS



Fonte: Autoria Própria (2024).

A implantação do *Virtual Lab* depende do acesso a uma conta da AWS com permissões administrativas para criar e gerenciar todos os serviços descritos na seção 3.3.

Além disso, é necessário ter o utilitário de linha de comando *aws-cli* instalado e configurado com as credenciais de acesso configuradas, visto que o processo é automatizado. (AMAZON WEB SERVICES, 2024p). A documentação mais detalhada sobre o processo de

configuração do *aws-cli* pode ser encontrada no site oficial da AWS, bem como na documentação do repositório do projeto.

A seguir são descritas as etapas necessárias para a implantação a partir do código fonte do sistema, utilizando um computador com as seguintes configurações:

- Sistema operacional: *Ubuntu 24.04 LTS*
- Processador: *Intel Core i7-1165G7 CPU @ 2.80GHz x 8*
- Memória RAM: 15,3 GiB
- Espaço em disco: 472,8 GiB

Contando também com os seguintes *softwares* instalados:

- *Node.js* versão 18.19.0
- *npm* versão 10.2.3
- *Docker* versão 24.0.5
- *aws-cli* versão 2.11.9

3.6.1 Configuração das dependências

Com o código fonte do sistema em mãos, o primeiro passo é instalar todas as dependências do projeto. Para isso, basta executar o seguinte comando em um terminal na raiz do projeto:

```
# npm install
```

Esse comando irá instalar todas as dependências do projeto, incluindo as dependências de desenvolvimento, que também são necessárias para o processo de implantação.

Com o término da execução, uma nova pasta chamada *node_modules* será criada na raiz do projeto, contendo todas as dependências instaladas.

3.6.2 Variáveis de ambiente

O próximo passo é configurar as variáveis de ambiente do sistema. Elas são necessárias para a identificação das credenciais de acesso à AWS, bem como para a configuração do comportamento de certos componentes do sistema.

A primeira etapa é criar um novo arquivo chamado *.env* na raiz do projeto. No projeto existe um arquivo chamado *.env.example* que contém um exemplo das variáveis de ambiente necessárias. Para criar uma cópia desse arquivo, basta executar o seguinte comando:

```
# cp .env.example .env
```

Todas variáveis do arquivo são opcionais, por se tratarem de variáveis de ambiente, o sistema irá utilizar valores padrão caso as variáveis não sejam definidas. A seguir é apresentada uma lista das variáveis de ambiente disponíveis e suas descrições:

- *READABLE_LOG_FORMAT*: Define o formato dos logs do sistema. Quando definido como *true*, os logs serão formatados de forma legível. Caso contrário, os logs serão formatados de forma compacta. O valor padrão é *false*.
- *RETAIN_USER_POOL_ON_DELETE*: Define se o *AWS Cognito User Pool* deve ser mantido após a exclusão do sistema. Quando definido como *true*, o *User Pool* será mantido. Caso contrário, o *User Pool* será excluído. O valor padrão é *false*. Em ambientes de produção, é recomendado manter o *User Pool* após a exclusão do sistema, para que os usuários possam ser recuperados em caso de falha.
- *USER_POOL_IDENTITY_PROVIDER*: Define se um provedor de identidade externo deve ser utilizado em conjunto com o *AWS Cognito User Pool*. Quando definido como *true*, um provedor de identidade externo será utilizado e um botão será habilitado na tela de *login* para utilizar essa funcionalidade. Caso contrário, o provedor de identidade padrão da AWS será utilizado. O valor padrão é *false*. Caso essa variável seja definida como *true*, as seguintes variáveis de ambiente também deverão ser definidas:
  - *USER_POOL_IDENTITY_PROVIDER_CLIENT_ID*: O ID do cliente do provedor de identidade externo.
  - *USER_POOL_IDENTITY_PROVIDER_CLIENT_SECRET*: O segredo do cliente do provedor de identidade externo.
  - *USER_POOL_IDENTITY_PROVIDER_ISSUER_URL*: A URL do provedor de identidade externo.
- *USER_POOL_SELF_SIGN_UP*: Define se os usuários podem se cadastrar no sistema por conta própria. Quando definido como *true*, o auto-cadastro será permitido e o cliente *web* apresentará uma tela de cadastro. Caso contrário, o auto-cadastro será desativado e o cliente *web* esconderá a tela de cadastro. O valor padrão é *false*. Vale lembrar que usuários auto-cadastrados recebem uma permissão de usuário pendente e precisam ser aprovados por um administrador antes de poderem acessar o sistema.
- *NEW_RELIC_LAMBDA_INSTRUMENTATION*: Define se a instrumentação do *New Relic* deve ser ativada para as funções *Lambda*. Quando definido como *true*, a instrumentação será ativada. Caso contrário, a instrumentação será desativada. O valor padrão é *false*. Caso essa variável seja definida como *true*, as seguintes variáveis de ambiente também deverão ser definidas:

  - *NEW_RELIC_ACCOUNT_ID*: O ID da conta do *New Relic*.
  - *NEW_RELIC_TRUSTED_ACCOUNT_KEY*: Caso a conta do *New Relic* seja gerenciada por outra conta do *New Relic*, o ID da conta gerenciadora, caso contrário, o valor deve ser o mesmo que o *NEW_RELIC_ACCOUNT_ID*.
  - *NEW_RELIC_LICENSE_KEY*: A chave de licença do *New Relic*.

- *CLIENT_CUSTOM_DOMAIN*: Define se o cliente *web* deve ser servido em um domínio personalizado. Quando definido como *true*, o cliente *web* será servido em um domínio personalizado. Caso contrário, o cliente *web* será servido em um domínio padrão. O valor padrão é *false*. Caso essa variável seja definida como *true*, as seguintes variáveis de ambiente também deverão ser definidas:

  - *CLIENT_CUSTOM_DOMAIN_NAME*: O nome do domínio personalizado. Exemplo: *virtual-lab.example.com*.
  - *CLIENT_CUSTOM_DOMAIN_CERTIFICATE_ARN*: O ARN do certificado do domínio personalizado criado manualmente no *AWS Certificate Manager* (AMAZON WEB SERVICES, 2024n). Essa é o único recurso que deve ser criado manualmente.

- *DOCS_CUSTOM_DOMAIN*: Define se o site de documentação deve ser servido em um domínio personalizado. Quando definido como *true*, o site de documentação será servido em um domínio personalizado. Caso contrário, o site de documentação será servido em um domínio padrão. O valor padrão é *false*. Caso essa variável seja definida como *true*, as seguintes variáveis de ambiente também deverão ser definidas:

  - *DOCS_CUSTOM_DOMAIN_NAME*: O nome do domínio personalizado. Exemplo: *docs.virtual-lab.example.com*.
  - *DOCS_CUSTOM_DOMAIN_CERTIFICATE_ARN*: O ARN do certificado do domínio personalizado criado manualmente no *AWS Certificate Manager* (AMAZON WEB SERVICES, 2024n). Caso o certificado utilizado no cliente *web* também suporte o domínio da documentação, essa variável pode receber o mesmo valor da variável *CLIENT_CUSTOM_DOMAIN_CERTIFICATE_ARN*.

Uma forma de indicar quais credenciais de acesso à AWS o sistema deve utilizar para fazer a implantação é inserir as seguintes variáveis de ambiente no arquivo *.env*:

- *AWS_ACCESS_KEY_ID*: O ID da chave de acesso da AWS.
- *AWS_SECRET_ACCESS_KEY*: A chave de acesso secreta da AWS.

3.6.3 Comando de implantação

Nesse ponto, todos os pré-requisitos foram atendidos e o sistema está pronto para ser implantado. Para isso, basta executar o seguinte comando na raiz do projeto, substituindo *<stage>* por um nome de ambiente desejado (por exemplo, *production* ou *staging*):

```
# npm run deploy - -stage <stage>
```

Esse comando irá provisionar todos os componentes de infraestrutura necessários na AWS, bem como implantar os serviços de *backend*, o cliente *web* e o site da documentação. O processo pode levar alguns minutos na primeira execução, já que tudo será criado do zero.

Com o processo concluído, uma mensagem de conclusão será exibida no terminal, contendo as URLs de acesso ao cliente *web* e ao site de documentação. O sistema estará disponível para ser acessado pelos usuários finais.

Caso uma mudança tenha sido feita no código fonte do sistema e seja necessário atualizar a implantação, basta executar o comando de implantação novamente. Apenas os componentes afetados pela mudança serão atualizados, o que torna o processo mais rápido.

4 RESULTADOS

Trata-se de um sistema complexo que foi modelado levando em consideração aspectos reais da utilização de software em cenários de uso reais. O projeto foi desenvolvido de forma consistente e replicável, seguindo as boas práticas de desenvolvimento de software e utilizando tecnologias modernas e de mercado.

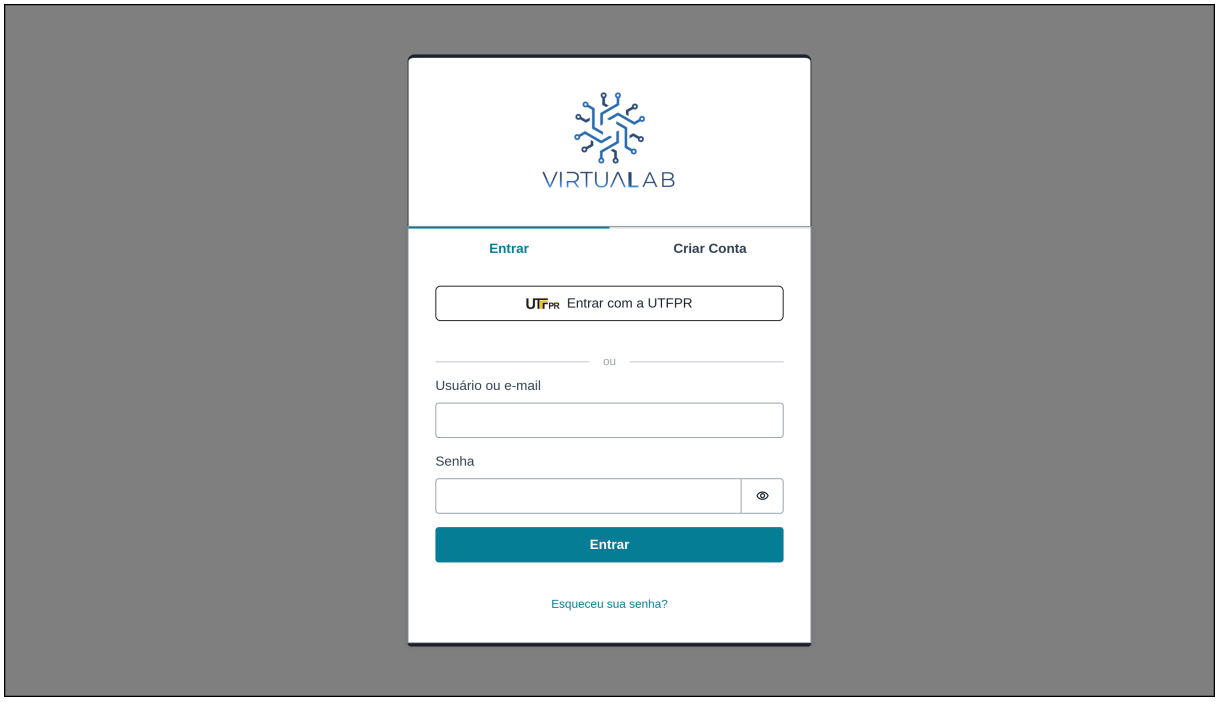
O principal resultado deste trabalho é um artefato completo e funcional que pode ser utilizado pela UTFPR para gerenciar instâncias de máquinas virtuais na AWS de forma segura e eficiente, bem como replicado e configurado para diferentes instituições de ensino com interesse em utilizar sistemas de VDI em nuvem pública para fins acadêmicos.

A demonstração completa dos resultados obtidos nesse trabalho se da através da apresentação do sistema em funcionamento pelo ponto de vista do usuário final. Sistema este que foi implementado e testado para satisfazer os requisitos e princípios de modelagem definidos na seção 3.2. Além disso será apresentada uma estimativa de custos para a utilização do sistema em diferentes cenários de uso.

4.1 O sistema em funcionamento

Ao acessar o cliente *web* através de um navegador pela primeira vez o usuário é redirecionado para a página de autenticação, como apresentado na Figura 17. Nesse momento ele pode inserir suas credencias de acesso se já possuir uma conta criada, ou acionar o botão com o título: `Entrar com a UTFPR`, que o redireciona para a página de autenticação do provedor de identidade externo da UTFPR. Há também a opção de redefinir senha, que oferece um método de recuperação de conta para usuários cadastrados diretamente do sistema.

Figura 17 – Página de autenticação



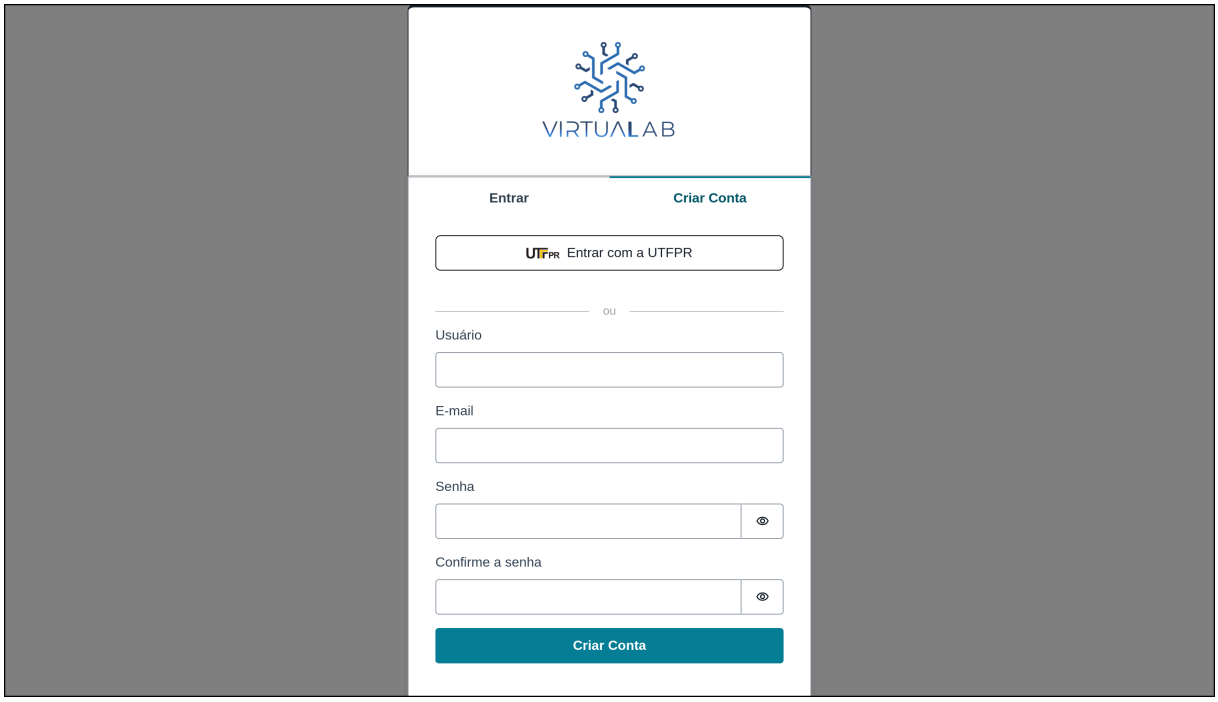
Fonte: Autoria Própria (2024).

No caso onde o usuário não possui uma conta cadastrada e nem credencias de acesso para utilizar o login integrado com a UTFPR, ele deve seguir para a página de cadastro de uma nova conta, como apresentado na Figura 18. Nessa página o usuário deve preencher os campos obrigatórios e acionar o botão `Criar Conta` para que a conta seja criada.

Mesmo podendo se autenticar com uma conta criada manualmente, ao usuário será atribuído o papel de UPEN, que não o permite interagir com os recursos do sistema.

Caso essa página não seja adequada para o ambiente de produção, ela pode ser desativada através das configuração de implantação do sistema. Isso vale também para o botão de integração com o provedor de identidade externo. Porém se as duas funcionalidades forem desabilitadas o sistema só permitirá a criação de usuários através da interface do serviço *AWS Cognito* no painel de administração da AWS.

Figura 18 – Página de cadastro



Fonte: Autoria Própria (2024).

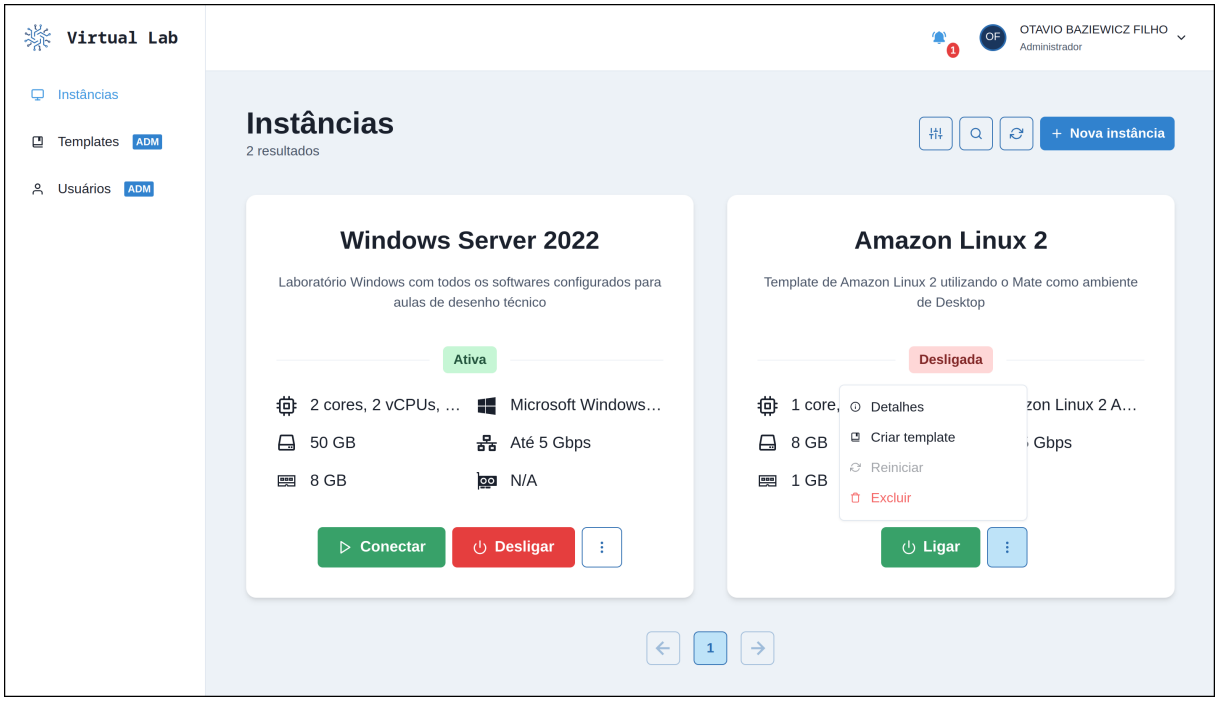
Caso o usuário receba acesso completo ao sistema, a página inicial do cliente *web* será a listagem das instâncias tanto para UCOMs e UADMs, como apresentado na Figura 19.

Nessa página o usuário pode visualizar todas as instâncias que ele possui acesso, bem como realizar as operações de gerenciamento das mesmas.

Caso a instância esteja ligada, o usuário pode desligá-la, reiniciá-la ou acessá-la através dos botões `Desligar`, `Reiniciar` e `Acessar`. Caso a instância esteja desligada, o usuário pode ligá-la ou deletá-la através dos botões `Ligar` e `Excluir` respectivamente.

Ainda nessa página o usuário pode aplicar filtros de busca para encontrar instâncias específicas ordenar as instâncias por diferentes critérios e criar novas instâncias através do botão `Nova Instância`, no canto superior direito da tela.

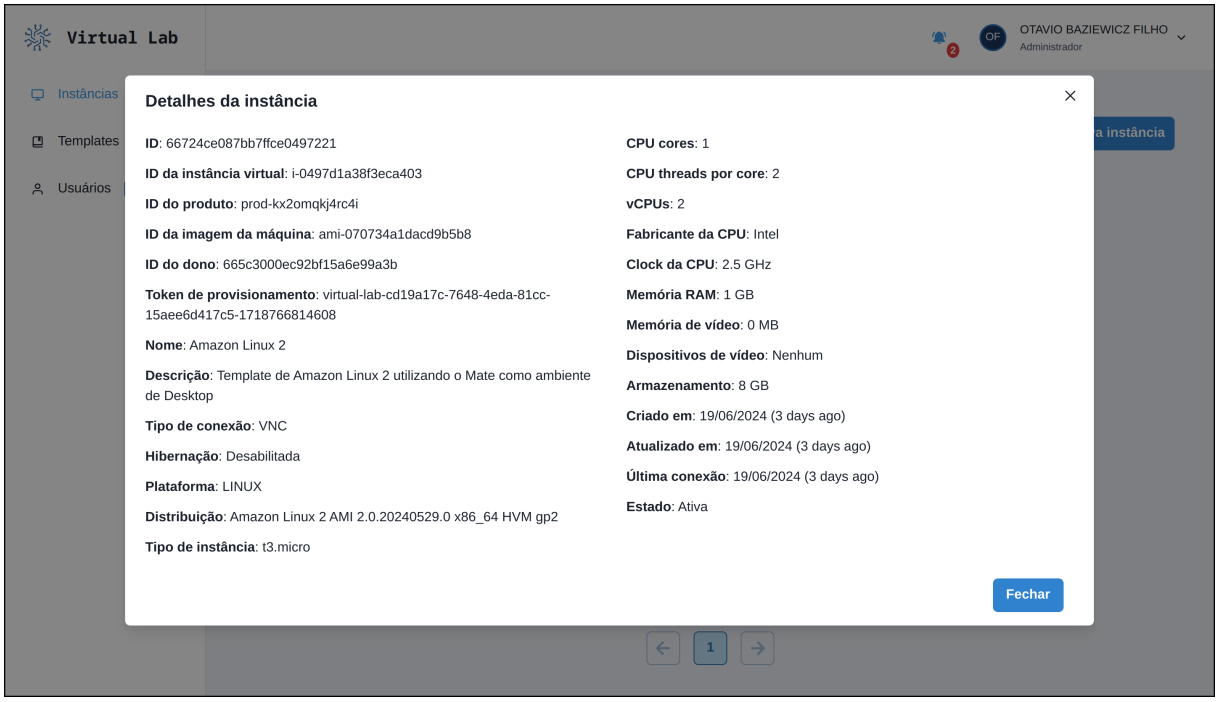
Figura 19 – Página de listagem de instâncias



Fonte: Autoria Própria (2024).

Todos os detalhes de uma instância podem ser visualizados através do botão `Detalhes`, que apresenta ao usuário o modal de detalhes da instância, como pode ser visto na Figura 20. Essas informações não são editáveis, mas permitem que o usuário tenha conhecimento de todos os detalhes da instância de maneira agrupada.

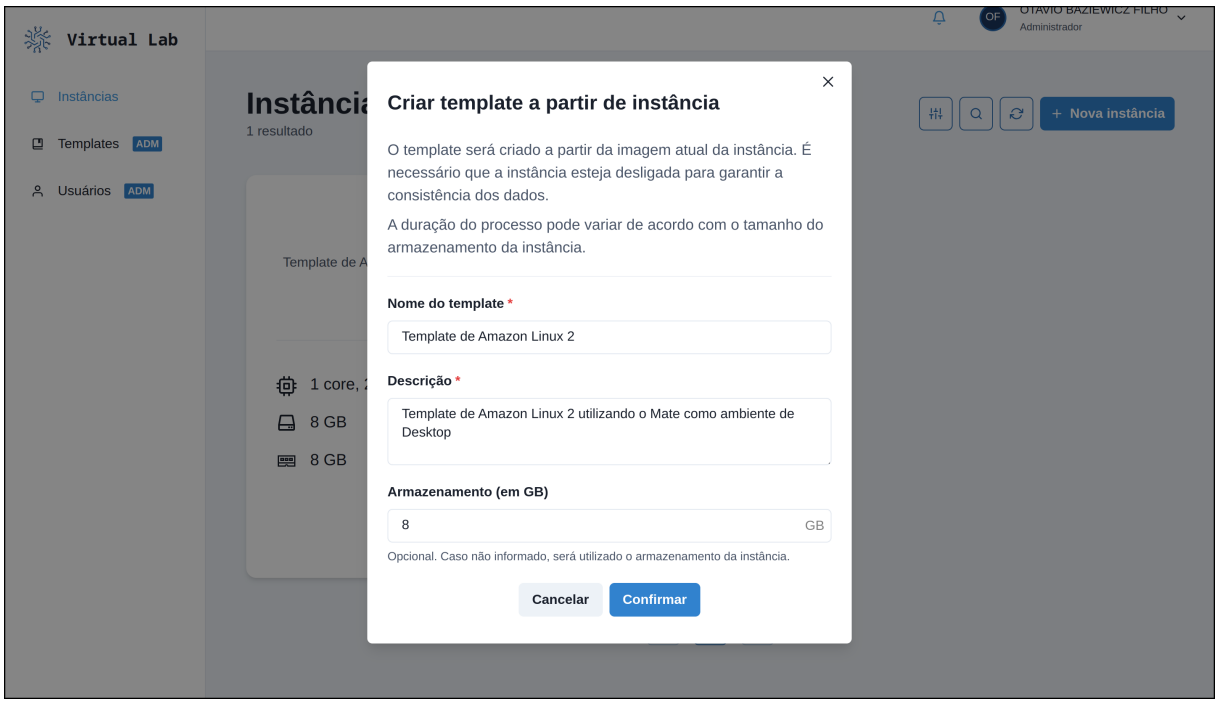
Figura 20 – Modal de detalhes de uma instância



Fonte: Autoria Própria (2024).

Para usuários com o papel de UADM há ainda a opção de de criar um template de instância a partir de uma instância existente, através do botão `Criar Template`. Ao acionar esse botão, o usuário deve preencher os campos obrigatórios na página de criação de template, como apresentado na Figura 21, e então acionar o botão `Criar Template` para confirmar a criação. Nesse momento, o sistema faz uma cópia da instância selecionada e a transforma em um template através de um processo que pode levar alguns minutos, dependendo da quantidade de dados a serem copiados e da quantidade de armazenamento selecionada para o template.

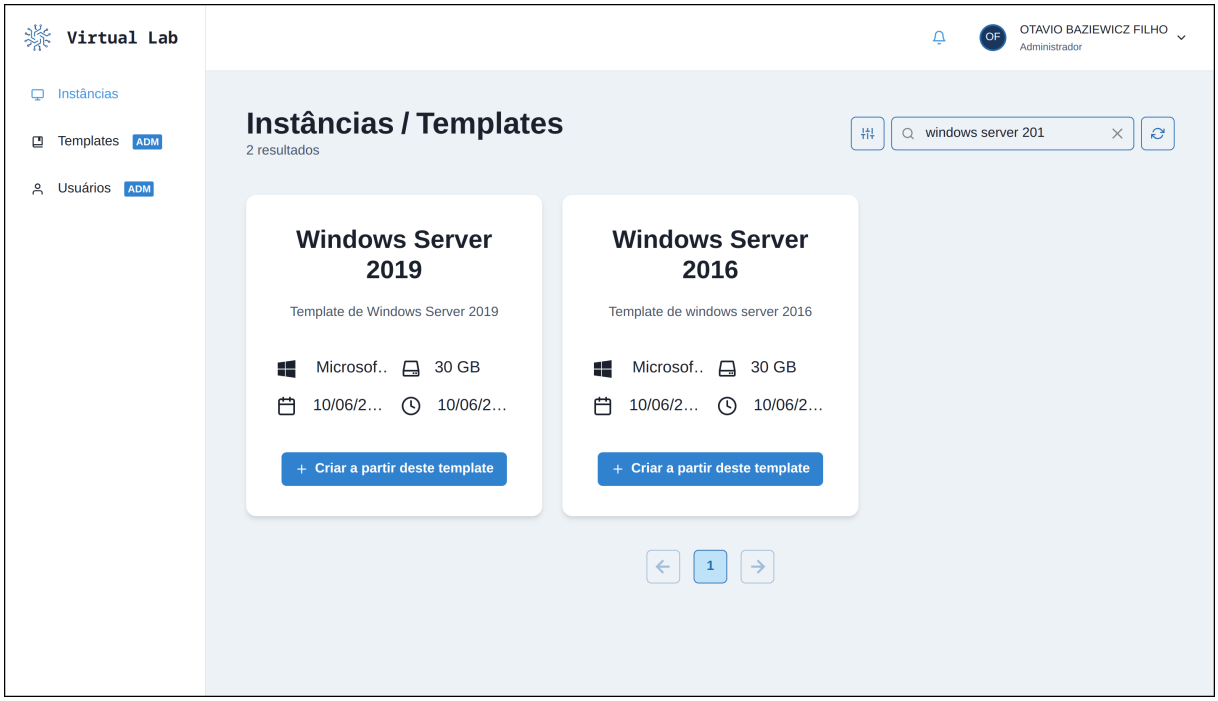
Figura 21 – Modal de criação de template a partir de uma instância existente



Fonte: Autoria Própria (2024).

Se o usuário optar por criar uma nova instância através do botão `Nova Instância`, ele será redirecionado para a página onde todos os templates de instâncias disponíveis são listados, como apresentado na Figura 22. Nessa página o usuário pode selecionar um template escolhido e acionar o botão `Criar a partir deste template`, que apresenta o modal de especificação dos parâmetros de criação da instância, como mostra a Figura 23.

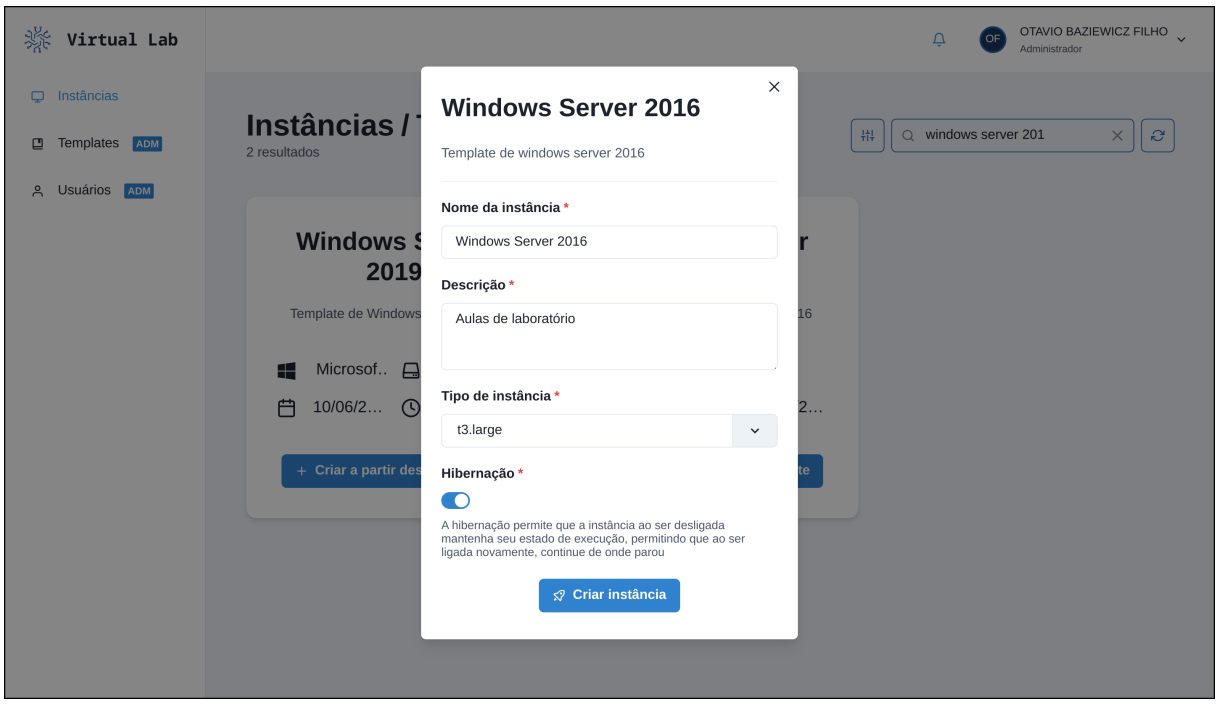
Figura 22 – Página de listagem dos templates de instâncias disponíveis para criação de uma nova instância



Fonte: Autoria Própria (2024).

Na página de especificação dos parâmetros para a criação de uma nova instância, o usuário tem a opção de atribuir um nome e uma descrição para a instância, bem como escolher o tipo de *hardware* da instância e se o recurso de hibernação será habilitado. A lista de tipos de *hardware* disponíveis é dinâmica e é carregada a partir da aplicação de API, onde são mostrados todas opções previamente atribuídas pelo administrador do sistema. Para iniciar o processo de criação o usuário pode acionar o botão `Criar Instância` que entra em modo de carregamento e em seguia redireciona o usuário para a listagem de instâncias, apresentada pela Figura 19.

Figura 23 – Modal de criação de uma nova instância



Fonte: Autoria Própria (2024).

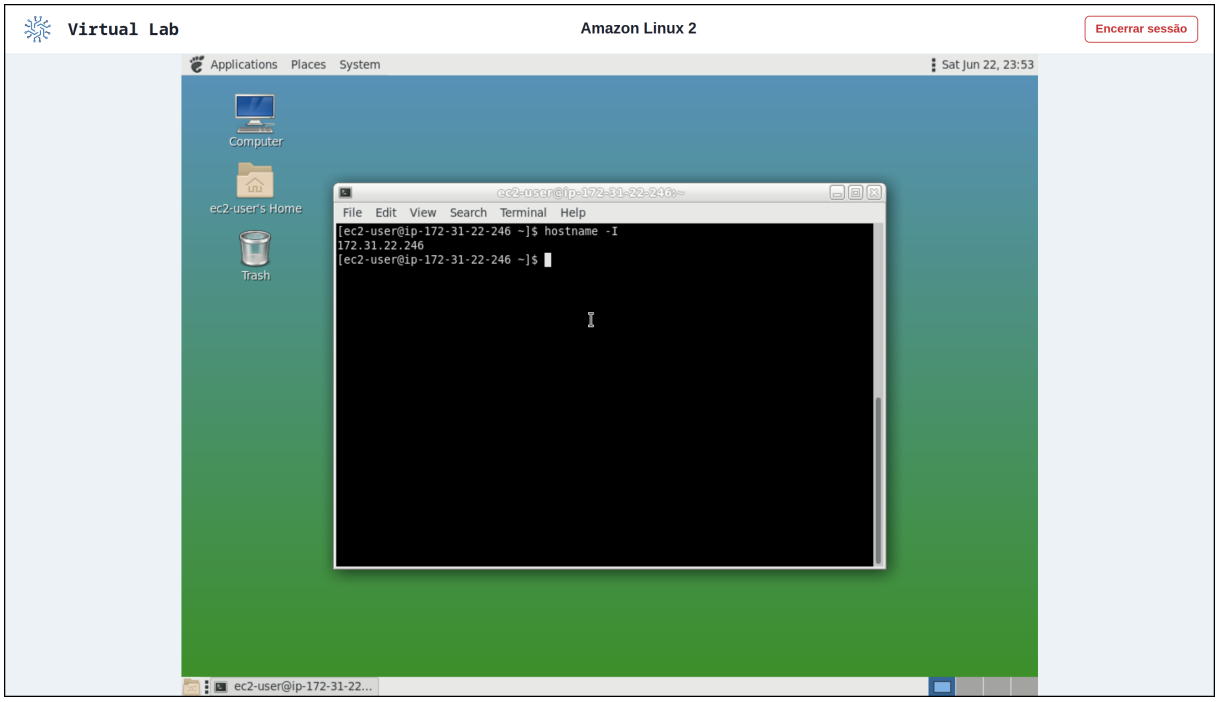
Enquanto a instância estiver sendo criada o usuário recebe notificações em tempo real sobre o andamento do processo, como apresentado na Figura 24. Essas notificações são exibidas em através de um botão com símbolo de sino que é fixado no canto inferior direito da tela e permite que o usuário marque as notificações como lidas ou as exclua.

Figura 24 – Componente de notificações

Fonte: Autoria Própria (2024).

Quando o processo de configuração da instância é concluído, o usuário pode iniciar uma conexão através do botão `Conectar` mostrado na Figura 19. Ao acionar esse botão o usuário é redirecionado para a página de conexão, onde o usuário tem acesso à interface remota da instância, como apresentado na Figura 25. Nessa página o usuário pode interagir com a instância e também voltar para a listagem de instâncias através do botão `Encerrar sessão`.

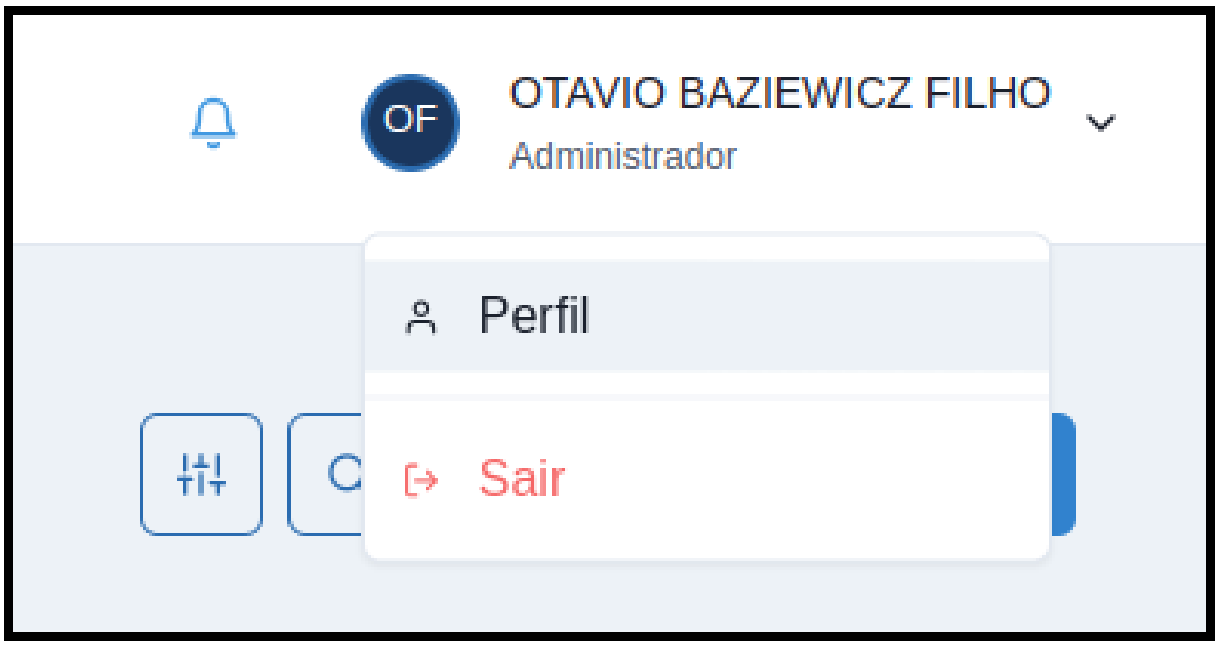
Figura 25 – Página de conexão com instância



Fonte: Autoria Própria (2024).

Novamente na página de listagem de instâncias, o usuário volta a visualizar os outros recursos disponíveis na interface. Para acessar suas informações pessoais, bem como suas cotas de uso do sistema, o usuário pode acionar o botão com seu nome de usuário no canto superior direito da tela, que apresenta os botões `Perfil` e `Sair`, como apresentado na Figura 26 e então acionar o botão `Perfil` para ser redirecionado para a página de perfil do usuário, como apresentado na Figura 27.

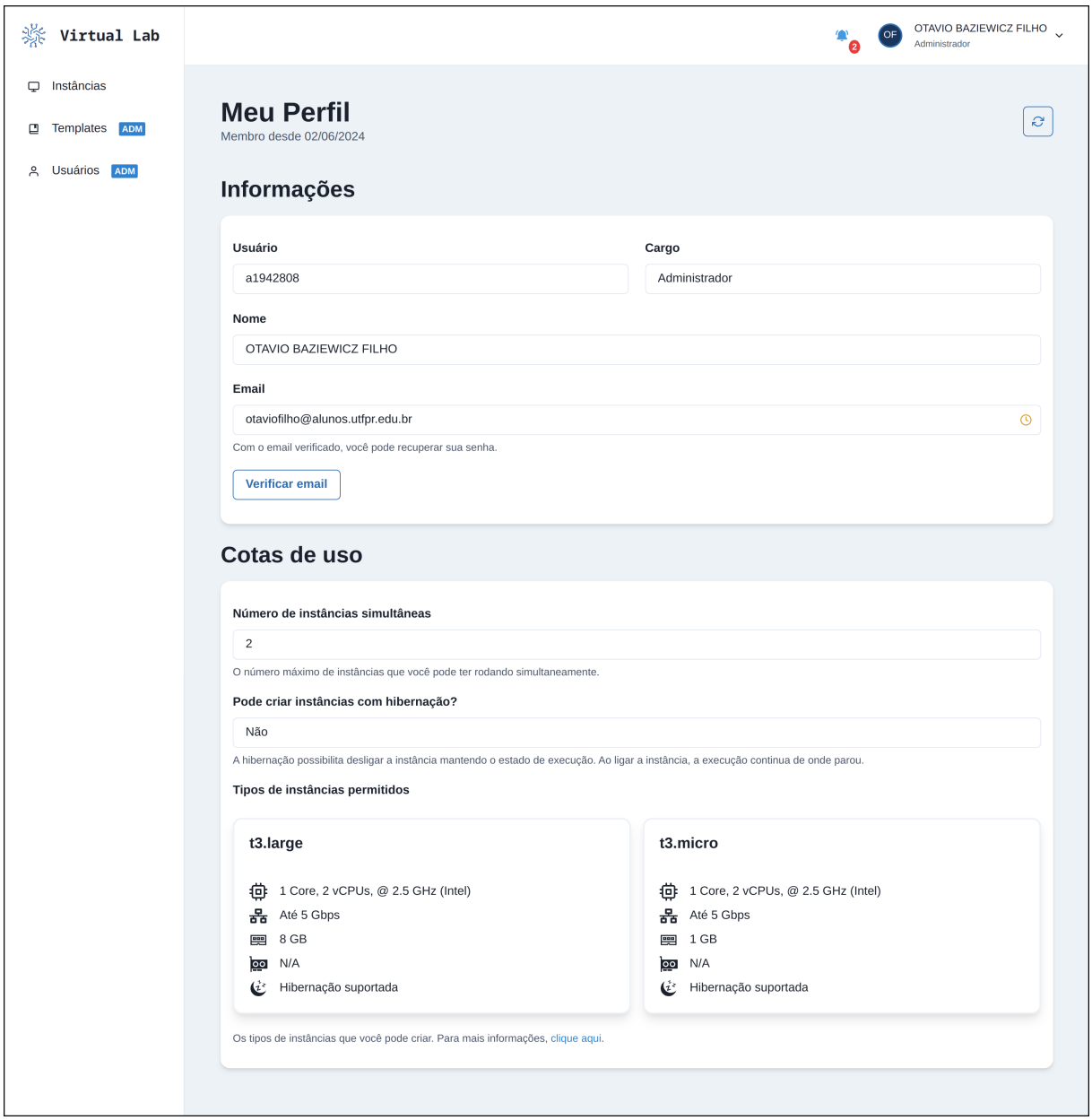
Figura 26 – Menu do usuário



Fonte: Autoria Própria (2024).

Na página de perfil do usuário o usuário pode visualizar suas informações pessoais, cotas de uso do sistema, bem como realizar a verificação de e-mail, que acontece através do envio de um e-mail de para o endereço cadastrado com um código que deve ser inserido no campo habilitado ao acionar o botão `Verificar Email`. É importante salientar que a verificação de e-mail é necessária para que o usuário possa recuperar a senha da conta, caso necessário. Por outro lado, essa funcionalidade não afeta usuários que utilizam o login integrado com o provedor de identidade externo.

Figura 27 – Página de perfil do usuário

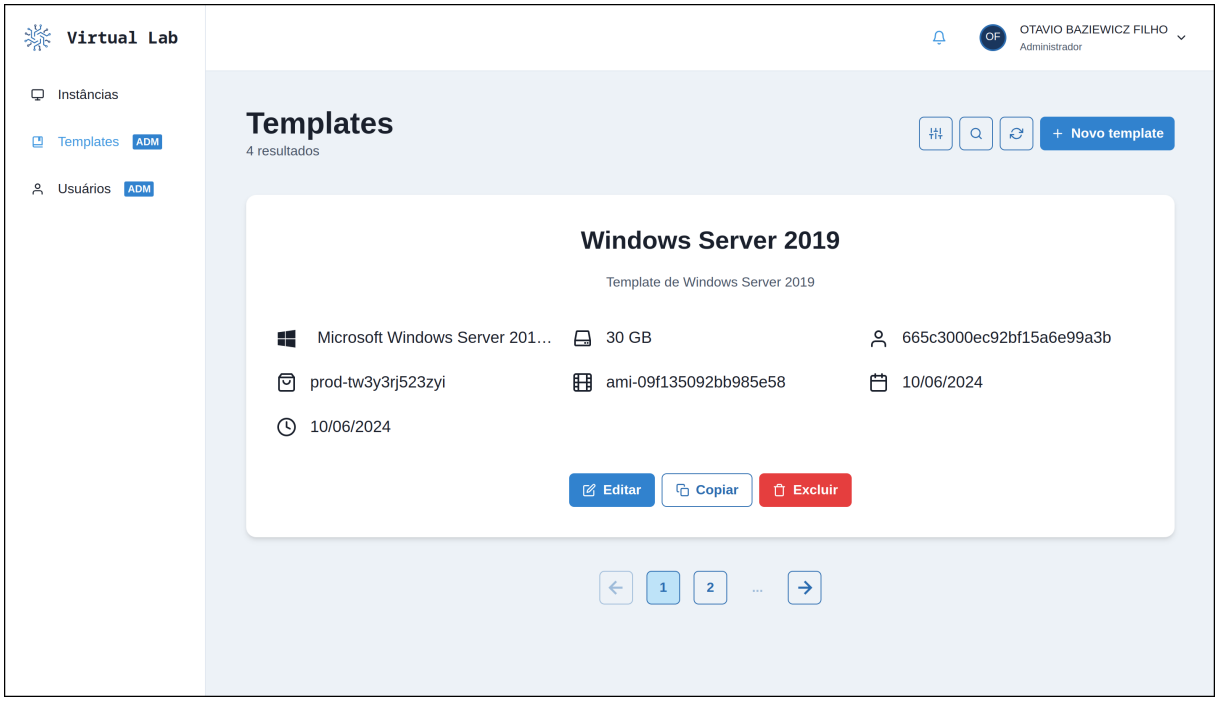


Fonte: Autoria Própria (2024).

Usuários com o papel de UADM possuem a página de gerenciamento de templates, onde podem visualizar todos os templates de instancias disponíveis, bem como deletar templates existentes e criar novos templates a partir de imagens oficiais de sistemas operacionais. Essa página é acessível através do menu lateral, como apresentado na Figura 28.

A listagem de templates é semelhante a listagem de instâncias, onde o usuário pode adaptar os filtros e utilizar a busca textual para encontrar templates específicos. Além disso, o usuário pode editar templates existentes, alterando o nome e a descrição, bem como excluir templates que não são mais necessários.

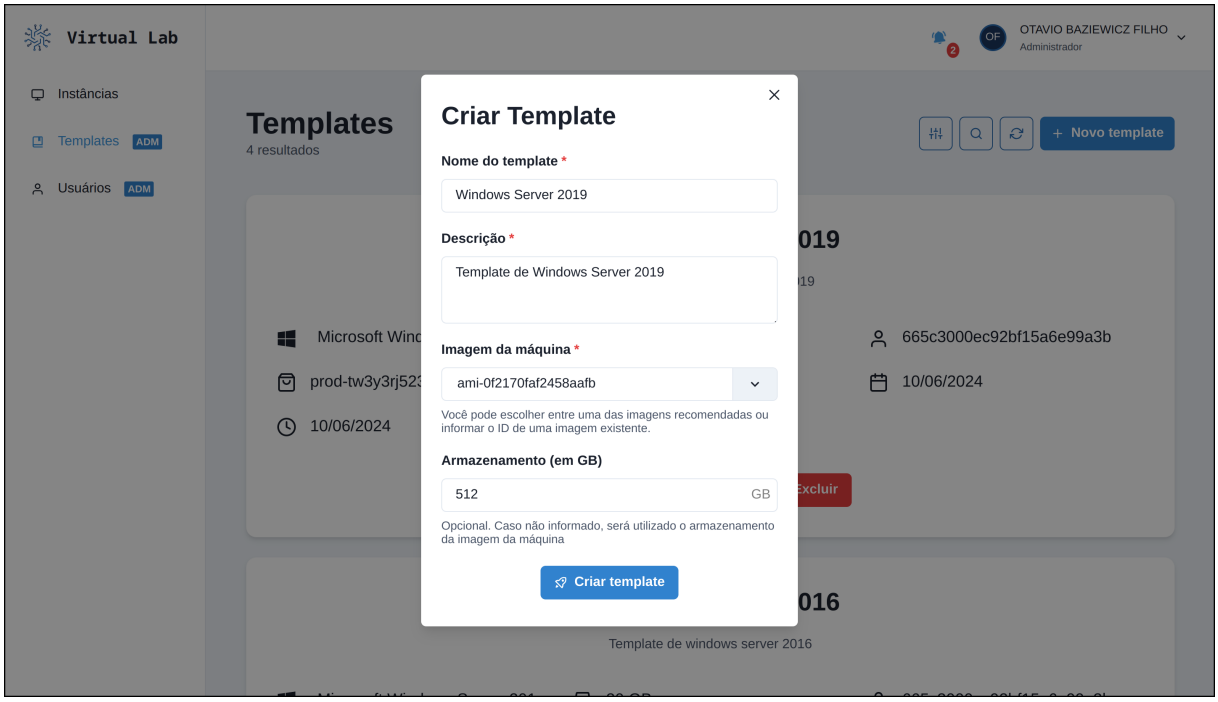
Figura 28 – Página de templates de instâncias



Fonte: Autoria Própria (2024).

Ainda nessa página, quando o usuário aciona tanto o botão `Novo Template` quanto o botão `Copiar` de um template existente, o modal de criação de template é apresentado, como pode ser visto na Figura 29. Nesse modal o usuário deve preencher os campos obrigatórios e escolher uma imagem de sistema operacional customizada ou recomendada pelo componente de escolha de imagem e em seguida acionar o botão `Criar Template` para confirmar a criação, que estará disponível disponível para todos os usuários com o papel de UCOM ou UADM. Diferente da criação de template a partir de uma instância existente, a criação de template a partir de uma imagem existente é instantâneo.

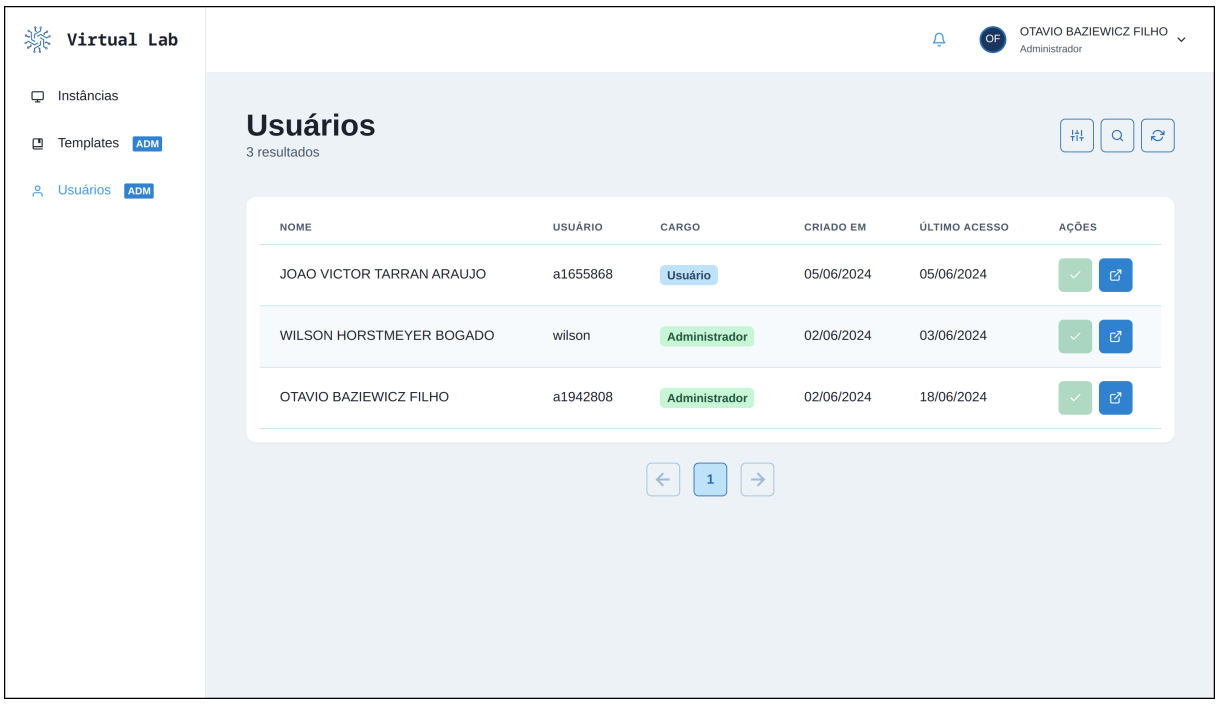
Figura 29 – Modal de criação de template de instância



Fonte: Autoria Própria (2024).

Usuários com papel de UADM possuem ainda a página de gerenciamento de usuários, onde podem visualizar todos os usuários cadastrados no sistema, bom como aprovar usuários pendentes e atualizar as cotas de uso dos usuários. Essa página é acessível através do menu lateral, como apresentado na Figura 30. A listagem de usuários é semelhante a listagem de instâncias e templates, onde o usuário pode adaptar os filtros e utilizar a busca textual para encontrar usuários específicos, com a diferença que a apresentação dos usuários é feita através de uma tabela, onde cada linha representa um usuário e cada coluna uma informação do usuário.

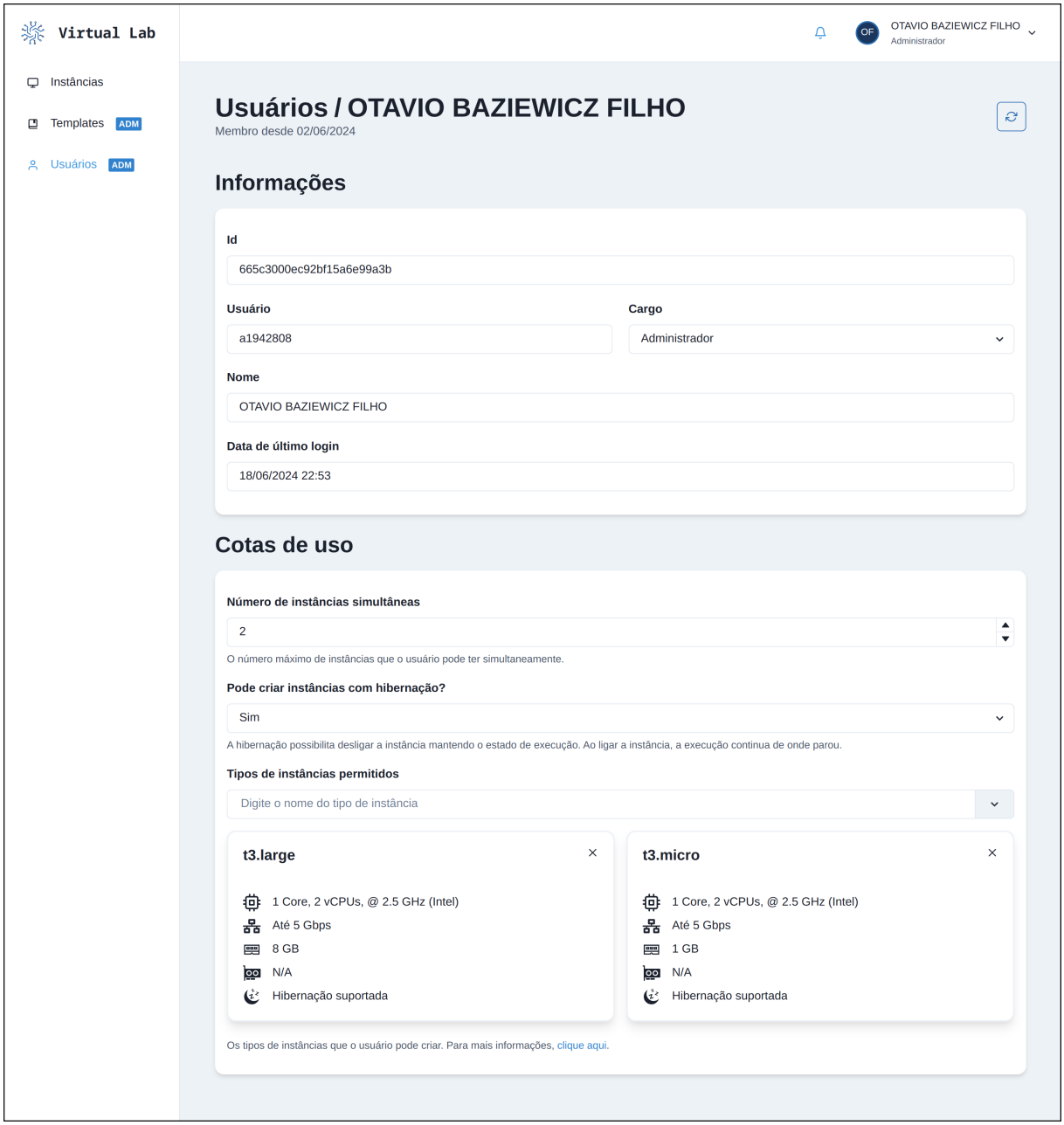
Figura 30 – Página de usuários



Fonte: Autoria Própria (2024).

Ao selecionar um usuário na listagem, agora na página de detalhes do usuário, é possível visualizar todas as informações do usuário, bem como atualizar o papel do usuário e as cotas de uso do sistema. Essa página é apresentada na Figura 31.

Figura 31 – Página de detalhes do usuário



Fonte: Autoria Própria (2024).

4.2 Estimativa de custos

Como parte do projeto, foi realizada uma estimativa de custos para a utilização do sistema em diferentes cenários de uso. A estimativa foi feita com base nos preços de mercado da AWS através da calculadora de preços disponível em seu site oficial.

A primeira estimativa leva em consideração um cenário de uso onde 100 instâncias são utilizadas por 1 hora e 30 minutos por dia, durante 30 dias, com picos de utilização de 5 dias por semana. As instâncias são do tipo `t3.micro` com 2 vCPU e 1GB de memória RAM.

O custo total estimado para esse cenário é de U$ 289.82/mês, ou R$ 1.573,72/mês com a cotação do dólar a R$ 5,43, como apresentado na Tabela 11.

Tabela 11 – Primeira estimativa de custos

Serviço	Custo
Amazon Cognito	R$ 0.75
S3 Standard	R$ 0.12
Data Transfer	R$ 4.50
AppSync API Request	R$ 0.01
Appsync Data Transfer	R$ 0.90
Amazon API Gateway	R$ 1.00
AWS Service Catalog	R$ 0.35
Standard topics	R$ 0.00
AWS Lambda	R$ 5.83
Application Load Balancer	R$ 36.87
Amazon Elastic Container Registry	R$ 0.20
Amazon EventBridge	R$ 1.00
AWS Fargate	R$ 26.00
Amazon EC2	R$ 212.29

Fonte: Autoria própria (2024)

O segundo cenário de uso leva em consideração a utilização apenas dos serviços essenciais para manter o sistema funcionando. Essa estimativa visa entender o custo da infraestrutura em períodos de recesso, onde o sistema deve ficar sem acessos por aproximadamente 1 mês.

O custo total estimado para esse cenário é de U$ 35.13/mês, ou R$ 185,32/mês com a cotação do dólar a R$ 5,43, como apresentado na Tabela 12.

Tabela 12 – Segunda estimativa de custos

Serviço	Custo
Amazon Cognito	R$ 0.00
S3 Standard	R$ 0.12
Data Transfer	R$ 4.50
AppSync API Request	R$ 0.00
Appsync Data Transfer	R$ 0.00
Amazon API Gateway	R$ 0.00
AWS Service Catalog	R$ 0.00
Standard topics	R$ 0.00
AWS Lambda	R$ 0.00
Application Load Balancer	R$ 16.43
Amazon Elastic Container Registry	R$ 0.20
Amazon EventBridge	R$ 0.00
AWS Fargate	R$ 13.88
Amazon EC2	R$ 0.00

Fonte: Autoria própria (2024)

A comparação de custos em dois cenários distintos de uso do sistema, onde um cenário representa um uso com picos diários e o outro um uso mínimo, permite que a instituição de ensino tenha uma visão mais clara dos custos envolvidos na utilização do sistema e possa planejar a utilização do sistema de acordo com suas necessidades e orçamento disponível.

Um ponto observado durante a estimativa de custos é que os serviços de infraestrutura que representam a maior parte dos custos quando o sistema está em sua capacidade mínima de utilização são os serviços que servem de base para o funcionamento do *gateway* de conexão. Por outro lado, quando o sistema está em sua capacidade máxima de utilização, os serviços de infraestrutura representam uma parcela menor dos custos totais, com os serviços de instâncias EC2 representando a maior parte dos custos.

5 CONCLUSÃO

A disponibilidade de recursos computacionais dentro de instituições de ensino superior é muita vezes limitado devido a processos burocráticos e financeiros, o que pode dificultar a disponibilidade de recursos para atividades de ensino e pesquisa. Esse cenário pode ser agravado em situações de emergência, como a pandemia de COVID-19, que forçou a migração de atividades presenciais para o ambiente virtual, sem tempo hábil para adaptações e investimento em infraestrutura para ensino. Em momentos como esse, alunos de baixa renda podem ser prejudicados, pois dependem de seus dispositivos pessoais para cumprir as demandas acadêmicas, o que pode ser um limitante para o acesso a *softwares* específicos e recursos computacionais mais robustos. Levando isso em consideração, o presente trabalho propôs o desenvolvimento de um sistema de gerenciamento de instâncias de máquinas virtuais, com o objetivo de facilitar o acesso a recursos computacionais de forma remota através de um navegador *web*.

Utilizando tecnologias como o Node.js, MongoDB e o Apache Guacamole™ foi possível modelar um sistema robusto e escalável, que permite a criação e gerenciamento de instâncias de máquinas virtuais de forma simples e intuitiva. O sistema desenvolvido permite a criação de instâncias de máquinas virtuais com diferentes configurações de recursos, como quantidade de memória, processamento e armazenamento, além de possibilitar a instalação de diferentes sistemas operacionais. A comunicação entre o sistema e as instâncias gerenciadas é feita através de um *Gateway* de conexão, que permite o acesso remoto às instâncias através de um navegador *web*, sem a necessidade de instalação de *softwares* adicionais. Essa abordagem permite que equipamentos com baixo poder de processamento possam acessar recursos computacionais mais robustos, sendo pagos apenas pelo tempo de uso.

Este trabalho contribui para a comunidade acadêmica e profissional, pois apresenta uma solução para o problema de acesso a recursos computacionais de forma remota, que pode ser utilizada por instituições de ensino superior e empresas que necessitam de recursos computacionais de forma temporária.

O sistema desenvolvido foi construído de forma completa, com perspectivas de escalabilidade e manutenibilidade, o que permite que novas funcionalidades sejam adicionadas de forma simples e rápida. O sistema foi testado em um ambiente de produção, com a criação de instâncias de máquinas virtuais e acesso remoto através de um navegador *web*, comprovando a viabilidade técnica da solução proposta.

Nota-se, portanto, que o trabalho desenvolvido atingiu os objetivos propostos na seção 1.1 com sucesso através da modelagem, implementação e documentação das concepções materializadas no artefato, que pode ser adaptado para diferentes cenários e necessidades. O sistema desenvolvido é uma solução inovadora e de grande relevância para a comunidade acadêmica e profissional. Logo, acredita-se que o trabalho desenvolvido contribui para a disseminação do conhecimento e redução das barreiras tecnológicas e financeiras para o acesso de recursos computacionais robustos.

Entretanto, o projeto ainda possui limitações, principalmente relacionadas à qualidade de infraestrutura do ambiente de utilização do sistema. Dado a natureza dos protocolos de conexão remota utilizados pelo sistema, que dependem de uma troca intensa de dados entre o cliente e o *Gateway* de conexão, a qualidade da conexão de internet pode impactar diretamente na experiência do usuário. Além disso, diversos pontos ainda podem ser aprimorados através de novas funcionalidades e integrações com outros sistemas, como apresentado na seção 5.1.

5.1 Perspectivas futuras

A partir do trabalho desenvolvido, é possível identificar pontos que podem ser aprimorados ou mesmo novas funcionalidades que podem ser implementadas. A seguir, são apresentadas algumas sugestões de trabalhos futuros:

- Implantação de um *proxy* de rede entre as instâncias gerenciadas pelo sistema e a internet, de forma a aplicar filtros de segurança e monitoramento de tráfego. Assim como é feito no ambiente de rede da UTFPR.
- Adição da funcionalidade de compartilhamento de sessão entre usuários, já que o protocolo base de comunicação do Apache Guacamole™ suporta essa funcionalidade.
- Implantação de um serviço de compartilhamento de arquivos em rede entre as instâncias do sistema, com acesso isolado para cada usuário.
- Implementação de um mecanismo para possibilitar o redimensionamento dos recursos de *hardware* das instâncias gerenciadas pelo sistema, como quantidade de armazenamento, memória e processamento.
- Integração com sistemas de ensino à distância, como o *Moodle*.

REFERÊNCIAS

ADEBAYO, S. **Chakra UI Documentation**. [*S.l.*], 2024. Disponível em: https://v2.chakra-ui.com/getting-started.

AMAZON WEB SERVICES. **AWS Lambda Powertools Documentation**. [*S.l.*], 2023. Disponível em: https://docs.powertools.aws.dev/lambda/typescript/latest/.

AMAZON WEB SERVICES. **Amazon API Gateway Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/apigateway/.

AMAZON WEB SERVICES. **Amazon CloudFront Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/cloudfront/.

AMAZON WEB SERVICES. **Amazon CloudWatch Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/cloudwatch/.

AMAZON WEB SERVICES. **Amazon Cognito Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/cognito/.

AMAZON WEB SERVICES. **Amazon EC2 Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/ec2/.

AMAZON WEB SERVICES. **Amazon Elastic Container Registry Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/ecr/.

AMAZON WEB SERVICES. **Amazon Elastic Container Service Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/ecs/.

AMAZON WEB SERVICES. **Amazon EventBridge Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/eventbridge/.

AMAZON WEB SERVICES. **Amazon S3 Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/s3/.

AMAZON WEB SERVICES. **Amazon Simple Notification Service Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/sns/.

AMAZON WEB SERVICES. **AWS Amplify Documentation**. [*S.l.*], 2024. Disponível em: https://docs.amplify.aws/.

AMAZON WEB SERVICES. **AWS AppSync Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/appsync/.

AMAZON WEB SERVICES. **AWS CDK Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/cdk/v2/guide/home.html.

AMAZON WEB SERVICES. **AWS Certificate Manager Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/acm/.

AMAZON WEB SERVICES. **AWS CloudFormation Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/cloudformation/.

AMAZON WEB SERVICES. **AWS Command Line Interface Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/cli/.

AMAZON WEB SERVICES. **AWS Identity and Access Management Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/iam/.

AMAZON WEB SERVICES. **AWS Lambda Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/lambda/.

AMAZON WEB SERVICES. **AWS Prescriptive Guidance - Hexagonal architecture pattern**. 2024. Disponível em: https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/hexagonal-architecture.html.

AMAZON WEB SERVICES. **AWS Service Catalog Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/servicecatalog/.

AMAZON WEB SERVICES. **AWS Systems Manager Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/systems-manager/.

AMAZON WEB SERVICES. **Elastic Load Balancing Documentation**. [*S.l.*], 2024. Disponível em: https://docs.aws.amazon.com/elasticloadbalancing/.

ANOMALY INNOVATIONS. **SST Documentation**. [*S.l.*], 2024. Disponível em: https://docs.sst.dev/.

APACHE SOFTWARE FOUNDATION. **Velocity Engine**. [*S.l.*], 2020. Disponível em: https://velocity.apache.org/engine/2.3/user-guide.html.

APACHE SOFTWARE FOUNDATION. **Apache Guacamole Documentation**. [*S.l.*], 2024. Disponível em: https://guacamole.apache.org/doc/gug/index.html.

AWS. **Modelo de responsabilidade compartilhada**. 2023. Disponível em: https://aws.amazon.com/pt/compliance/shared-responsibility-model/. Acesso em: 19 mai. 2023.

BHOWMIK, S. **Cloud computing**. Cambridge University Press, 2017. ISBN 9781316638101. Disponível em: https://books.google.com.br/books?id=jeTFDgAAQBAJ.

BROWN, S. **The C4 model for visualising software architecture**. [*S.l.*], 2018. Disponível em: https://c4model.com/.

COCKBURN, A. **Hexagonal architecture**. 2017. Disponível em: https://alistair.cockburn.us/hexagonal-architecture/.

COMMONMARK. **Markdown Documentation**. [*S.l.*], 2024. Disponível em: https://spec.commonmark.org/0.31.2/.

DOCKER INC. **Docker Documentation**. [*S.l.*], 2024. Disponível em: https://docs.docker.com/engine/.

ECMA INTERNATIONAL. **ECMAScript Language specification**. [*S.l.*], 2022. Disponível em: https://262.ecma-international.org/13.0/.

EVAN YOU. **Vite Documentation**. [*S.l.*], 2024. Disponível em: https://vitejs.dev/guide/.

INEP. **RESUMO TÉCNICO DO CENSO DA EDUCAÇÃO SUPERIOR 2021**. [*S.l.*], 2021. Disponível em: https://www.gov.br/inep/pt-br/areas-de-atuacao/pesquisas-estatisticas-e-indicadores/censo-da-educacao-superior/resultados. Acesso em: 02 jun. 2023.

META PLATFORMS. **Docusaurus Documentation**. [*S.l.*], 2024. Disponível em: https://docusaurus.io/docs/.

META PLATFORMS. **React Documentation**. [*S.l.*], 2024. Disponível em: https://react.dev/reference/react.

MICROSOFT. **PowerShell Documentation**. [*S.l.*], 2024. Disponível em: https://docs.microsoft.com/en-us/powershell/.

MICROSOFT. **TypeScript Documentation**. [*S.l.*], 2024. Disponível em: https://www.typescriptlang.org/docs/.

MONGODB INC. **MongoDB Atlas Documentation**. [*S.l.*], 2024. Disponível em: https://www.mongodb.com/docs/atlas/.

NAKAZAWA, M.; KOIZUMI, D.; HIRASAWA, S. The influence of qos on e-learning environment under virtual desktop infrastructure. *In*: **The 5th International Conference on Communications, Computers and Applications (MIC-CCA2012)**. [*S.l.*: *s.n.*], 2012. p. 174–178.

NPM INC. **NPM Documentation**. [*S.l.*], 2024. Disponível em: https://docs.npmjs.com/.

OPENJS FOUNDATION. **ESLint Documentation**. [*S.l.*], 2024. Disponível em: https://eslint.org/docs/v8.x/.

OPENJS FOUNDATION. **Jest Documentation**. [*S.l.*], 2024. Disponível em: https://jestjs.io/pt-BR/docs/getting-started.

OPENJS FOUNDATION. **Node.js Documentation**. [*S.l.*], 2024. Disponível em: https://nodejs.org/docs/latest-v18.x/api/index.html.

PRESSMAN ROGER S.; MAXIM, B. R. **Engenharia de Software - 8.ed.** AMGH Editora Ltda, 2016. ISBN 9788580555349. Disponível em: https://books.google.com.br/books?id=wexzCwAAQBAJ.

PRESTON-WERNER, T. **Semantic Versioning**. [*S.l.*], 2023. Disponível em: https://semver.org/.

PRETTIER. **Prettier Documentation**. [*S.l.*], 2024. Disponível em: https://prettier.io/docs/en/.

SCANDURA, C. **Educação remota através do Fire TV Stick e Laboratórios Virtuais com instâncias Amazon EC2 Spot**. 2022. Disponível em: https://aws.amazon.com/pt/blogs/aws-brasil/educacao-remota-atraves-do-fire-tv-stick-e-laboratorios-virtuais-com-instancias-amazon-ec2-spot/. Acesso em: 03 mai. 2023.

SMARTBEAR SOFTWARE. **Stoplight Elements Documentation**. [*S.l.*], 2024. Disponível em: https://docs.stoplight.io/docs/elements/d6a8ba3f3c186-stoplight-elements.

TAURION, C. **Cloud Computing - Computação em Nuvem**. Brasport, 2009. ISBN 9788574524238. Disponível em: https://books.google.com.br/books?id=mvir2X-A2mcC. Acesso em: 17 mai. 2023.

THE LINUX FOUNDATION. **OpenAPI Specification**. [*S.l.*], 2020. Disponível em: https://spec.openapis.org/oas/v3.0.3.

THE OPEN GROUP. **Shell Command Language**. [*S.l.*], 2018. Disponível em: https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html.

ZOD. **Zod Documentation**. [*S.l.*], 2024. Disponível em: https://zod.dev/.

GLOSSÁRIO

adapter é um termo utilizado dentro da arquitetura hexagonal para se referir a uma classe que implementa uma interface utilizada na definição das regras de negócio dentro de um caso de uso. 31, 43, 46

Apache Guacamole™ é um cliente de desktop remoto baseado em navegador que suporta protocolos de desktop remoto como VNC, RDP e *Secure Shell* (SSH). 19, 36, 37, 41, 42, 74, 75

desktop é um ambiente gráfico de trabalho que permite ao usuário interagir com o sistema operacional por meio de uma interface gráfica. 2, 3, 11, 12, 20

EC2 *Spot Instance* é um tipo de instância de máquina virtual oferecida pela AWS que permite ao usuário solicitar capacidade computacional a preços reduzidos em comparação com as instâncias sob demanda. 19

On-Premise é um modelo de implantação de software em que o software é instalado e executado em servidores locais da organização do usuário, em vez de servidores remotos ou em nuvem pública. 11

port é um termo utilizado dentro da arquitetura hexagonal para se referir a uma interface utilizada na definição das regras de negócio dentro de um caso de uso. 31

APÊNDICE A – Relatório de Cobertura de Testes

Este apêndice apresenta o relatório de cobertura de testes dos arquivos importados em casos de teste. A Tabela 13 apresenta a cobertura de linhas, funções e condicionais dos arquivos importados em casos de teste. Esses dados foram obtidos a partir do relatório gerado pelo *framework* de testes Jest, que é utilizado para executar os testes automatizados do projeto.

É importante ressaltar que as estatísticas são calculadas com base nos arquivos que são importados em casos de testes. Portanto, arquivos não importados ficam de fora do cálculo.

O resultado acumulado da cobertura de linhas, funções e condicionais dos arquivos é de 1043/1117 (93.4%), 218/244 (89.3%) e 443/618 (71.7%), respectivamente. Calculados a partir da execução de 131 casos de teste, distribuídos em 26 arquivos.

Tabela 13 – Cobertura dos arquivos importados em casos de teste

Arquivo	Linhas	Funções	Condicionais
__tests__/fixtures/testDatabaseInstanceManager.ts	11/17 (64.7%)	1/3 (33.3%)	0/1 (0.0%)
packages/api/application/auth.ts	25/25 (100.0%)	7/7 (100.0%)	1/4 (25.0%)
packages/api/application/use-cases/instance/auto-turn-instance-off.ts	13/13 (100.0%)	2/2 (100.0%)	7/11 (63.6%)
packages/api/application/use-cases/instance/delete-instance.ts	19/19 (100.0%)	2/2 (100.0%)	9/12 (75.0%)
packages/api/application/use-cases/instance/get-instance-connection.ts	37/37 (100.0%)	2/2 (100.0%)	17/20 (85.0%)
packages/api/application/use-cases/instance/launch-instance.ts	42/44 (95.5%)	2/2 (100.0%)	17/24 (70.8%)
packages/api/application/use-cases/instance/link-launched-instance.ts	23/23 (100.0%)	2/2 (100.0%)	8/11 (72.7%)
packages/api/application/use-cases/instance/list-instances.ts	25/25 (100.0%)	5/5 (100.0%)	12/15 (80.0%)
packages/api/application/use-cases/instance/notify-instance-state-change.ts	25/25 (100.0%)	2/2 (100.0%)	10/13 (76.9%)
packages/api/application/use-cases/instance/reboot-instance.ts	28/28 (100.0%)	2/2 (100.0%)	14/17 (82.4%)
packages/api/application/use-cases/instance/schedule-instance-operation.ts	8/8 (100.0%)	2/2 (100.0%)	5/8 (62.5%)
packages/api/application/use-cases/instance/turn-instance-off.ts	29/29 (100.0%)	2/2 (100.0%)	14/17 (82.4%)
packages/api/application/use-cases/instance/turn-instance-on.ts	29/29 (100.0%)	2/2 (100.0%)	14/17 (82.4%)
packages/api/application/use-cases/instance/unschedule-instance-operation.ts	8/8 (100.0%)	2/2 (100.0%)	5/8 (62.5%)
packages/api/application/use-cases/instance-template/create-instance-template-from-instance.ts	28/28 (100.0%)	2/2 (100.0%)	10/13 (76.9%)
packages/api/application/use-cases/instance-template/create-instance-template.ts	26/26 (100.0%)	2/2 (100.0%)	9/13 (69.2%)
packages/api/application/use-cases/instance-template/delete-instance-template.ts	18/19 (94.7%)	3/4 (75.0%)	6/9 (66.7%)

(continua)

Tabela 13 – Cobertura dos arquivos importados em casos de teste

(continuação)

Arquivo	Linhas	Funções	Condicionais
packages/api/application/use-cases/instance-template/get-instance-template.ts	15/15 (100.0%)	2/2 (100.0%)	6/9 (66.7%)
packages/api/application/use-cases/instance-template/list-instance-templates.ts	15/15 (100.0%)	2/2 (100.0%)	6/10 (60.0%)
packages/api/application/use-cases/instance-template/update-instance-template.ts	21/21 (100.0%)	3/3 (100.0%)	8/12 (66.7%)
packages/api/application/use-cases/misc/list-instance-types.ts	12/12 (100.0%)	2/2 (100.0%)	5/8 (62.5%)
packages/api/application/use-cases/misc/list-recommended-machine-images.ts	12/12 (100.0%)	2/2 (100.0%)	5/8 (62.5%)
packages/api/application/use-cases/user/get-user.ts	18/18 (100.0%)	2/2 (100.0%)	11/14 (78.6%)
packages/api/application/use-cases/user/list-users.ts	13/13 (100.0%)	2/2 (100.0%)	5/8 (62.5%)
packages/api/application/use-cases/user/sign-in-user.ts	14/14 (100.0%)	2/2 (100.0%)	7/10 (70.0%)
packages/api/application/use-cases/user/sign-up-user.ts	22/22 (100.0%)	2/2 (100.0%)	9/12 (75.0%)
packages/api/application/use-cases/user/update-user-quotas.ts	22/22 (100.0%)	4/4 (100.0%)	10/13 (76.9%)
packages/api/application/use-cases/user/update-user-role.ts	19/19 (100.0%)	2/2 (100.0%)	8/11 (72.7%)
packages/api/domain/application-events/instance-connection-ended.ts	6/6 (100.0%)	1/1 (100.0%)	0/0 (100.0%)
packages/api/domain/application-events/instance-launched.ts	8/8 (100.0%)	1/1 (100.0%)	0/0 (100.0%)
packages/api/domain/application-events/instance-state-changed.ts	8/8 (100.0%)	1/1 (100.0%)	0/0 (100.0%)
packages/api/domain/decorators/use-case-execute.ts	11/11 (100.0%)	3/3 (100.0%)	1/1 (100.0%)
packages/api/domain/dtos/application-event.ts	8/8 (100.0%)	2/2 (100.0%)	0/0 (100.0%)
packages/api/domain/dtos/errors.ts	21/23 (91.3%)	7/8 (87.5%)	16/18 (88.9%)
packages/api/domain/dtos/instance-connection-type.ts	2/2 (100.0%)	0/0 (100.0%)	0/0 (100.0%)
packages/api/domain/dtos/instance-platform.ts	2/2 (100.0%)	0/0 (100.0%)	0/0 (100.0%)
packages/api/domain/dtos/instance-state.ts	2/2 (100.0%)	0/0 (100.0%)	0/0 (100.0%)
packages/api/domain/dtos/principal.ts	2/2 (100.0%)	0/0 (100.0%)	0/0 (100.0%)
packages/api/domain/dtos/role.ts	2/2 (100.0%)	0/0 (100.0%)	0/0 (100.0%)
packages/api/domain/dtos/seek-paginated.ts	2/2 (100.0%)	0/0 (100.0%)	0/0 (100.0%)
packages/api/domain/dtos/virtual-instance-type.ts	2/2 (100.0%)	0/0 (100.0%)	0/0 (100.0%)
packages/api/domain/entities/instance-template.ts	31/34 (91.2%)	8/9 (88.9%)	2/9 (22.2%)
packages/api/domain/entities/instance.ts	44/47 (93.6%)	16/17 (94.1%)	16/21 (76.2%)
packages/api/domain/entities/user.ts	33/36 (91.7%)	10/11 (90.9%)	15/21 (71.4%)

(continua)

Tabela 13 – Cobertura dos arquivos importados em casos de teste

(continuação)

Arquivo	Linhas	Funções	Condicionais
packages/api/infrastructure/auth/in-memory-auth.ts	7/7 (100.0%)	2/2 (100.0%)	5/6 (83.3%)
packages/api/infrastructure/config-vault/in-memory-config-vault.ts	4/4 (100.0%)	2/2 (100.0%)	0/1 (0.0%)
packages/api/infrastructure/connection-encoder/guacamole-connection-encoder.ts	18/18 (100.0%)	4/4 (100.0%)	0/1 (0.0%)
packages/api/infrastructure/event-publisher/in-memory-event-publisher.ts	7/9 (77.8%)	4/4 (100.0%)	3/4 (75.0%)
packages/api/infrastructure/instance-repository/in-memory-instance-repository.ts	53/59 (89.8%)	22/23 (95.7%)	36/48 (75.0%)
packages/api/infrastructure/instance-template-repository/in-memory-instance-template-repository.ts	39/45 (86.7%)	15/16 (93.8%)	27/35 (77.1%)
packages/api/infrastructure/logger/in-memory-logger.ts	12/14 (85.7%)	5/7 (71.4%)	1/1 (100.0%)
packages/api/infrastructure/user-repository/in-memory-user-repository.ts	35/52 (67.3%)	12/22 (54.5%)	17/35 (48.6%)
packages/api/infrastructure/virtualization-gateway/in-memory-virtualization-gateway.ts	107/128 (83.6%)	36/41 (87.8%)	66/89 (74.2%)

Fonte: Autoria própria (2024)

APÊNDICE B – Código Fonte do Fluxo de Integração Contínua

O código da Listagem 1 apresenta as etapas que compõem o fluxo de integração contínua do projeto. O arquivo é escrito em *YAML* e é responsável por automatizar a execução de tarefas como instalação de dependências, execução de testes, análise estática de código e envio de relatórios de cobertura de testes para o serviço de terceiros *Codecov*, que é um serviço de acompanhamento de cobertura de testes.

Listagem 1 – Arquivo de configuração do fluxo de integração contínua

```yaml
name: CI

on: ['push', 'pull_request']

concurrency:
    group: '${{ github.workflow }}-
        ${{ github.event.pull_request.number || github.ref }}'
    cancel-in-progress: true

jobs:
    continuous-integration:
    runs-on: ubuntu-latest
    steps:
        - name: Checkout
        uses: actions/checkout@v4
        with:
            fetch-depth: 0

        - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
            node-version-file: '.nvmrc'
            cache: 'npm'
            cache-dependency-path: |
            package-lock.json
            packages/*/package-lock.json

        - name: Install dependencies, typecheck, lint, format, test
        run: |
            npm ci
            npm run typecheck
            npm run lint
            npx prettier --check .
            npx jest --ci --coverage
        env:
            CI: true

        - name: Upload coverage reports to Codecov
        uses: codecov/codecov-action@v4.0.1
        with:
            token: ${{ secrets.CODECOV_TOKEN }}
```

Fonte: Autoria própria (2024).