UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

PAULO RAVI FEIJÃO LEAL

ANÁLISE E MELHORIA DO CÓDIGO DE UM SISTEMA DE ALOCAÇÃO DE SALAS: UMA ABORDAGEM BASEADA EM MÉTRICAS

QUIXADÁ

2023

PAULO RAVI FEIJÃO LEAL

ANÁLISE E MELHORIA DO CÓDIGO DE UM SISTEMA DE ALOCAÇÃO DE SALAS: UMA ABORDAGEM BASEADA EM MÉTRICAS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Jefferson de Carvalho Silva.

QUIXADÁ

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

L472a Leal, Paulo Ravi Feijão.
Análise e Melhoria do Código de um Sistema de Alocação de Salas: Uma Abordagem Baseada em Métricas / Paulo Ravi Feijão Leal. – 2023.
59 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2023.
Orientação: Prof. Dr. Jefferson de Carvalho Silva.

1. Alocação de salas. 2. Desenvolvimento web. 3. Manutenibilidade. I. Título.

CDD 005.1

PAULO RAVI FEIJÃO LEAL

ANÁLISE E MELHORIA DO CÓDIGO DE UM SISTEMA DE ALOCAÇÃO DE SALAS: UMA ABORDAGEM BASEADA EM MÉTRICAS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: _____/_____/_____.

BANCA EXAMINADORA

Prof. Dr. Jefferson de Carvalho Silva (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Victor Aguiar Evangelista de Farias
Universidade Federal do Ceará (UFC)

Prof. Dr. Luís Gustavo Coutinho do Rêgo
Instituto Federal do Ceará (IFCE)

AGRADECIMENTOS

Agradeço a Deus e a minha família, em especial meu pai, José Aurélio Almeida Leal, minha mãe, Ana Valéria Feijão Leal e meu irmão, Levi Feijão Leal, que me apoiaram e foram pacientes durante todos os anos que decorreram o curso, ajudando nos desafios do cotidiano e motivando para que eu pudesse seguir com minha graduação. Além disso, agradeço a minha namorada, Leandra Kelly Alves Lima, pela motivação, bons momentos e incentivos para minha evolução e bem-estar.

Também agradeço aos amigos que me acompanharam durante toda trajetória do curso, sejam esses momentos de estudo árduo ou alegria, experiências nas quais me ajudaram a progredir muito e foram essenciais para minha graduação. Em especial, agradeço ao Pedro Anderson Costa Martins, Jhonattan Nascimento Barbosa, Daniel de Moura Mascarenhas, Gerardo Magela dos Santos Filho e Aurislânia Pereira Batista, pelos momentos de troca de conhecimento e alegria que foram experienciados e que levaremos como lembranças.

Agradeço a UFC Quixadá que proporcionou o compartilhamento de conhecimento de docentes altamente qualificados durante minha graduação e disponibilizou um ambiente propício à evolução dos discentes. Também agradeço ao Prof. Dr. Jefferson de Carvalho Silva por todas as orientações e caminhos que direcionaram para conclusão deste trabalho.

RESUMO

Com a evolução constante da indústria, inovações tecnológicas possibilitaram conexão entre indivíduos de forma remota de qualquer parte do mundo. As universidades também utilizam desses sistemas para auxiliar em atividades do cotidiano acadêmico. Para gerenciamento de alocação de salas é importante que todos os envolvidos no processo estejam cientes das informações e decisões realizadas. Para tal, a utilização de um sistema *web* pode facilitar o cotidiano dos usuários, que podem ter acesso às informações e controlar especificidades do processo de forma remota. Nesse sentido foi desenvolvido parte de um sistema para alocação de salas visando o contexto acadêmico. Esse sistema será refatorado neste trabalho com o intuito de garantir que o mesmo tenha o nível de qualidade adequado para manter uma evolução constante sem grande complexidade e livre de impactos negativos nas funcionalidades já desenvolvidas. Logo, é realizada a avaliação do sistema antes e depois das modificações, assim como a documentação do que é realizado. De acordo avaliação do sistema, no mesmo não foram identificados problemas relacionados a modularidade, mas seus testes estavam com percentual baixo e uma configuração de cobertura incorreta. Após modificações o sistema conseguiu chegar próximo a 100% de cobertura em todos os contextos analisados. Essa melhoria na cobertura dos testes contribui para aumentar a confiabilidade do sistema, garantindo que as funcionalidades sejam consistentes e livres de erros.

Palavras-chave: Alocação de salas; Desenvolvimento *web*; Manutenibilidade.

ABSTRACT

With the constant evolution of the industry, technological innovations have made it possible for individuals to connect remotely from anywhere in the world. Universities also use these systems to assist with daily academic activities. For room allocation management, it is important that all parties involved in the process are aware of the information and decisions made. In this regard, the use of a web-based system can facilitate the daily routines of users, who can access information and control specific aspects of the process remotely. In this sense, a part of a system for room allocation has been developed, focusing on the academic context. This system will be continuously developed in this work in order to ensure that it reaches the appropriate level of quality to maintain constant evolution without significant complexity and free from negative impacts on the already developed functionalities. Therefore, an evaluation of the system is performed before and after the modifications, as well as documentation of what is done. According to the evaluation of the system, no problems related to modularity were identified, but its tests had a low percentage and incorrect coverage configuration. After the modifications, the system was able to achieve close to 100% coverage in all analyzed contexts. This improvement in test coverage contributes to increasing the reliability of the system, ensuring that the functionalities are consistent and error-free.

Keywords: Room allocation; Web development; Maintainability.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AWS	*Amazon Web Services*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
HTML	*HyperText Markup Language*
HTTP	Hypertext Transfer Protocol
IA	*Inteligência Artificial*
IEC	*International Electrotechnical Commission*
IES	*Instituições de Ensino Superior*
ISO	*International Organization for Standardisation*
JSON	*JavaScript Object Notation*
PHP	*Hypertext Preprocessor*
QME	Quality Measure Elements
REST	*Representational State Transfer*
S3	*Simple Storage Serviçe*
SPA	*Single-Page Application*
SQuaRE	*Software product Quality Requirements and Evaluation*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Languagel*

SUMÁRIO

1 INTRODUÇÃO

Assim como citado no trabalho de German *et al.* (2021), com a evolução constante da indústria as inovações para melhorar processos através de sistemas informatizados se tornou essencial para diminuir atrasos, e auxiliar no gerenciamento de atividades da mesma. Muitas instituições se beneficiam de sistemas informatizados tornando seus processos mais eficientes e dessa forma, reduzindo o tempo necessário para realizá-los. A comunicação entre indivíduos também foi impactada com a aplicação soluções tecnológicas, possibilitando a troca de mensagens através de sistemas computadorizados em lugares geograficamente distantes.

Ainda no contexto apresentado, é possível que sejam realizadas trocas de informações com finalidade de alocação de quartos e hotéis através da *internet*. Um exemplo de utilização dessas tecnologias é a desenvolvida por Airbnb (2022), que trabalha com um sistema centralizado intermediando na comunicação entre anfitriões e viajantes, com intuito de facilitar a reserva de acomodações em qualquer lugar do mundo.

As instituições com necessidades de alocação e reserva de salas podem ter dificuldades para organizar e gerenciar todo o processo caso não possuam sistemas para gerir esses recursos. Todas as etapas, desde a escolha até a conclusão da alocação dos recursos, podem enfrentar obstáculos relacionados ao deslocamento físico dos envolvidos, gerenciamento de diversos documentos e verificação de disponibilidade das salas.

De acordo com Singh e Shah (2020), todo o processo pode se tornar muito burocrático e confuso quando a troca de informações entre os envolvidos é afetada por não haver conhecimento suficiente sobre a disponibilidade das salas. Assim como é descrito por Sanchez *et al.* (2019), as informações sobre os equipamentos que estão presentes na sala e sobre a capacidade de indivíduos da mesma também são importantes e podem fazer com que a atividade desejada não ocorra.

Segundo German *et al.* (2021), em *Instituições de Ensino Superior* (IES) o gerenciamento de recursos, como as salas de aula, são de fundamental importância para a realização das atividades dos docentes e discentes. O processo para alocação de tais atividades necessita de documentação e gerenciamento das salas, de forma que não ocorra a sobreposição de alocações ou reservas.

É possível que aplicações tecnológicas possam otimizar o processo de alocação e reservas de salas, oferecendo suporte de gerenciamento das salas através de um sistema centralizado, focado em diminuir o tempo gasto desde a visualização de informações da sala

desejada, até a análise de solicitação. Ambos os atores desse processo podem se beneficiar da utilização de tal sistema, pois, disponibilizando informações sobre as salas e as solicitações, é possível que os mesmos possam de forma remota efetuar solicitações, avaliá-las e até mesmo informar motivos de recusa.

Embora um sistema possa suprir determinadas regra de negócio, o mesmo ainda pode ser inviável do ponto de vista técnico, impossibilitando sua expansão, e limitando todo o potencial de crescimento do mesmo. Através de análises cuidadosas ao nível técnico é possibilitada a identificação de problemas ou possibilidade de melhorias.

Para Wong *et al.* (2011), durante o desenvolvimento de *software* podem ocorrer problemas por diversos motivos como a implementação realizada por um desenvolvedor inexperiente e prazos de entrega curtos. Para ajudar a evitar algum destes problemas, os princípios da modularidade de *software* podem orientar o desenvolvimento *software* para que o mesmo possa evoluir sem enfrentar problemas de dependências. Aplicando tais princípios, módulos podem ser escritos de forma que possam ser mantidos, independentes, rápidos de serem implementados e que possam ter a menor quantidade de códigos sujos possível.

No contexto apresentado, a criação de um sistema para alocação e reserva de salas se torna viável para realização dessas atividades que muitas vezes são realizadas através de um processo que demanda documentação burocrática e deslocamento físico dos envolvidos. Para garantir que tal sistema está sendo desenvolvido de forma que possa transmitir informações corretamente e possibilitar a usabilidade do mesmo, se faz necessária a escolha de métricas para geração de dados mensuráveis. Tais dados são utilizados para validar que o referido sistema está progredindo através dos bons princípios de modularidade e possa ser utilizado sem problemas.

Por tais motivos, o presente trabalho se destina a expansão do desenvolvimento de um sistema de alocação e reservas de salas, que foi previamente implementado pelo autor deste trabalho, presente no repositório *GitHub* do mesmo e descrito no Apendice A. Onde o trabalho referido conta com um sistema *back-end* para servir os dados e um *front-end* para apresentação. Sendo desenvolvido para servir como interface de gerenciamento das atividades necessárias e orientado por métricas de manutenibilidade para mensurar a qualidade de desenvolvimento. Dessa forma, ao se utilizar de análises realizadas no código-fonte, ajustes são realizados com base nos resultados obtidos para melhorar o sistema.

Este trabalho está disposto seguindo a seguinte estrutura: O capítulo 2 aborda os conceitos fundamentais para desenvolvimento do projeto. O capítulo 3 descreve os trabalhos

relacionados, realizando comparativo entre estes e o presente trabalho. Para o capítulo 4 são descritos os procedimentos metodológicos que devem ser seguidos. Já o capítulo 5 apresenta os resultados obtidos e no capítulo 6 são apresentados as considerações finais sobre o trabalho e atividades futuras.

1.1 Objetivos

A presente seção é destinada à apresentação dos objetivos, descrevendo os mesmos de forma geral e dispondo seu escopo em objetivos específicos.

1.1.1 Objetivo geral

Melhorar a referida aplicação desenvolvida, focando na alocação e reserva de salas, por implementações que se adequam às métricas citadas, limitações de arquitetura, implementação de código e estratégias de validação escolhidas.

1.1.2 Objetivos específicos

- Identificar métricas para análise de manutenibilidade.
- Propor soluções arquiteturais e de codificação.
- Desenvolver uma aplicação *web* orientada as métricas previamente definidas.
- Validar a aplicação com ferramentas automatizadas.

2 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresenta conceitos relacionados ao trabalho que estão diretamente ligados a implementação, apresentando modelos de arquitetura e de desenvolvimento de aplicações *web*. Estes são utilizados para o desenvolvimento do sistema de alocação e reserva de salas, servindo como orientação e base para sua expansão.

Os conceitos citados estão distribuídos em quatro subseções, descrevendo suas características e aplicabilidade neste trabalho. Como o foco deste trabalho envolve o desenvolvimento de uma aplicação reserva e alocação de salas, as características citadas foram selecionadas visando o contexto de desenvolvimento *web*. Para a Seção 2.1, é descrito o conceito de Aplicação *Web*, que orienta o desenvolvimento do *front-end* do sistema. Já para a Seção 2.2, o conceito abordado é o de *Application Programming Interface* (API) *Representational State Transfer* (REST), utilizado para orientar o desenvolvimento do *back-end* do sistema. Envolvendo os conceitos apresentados anteriormente, a Seção 2.3 descreve a arquitetura de Microsserviços e como essa é utilizada num escopo mais abrangente da aplicação, conectando seus componentes. A Seção 2.4 apresenta conceitos sobre métricas que servem como base para identificação de formas de mensurar o sistema a ser analisado.

2.1 Aplicação Web

Segundo Gellersen e Gaedke (1999), aplicação *web* ou *WebApp* é o conceito utilizado para nomear aplicações *front-end* que utilizam do *HyperText Markup Language* (HTML) e da adoção geral dos navegadores, sendo multiplataformas, para desenvolver uma aplicação *web* de forma universal. Com isso diversos sistemas operacionais e plataformas podem utilizar de aplicações desenvolvidas para a *web* de forma similar. Por ser uma aplicação centralizada para diversas plataformas, a manutenção do mesmo é menos custosa se comparada a necessidade de manutenção de diversas aplicações *front-end*.

Esse tipo de aplicação utiliza de características comuns no ambiente *web* que envolvem o desenvolvimento de aplicações baseadas em HTML para apresentação dos dados, consumindo os dados de algum servidor e assim caracterizando a comum arquitetura cliente-servidor. Para o presente trabalho é desenvolvido uma aplicação *web* utilizando de um *framework* para auxiliar no desenvolvimento do sistema, este é nomeado *VueJs* e utiliza da característica *Single-Page Application* (SPA) para construir aplicações *web* de página única. Conceitos fun-

damentais de desenvolvimento *web front-end* são considerados para esta aplicação, sendo eles: HTML, *Cascading Style Sheets* (CSS) e *JavaScript*. Estes conceitos brevemente citados servem como base para implementação da camada de apresentação dos dados do presente sistema, e são detalhados mais adiante.

2.1.1 Kit de desenvolvimento Web

Durante o desenvolvimento de aplicações *web* são utilizados diversos tipos de tecnologias, entre elas o *Javascript*, CSS e HTML. Alguns desdes, que já foram citados anteriormente, contribuem para o desenvolvimento de aplicações *web*. Assim como descrito por Berners-Lee e Connolly (1995), o HTML é uma linguagem de marcação que pode ser utilizada para criação de documentos de hipertexto. Esse conceito é independente da plataforma, podendo ser utilizado para apresentações das mais diversas informações, independente do domínio. Junto a essa linguagem de marcação pode ser utilizado o CSS, que segundo Flanagan (2016) é usado para estilização, onde o mesmo pode ser implementado de forma conjunta com o HTML para estilização de documentos e organização de informações. Para a aplicação de lógicas mais complexas é utilizado o *Javascript*, que segundo MDN (2022) é uma linguagem de scripts desenvolvida inicialmente para páginas *web*, mas posteriormente aplicada em diversos outros ambientes. Tal linguagem é utilizada para interações mais complexas na aplicação *web*, sendo responsável por aplicações de lógicas mais complexas e a base para criação de *frameworks* como *VueJs*.

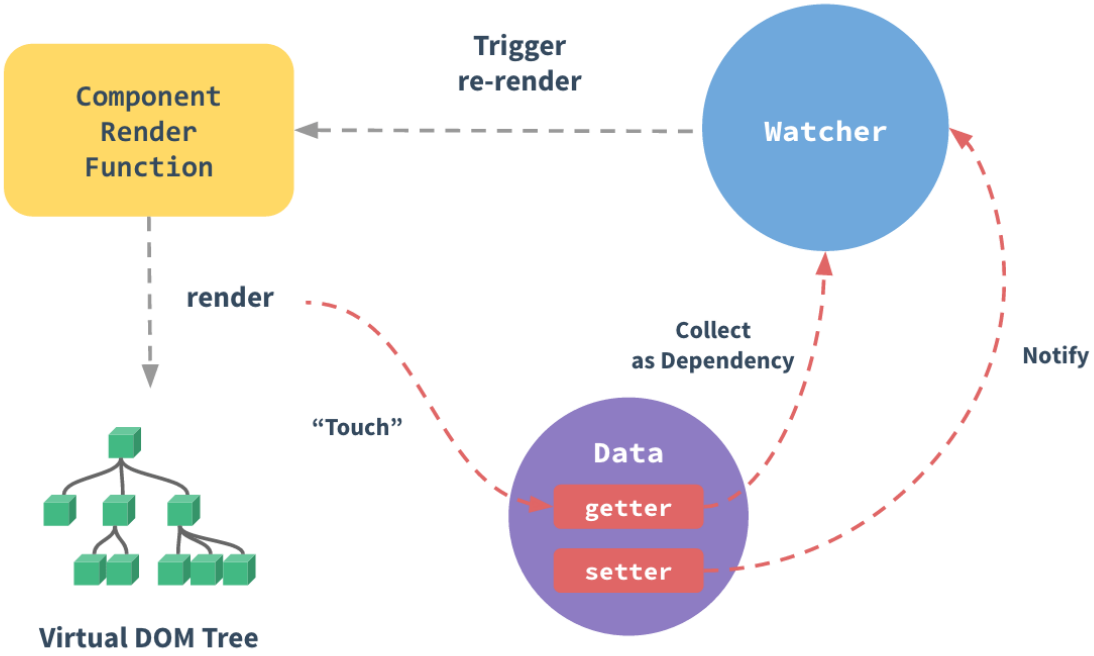
2.1.2 VueJs

Assim como citado anteriormente, o presente trabalho se utilizada do *framework VueJs* parar orientar o desenvolvimento *front-end* do sistema. Segundo a documentação de VueJs.Org (2022a), este é um *framework* progressivo utilizado para a criação de interfaces *web*. Apesar de ser um *framework*, este pode ser utilizado como uma biblioteca para auxiliar na apresentação visual, simples de ser implementado a projetos já existentes, com outras bibliotecas. Por outro lado, uma grande vantagem de utilizar este como *framework* é a possibilidade de criação de aplicações mais complexas utilizando do conceito SPA, que é melhor descrito mais a frente.

Segundo o guia de VueJs.Org (2022b), uma das características importantes desde *framework* é a reatividade discreta, que por sua vez é utilizado para rastrear as modificações que deverão ser realizadas. Em resumo o *framework* trata modelos como objetos *javascript*,

rastreando eles para que atualizações sejam realizadas dependendo das modificações rastreadas nesses objetos.

Figura 1 – *VueJs* - Ciclo de reatividade



Fonte: VueJs.Org (2022b).

O ciclo de reatividade da Figura 1 apresenta como os dados são tratados no *VueJs*. Através dele é possível perceber que os dados assistidos são notificados assim que são realizadas modificações que modificam os dados, da mesma forma estes são formados por outros dados que estão sendo assistidos pelo *framework*. Quando os dados assistidos são alterados, diversos eventos de re-renderização dos componentes que dependem destes são disparados, formando os componentes do *Document Object Model* (DOM) para a renderização de todos os dados componentes visuais dependentes.

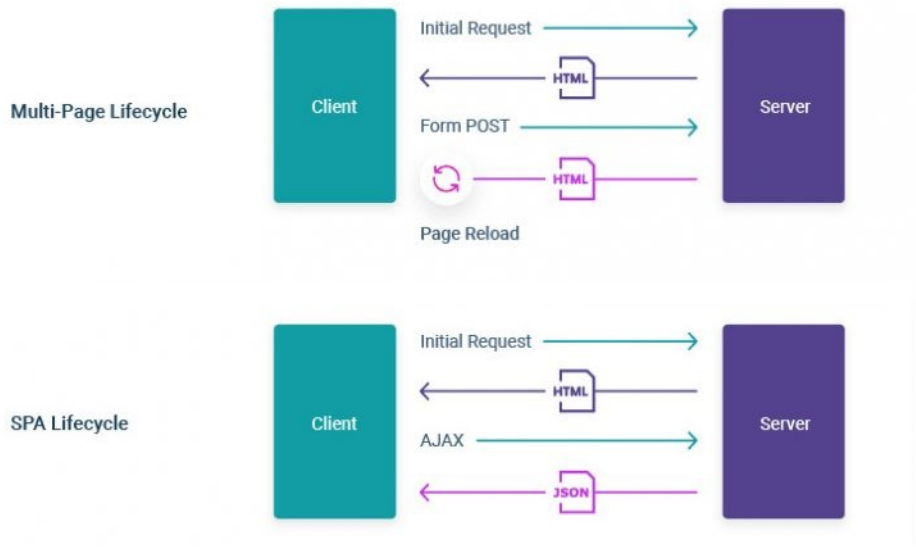
2.1.3 SPA

Assim como descrito anteriormente, *VueJs* utiliza da característica SPA para construção de aplicações *web*, ajudando a criar aplicações mais sofisticadas e complexas. Sobre essa característica a documentação de VueJs.Org (2022c) descreve o mesmo tem suporte a diversas bibliotecas que auxiliam no desenvolvimento de aplicações.

A motivação para o *VueJs* suportar essa característica é descrita como necessidade

de aplicações *front-end* necessitarem de lógicas mais complexas e maior interatividade para apresentação das informações da aplicação. Para isso acontecer se faz necessário utilizar de estratégias mais eficientes para construção de aplicativos, de forma que a arquitetura *web* desenvolvida possa controlar páginas inteiras manipulando e atualizando dados sem necessidade de recarregá-las. O comportamento descrito é característico de diversas aplicações mais complexas é nomeado como SPA, ou seja, aplicativo de página única.

Figura 2 – Single Page Application vs Multi Page Application



Fonte: Oktriwina (2020).

Para aplicações *web* que utilizam de diversas páginas, é comum o carregamento de uma nova página ao haver a necessidade de apresentação de dados presentes na página apresentada. Assim como citado no parágrafo anterior, para a criação de páginas mais complexas que necessitem de maior interatividade, o modelo de múltiplas páginas não se torna eficiente, de forma que são necessários diversos recarregamentos para apresentação de novos dados. Como é apresentado na Figura 2, o SPA possui um ciclo de vida que não requer o recarregamento das páginas para atualização das informações. Ao realizar uma nova requisição a aplicação aguarda os dados requisitados, e logo após a mesma se encarregada de atualizar os dados afetados, assim como os dependentes deste.

2.2 API Rest

Assim como foi descrito anteriormente, em aplicações *web* é comum o uso de arquiteturas no formato cliente-servidor, ou seja, possuam aplicações isoladas para apresentação

dos dados e uma aplicação para servir os dados requisitados do domínio. A aplicação *back-end*, que muitas vezes é armazenado em um servidor, pode definir padrões para comunicação com a mesma, descrevendo como podem ser implementados requisições a esta e como ela deve responder.

Dentre diversas definições de interface o presente trabalho utiliza do conceito de API REST para trabalhar com o *back-end* da aplicação. De acordo com RedHat (2020), API REST, que pode ser nomeada de API RESTful, define interfaces para a programação de aplicações. Estas devem estar de acordo com as definições da arquitetura REST para possibilitar a integração com aplicativos *web*. Ao realizar a implementação da aplicação *back-end* seguindo padrões de API REST, o presente sistema possibilita a busca de dados para apresentação na aplicação *front-end*. Por tanto, o *front-end* o sistema deve se comunicar com o *back-end* seguindo os padrões definidos e esperados dos conceitos de API REST, que por usa vez é desenvolvida com *Node.Js*.

2.2.1 API

De acordo com AWS (2022), uma API pode ser traduzida como Interface de Programação de Aplicação, e funciona como um mecanismo que reuni um conjunto de definições e protocolos para que os *softwares* interessados possam se comunicar transferindo dados. Como uma interface trabalha facilitando e padronizando a forma de comunicação entre as partes interessadas, é comum a documentar a mesma descrevendo como ambas partes devem requisitar e/ou enviar dados. Elas podem ser consideradas contratos, onde cada parte interessada deve obedecer às regras impostas, assinando-os para que a comunicação possa acontecer.

Existem diversas categorias de padrões arquiteturais que podem ser utilizados para definições de API entre elas são as seguintes:

- **SOAP**: APIs que utilizam deste padrão realizam a comunicação entre as partes interessadas em usar objetos *eXtensible Markup Languagel* (XML). Essa é uma categoria de API que foi substituída por outras mais flexíveis, sua sigla significa *Simple Object Access Protocol* (Protocolo de Acesso a Objetos Simples).

- **RPC**: as APIs que utilizam das definições do *Remote Procedure Calls* (Chamadas de Procedimento Remoto), utilizam de funções/procedimentos para realizar a comunicação de envio/recebimento de dados.

- **WebSocket**: essa categoria de API utiliza de objetos no padrão *JavaScript Object*

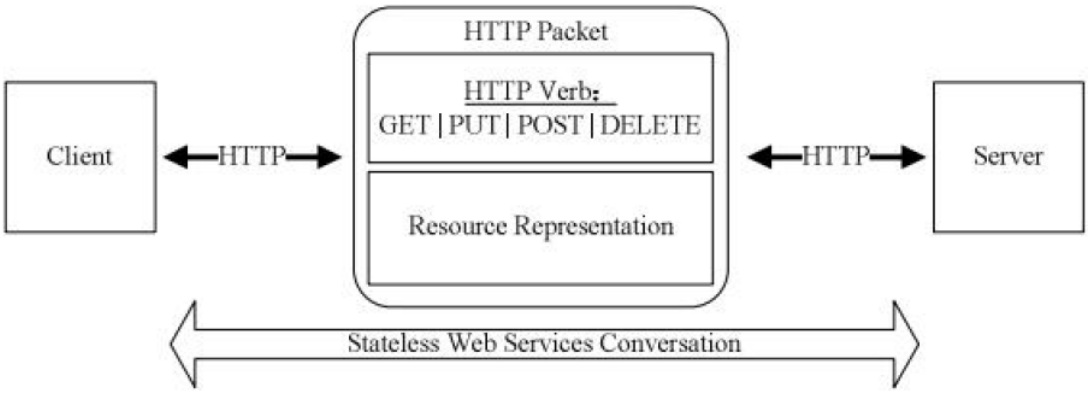
Notation (JSON) para enviar/receber os dados necessários entre as partes interessadas. Como esse padrão necessita que os usuários estejam conectados para que a comunicação seja realizada, o envio de dados é realizado em retornos de chamada.

- **REST**: este padrão traz as definições de comunicação abordadas neste trabalho. Seu benefício é a possibilidade de criação de APIs mais flexíveis onde as partes interessadas se comunicam solicitando dados, e parte retentora dos dados realiza uma série de processamentos para retornar a mensagem desejada. Esta é melhor descrita na subseção mais a frente.

2.2.2 REST

Assim como citado anteriormente, o padrão REST é adotado para o presente sistema. O mesmo é descrito pela MDN (2021) como padrão de projeto arquitetural para sistemas distribuídos. Alguns dos benefícios da utilização desse padrão é a eficiência, confiabilidade e escalabilidade que são possíveis de serem conseguidas ao desenvolver um sistema que obedeça a esse padrão.

Figura 3 – Arquitetura de serviços *Web RESTful*



Fonte: Feng *et al.* (2009).

Assim como Feng *et al.* (2009) descreve na Figura 3, um recurso no contexto da arquitetura REST pode ser qualquer coisa que possa ser referenciado no domínio da aplicação. A implementação das definições REST junto ao protocolo Hypertext Transfer Protocol (HTTP) permite a manipulação dos recursos através de diversos verbos HTTP. Entre esse verbos existem os mais comuns, que segundo Fielding e Reschke (2014) são utilizados para:

- ***GET***: método para busca e recuperação de dados relacionados a um recurso.
- ***POST***: usado para manipulação de dados referente a um recurso identificado.
- ***PUT***: utilizado para atualização ou criação de recurso caso o mesmo não exista.

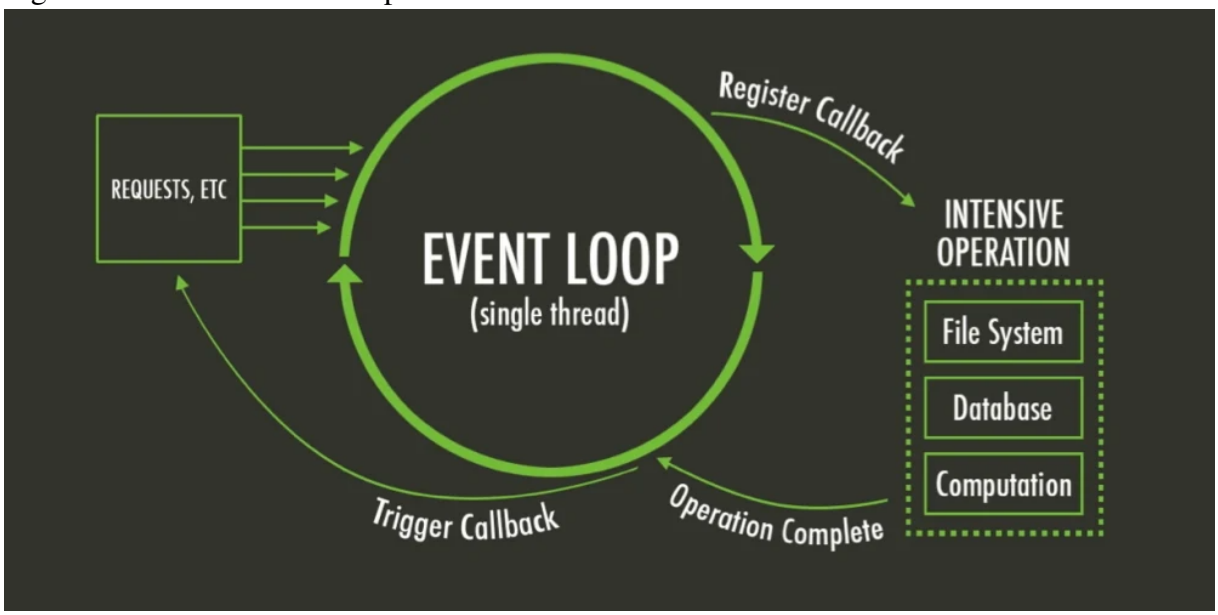
- ***DELETE***: usado para remoção de um recurso existente.

Os conceitos descritos sobre REST aplicados juntamente a uma API são implementados de forma que possam padronizar a forma de acesso aos recursos da aplicação. O sistema utiliza na interface disponibilizada no padrão citado para que a comunicação entre *front-end* e *back-end* possa acontecer se beneficiando das características apresentados sobre a arquitetura descrita.

2.2.3 NodeJs

Segundo Nodejs.Org (2022) o *Node.Js*, foi desenvolvido para criação de aplicações *web* escaláveis, para isso mesmo utiliza da execução de código *Javascript* em tempo de execução. Além disso, é descrito que o mesmo é utilizado para chamadas de forma assíncrona e orientada a eventos. Um dos benefícios de utilizar *NodeJs* se dá por conta de sua característica que não bloqueio de *threads*, algo que geralmente tornaria ineficiente a utilização do mesmo caso viesse acontecer.

Figura 4 – NodeJs Event Loop



Fonte: Santos (2019).

Como é possível ver na Figura 4, para realizar o gerenciamento dos processos é apresentado o *loop* de eventos, utilizado em tempo de execução para gerenciar a execução de *scripts*. Dessa forma o *NodeJs* sai do *loop* assim que não há mais nenhum evento para ser processado. Como o *NodeJs* utiliza da *engine V8* para execução do *Javascript*, assim como no navegador *chrome*, ele se torna capaz de interpretar *Javascript* por meio dessa ferramenta.

O sistema a ser desenvolvido utiliza das características de API REST mencionadas anteriormente, desenvolvida em *Node.Js*, utilizando dos benefícios citados acima sobre essa tecnologia.

2.3 Microsserviços e Serviços em Nuvem

Segundo AWS Docs (2022b), os microsserviços são utilizados como uma abordagem arquitetural quando é necessário realizar a implementação de sistemas independentes que expõem sua interface de comunicação através de uma API. Tais serviços têm características de serem pequenos, realizando apenas o que foi proposto por eles, sendo autossuficientes. Essas aplicações têm outra característica importante que impacta na escalabilidade e introdução de novas funcionalidades no produto final, pois como são independentes, podem ser integradas ao produto sem necessidade de grandes mudanças no mesmo. O *back-end* desenvolvido deste trabalho é implementado como uma API, dessa forma o mesmo pode ser focado apenas no processo de reserva e alocação de salas, disponibilizando uma interface com padrões REST, sendo bem semelhante ao funcionamento de um microsserviço. É importante perceber que o este trabalho utiliza de conceitos bastante utilizados em microsserviços, mas não terá como foco a implementação de todos os conceitos que caracterizam esse tipo de arquitetura.

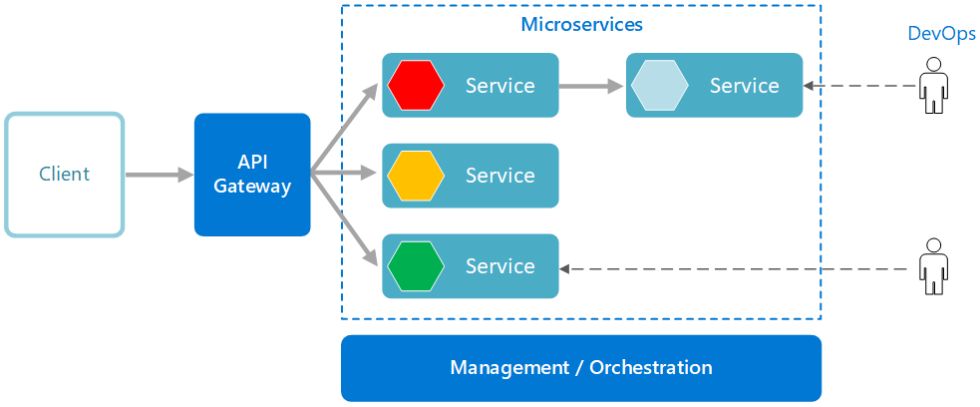
Diversos serviços podem ser criados para objetivos específicos, como no caso do *Amazon Cognito*, que segundo AWS Docs (2022a), pode ser utilizado para controle de acesso dos usuários de uma aplicação. Com esse serviço é possível o gerenciamento de usuário através de criação de contas, *login*, autenticação e entre outros. Ainda no conceito de serviços fornecidos pela *Amazon Web Services* (AWS), é perceptível que a utilização do *Simple Storage Serviçe* (S3) pode ser interessante para criação de aplicações no contexto da AWS, pois o mesmo pode armazenar informações a baixo custo dependendo de sua utilização. Para o presente trabalho esses serviços são utilizados de forma integrada ao *front-end* da aplicação, mais detalhes estão descritos adiante.

2.3.1 Arquitetura

Esse modelo de arquitetura utiliza dos conceitos já apresentados anteriormente sobre microsserviços, sempre limitando o contexto de aplicação do mesmo a responsabilidade única no sistema. Assim como é descrito por Microsoft Azure (2022), cada serviço trabalha

de forma autônoma, mas como são limitados, é comum a interação entre serviços através de chamadas de outros serviços, com isso uma quantidade abundante desses podem ser utilizados num sistema complexo. Já quando os mesmos são utilizados em plataformas que dão maior suporte a microsserviços é comum que estes sejam orquestrados através do gerenciamento de chamadas de uma API exposta para a *web*.

Figura 5 – Estilo de arquitetura de microsserviço



Fonte: Microsoft Azure (2022).

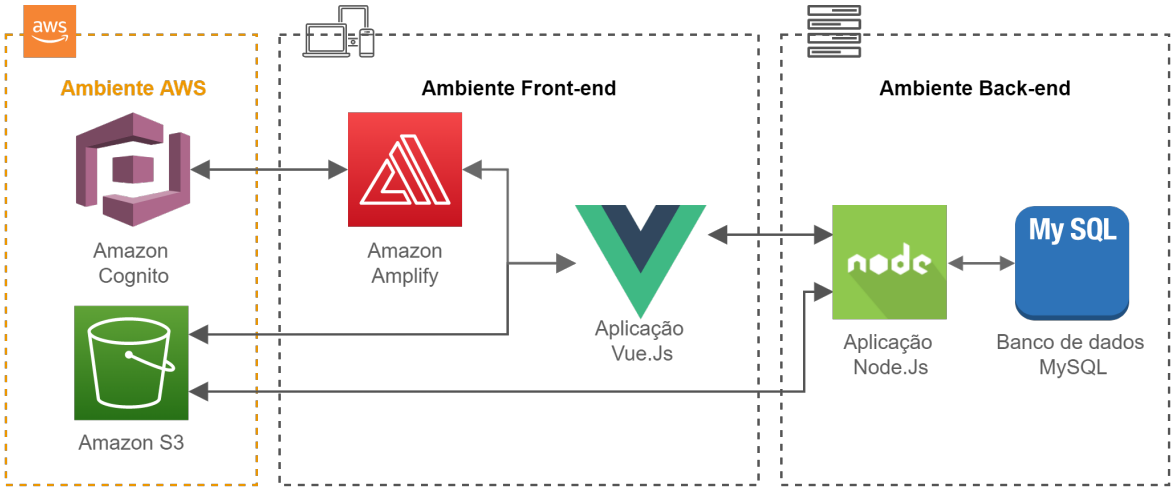
Ainda sobre arquitetura de microsserviços, essa pode ser representada como na Figura 5 de Microsoft Azure (2022), que complementa descrevendo que essa arquitetura possui vários benefícios, entre elas a agilidade desenvolvida para trabalhar com microsserviços, pois esses são independentes e fáceis para dar manutenção, assim como para implementação de novas funcionalidades. Por serem pequenos é possível dedicar uma equipe pequena para realizar implementações e aumentara a produtividade, melhorar o gerenciamento e diminuir a sobrecarga dos integrantes. Tudo isso corrobora para melhora a estabilidade do serviço, pois o mesmo possui pequena base de código que acaba sendo isolada de falhas externas e possibilita que a equipe responsável foque apenas mesma aplicação. Assim como mencionado anteriormente é possível que esses pequenos serviços possam consumir outro, ajudando na combinação de diversas funcionalidade para atingir o objetivo do serviço sem grande preocupação.

2.3.2 Microsservicos AWS

Como a arquitetura de microsserviços possibilita a implementação de *softwares* dedicados a alguma especialidade e estes podem ser consumidos por outros *softwares*, é possível utilizar dos serviços disponibilizados pela AWS para agregar valor ao projeto desenvolvido. Para isso, podem ser escolhidos serviços como *Amazon Cognito* e *Amazon* S3, já citados anteriormente,

para realizar o gerenciamento de usuários através do *framework Amazon Amplify*[1] com *Vue.Js* e o envio e consumo de imagens necessárias para o sistema, respectivamente.

Figura 6 – Arquitetura de microsserviços *AWS* para consumo no *front-end*



Fonte: Elaborado pelo autor.

No ambiente integrado da arquitetura de microsserviços ilustrado na Figura 6, as requisições de serviços por outros serviços se torna comum de forma que é possível que o *back-end* trabalhar como um serviço e utilizar de outros para implementação de novas funcionalidades. No caso do *Amazon* S3 é possível realizar a criação de *Uniform Resource Locator* (URL) pré-assinadas no *back-end* para realização de *upload* por meio do *front-end*. Dessa forma é possível que o *front* realize uma requisição para o serviço da aplicação que gerencia o sistema apenas para conseguir os dados necessários para então realizar o *upload* sem a necessidade de intermédio direto da aplicação *back-end*.

2.4 ISOs

Assim como descrito em ISO.org (2023), existem diversos tipos padrões que podem ser definidos parar ajudar na eficiência, redução de falhas, segurança de dados e trabalho, gasto desnecessários de recursos e diminuir impactos ambientais. Tais padrões são categorizados dependendo do contexto de aplicação, como padrões de gerenciamento de qualidade, segurança em tecnologia da informação ou de saúde. Cada um desses possui ampla categorias de informações úteis e padrões que auxiliam nos desafios abordados da área. A definição dos padrões

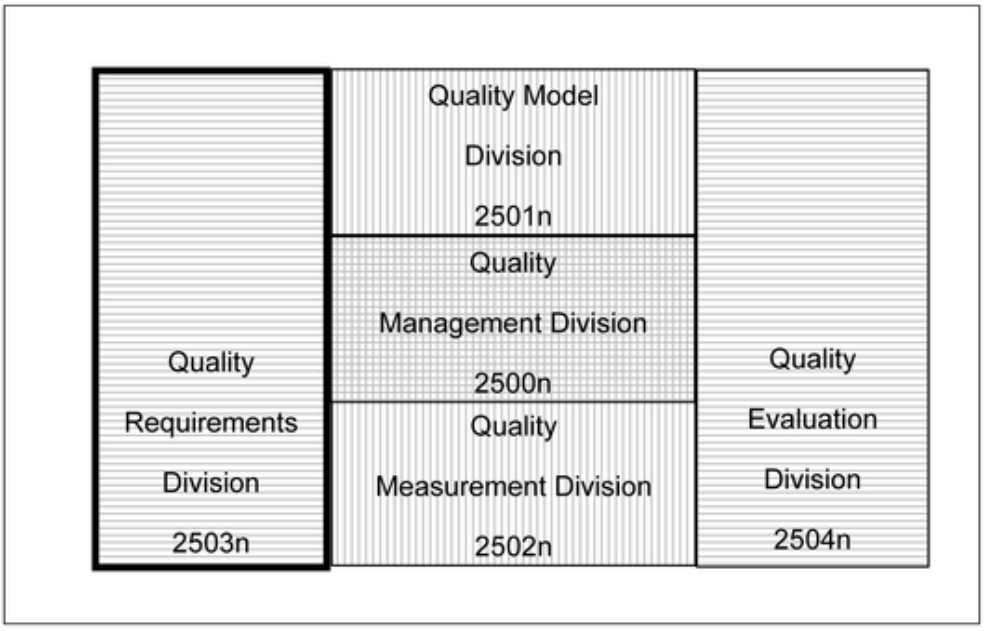
[1] Plataforma de desenvolvimento de aplicativos moveis e *web* que oferece recursos, para hospedagem, autenticação, armazenamento e muito mais. Disponível em https://aws.amazon.com/amplify.

utilizados são descritos por pessoas com experiências em suas respectivas áreas de atuação, conhecendo as necessidades as áreas de atuação, sejam elas fabricantes, vendedores, associações comerciais ou reguladoras. Logo, os documentos elaborados pela *International Organization for Standardisation* (ISO) tem um padrão adotado internacionalmente, elaborado por especialista, e que descreve como uma fórmula, a melhor maneira de fazer algo.

2.4.1 Qualidade de software

Assim como descrito anteriormente, as ISOs podem ser específicas de cada área, logo existem documentos específicos que auxiliam nos trabalhos no campo da tecnologia da informação. Juntos, *International Electrotechnical Commission* (IEC) e ISO definiram padrões relacionados a engenharia de *software* que orientam sobre a medição de qualidade de sistemas, documentando tais orientações nas ISOs da série 250nn. Portanto, pertencendo à série de normas da *Software product Quality Requirements and Evaluation* (SQuaRE), que segundo ISO/IEC JTC 1/SC 7 (2015), combina alguns padrões de uso e orientam a divisão dos conceitos abordados nos documentos da série.

Figura 7 – Divisões SQuaRE



Fonte: ISO/IEC (2007).

Assim como apresentado na Figura 7 é descrito através da ISO/IEC 25022 (2012), a SQuaRE tem seu conteúdo disposto em seis partes, sendo elas: Divisão de Gestão da Qualidade (2500n), Divisão de Modelos de Qualidade (2501n), Divisão de Medição de Qualidade (2502n),

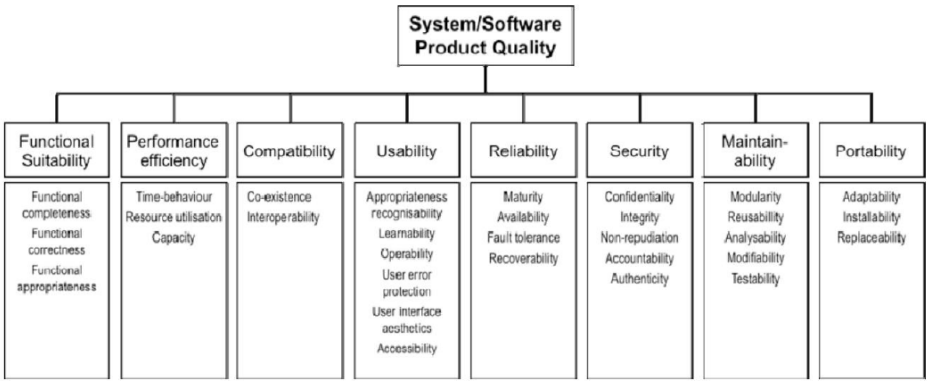
Divisão de Requisitos de Qualidade (2503n), Divisão de Avaliação da Qualidade (2504n) e Divisão de Extensão ISO/IEC 25050 - 25099 SQuaRE.

Conforme o descrito na Tabela 1 da ISO/IEC 25000 (2005), os antigos modelos de métricas de qualidade interna[1] de produtos presentes na ISO/IEC 9126 e ISO/IEC 14598, foram traduzidos para o ISO/IEC 25022 (2012), e para qualidade externa[2], na ISO/IEC 25023 (2011). E assim como descrito nessa última, os padrões documentados na série 2502n contextualizam à mensuração da qualidade de *software* por meio de medidas matemáticas. Tais medidas podem ser utilizadas para avaliação de características de qualidade interna e externa. Nos documentos dessa série estão presentes as Quality Measure Elements (QME)s, que servem como base para as definições das medidas.

2.4.2 Manutenibilidade

Assim como descrito anteriormente, a ISO/IEC 25000 (2005) orienta a utilização da ISO/IEC 25022 (2012) para a descrição de conteúdos sobre atributos de qualidade interna. Dentre estes atributos, existem algumas subdivisões que podem ser utilizadas para auxiliar a mensurar a qualidade interna de um *software*.

Figura 8 – Modelo de qualidade de produto



Fonte: ISO/IEC 25010 (2011).

Como é possível ver na Figura 8, existem 8 divisões, ou sub características, que auxiliam a definir a qualidade interna e externa de um *software*. Entre elas existe a Manutenibilidade, que segundo a ISO/IEC 25010 (2011), descreve o quanto um sistema pode ser mantido realizando modificações necessárias, podendo ser elas correções, melhorias, ou adaptações no

[1] Capacidade de um conjunto de atributos estáticos de um *software* satisfazer as necessidades. ISO/IEC 25000 (2005)

[2] Capacidade do comportamento do código, quando executado, satisfazer as necessidades. ISO/IEC 25000 (2005)

software.

Já a ISO/IEC 25023 (2011), ajuda a mensurar a manutenibilidade um *software* por meio de exemplos de modelos de medida de manutenibilidade. Que segundo a mesma, são medidas capazes de definir um nível de esforço necessário para modificar o *software*. Entres os modelos orientados no documento referido, estão medidas de modularidade, reusabilidade, analisabilidade, modificabilidade e testabilidade.

3 TRABALHOS RELACIONADOS

Aplicações *web* de forma geral acabam sendo utilizadas de forma que abrangem uma quantidade abundante de plataformas, pois possuem a vantagem de serem apresentadas através de um navegador *web*. Mesmo que estas sejam aplicações que possam abranger um público maior, ainda é possível que aplicações específicas para diversas plataformas possam ser desenvolvidas para solucionar problemas. Dentre os diversos sistemas desenvolvidos, existem aqueles que tem a responsabilidade de gerenciamento de recursos dependendo do contexto, assim como o citado o sistema apresentado logo na introdução deste trabalho. Considerando contexto de alocação e reservas para área acadêmica, é interessante a análise de sistemas semelhantes no contexto apresentado, onde estes descrevem formas de solucionar problemas existentes em seus ambientes acadêmicos. As ferramentas propostas por esses trabalhos, independente da plataforma, buscam solucionar problemas relacionados a alocação de salas em seus contextos acadêmicos.

Está seção apresenta os trabalhos relacionados a este, que estão distribuídos nas subseções descritas adiante. A Seção 3.1 apresenta todos os trabalhos relacionados a este. Já para a Seção 3.2, é realizada a comparação entre os trabalhos apresentados, incluindo este.

3.1 Trabalhos

3.1.1 Design and Development of an Integrated Room Reservation System for Higher Education Institutions

O trabalho de German *et al.* (2021), apresenta uma proposta de aplicação para reservas de salas em uma instituição de ensino superior nas Filipinas. O trabalho introduz o tema mencionando a aplicação de tecnologias para a evolução industrial, eletrônica e entre outras. Logo em seguida é descrito a utilização de sistemas de informação para automatizar processo, trazendo benefícios como aumento da eficiência e produtividade a partir dos sistemas desenvolvidos. Também é mencionado que no meio acadêmico é possível ser realizado a criação de sistemas para gerenciamento de trabalho de forma que possa aliviar responsabilidades, simplificar processos e diminuir a utilização de documentos.

Dentre as diversas ações que uma instituição de ensino realiza, uma delas é o gerenciamento de reservas de instalações da mesma. Essa atividade pode ser realizada todo o semestre, principalmente quando isso inclui o ingresso de novos alunos na instituição, por conta

disso o processo se torna mais demorado e inconveniente. Como essas instituições estão cada vez mais dependentes de computadores, a criação de sistemas de informação se tornam ideias para melhorar o desempenho das atividades comuns. Entre essas atividades foi percebido que o gerenciamento de salas pode ser realizada com auxílio de um sistema, ajudando a flexibilizar e aumentar a visibilidade dessas ações de gerenciamento de salas. O sistema desenvolvido a partir desse estudo teve como base a plataforma *web*, criando uma aplicação que facilite o gerenciamento de salas e instalações, melhore o tempo necessário para efetuar uma reserva e permitir que diversos usuários possam acessar o sistema e visualizar as informações.

Os pesquisadores realizaram entrevistas com alunos, professores, funcionários e diversos setores da instituição, diversos resultados coletados foram aplicados na escala de *Likert*[1], e com isso foi possível identificar problemas no processo classificando-os em oito tipos. Ao organizar os dados, foi percebido que 61,7% dos entrevistados concordaram totalmente e 26,2% concordaram que é realizado deslocamento desnecessário para solicitar a reserva de salas. Ainda sobre os dados coletados, foi percebido que 61,3% concordaram totalmente e 19,1% concordaram que existe atraso considerável para a aprovação dos pedidos de reserva.

Para poder visualizar como o sistema deve estar estruturado foi elaborado um diagrama de caso de uso e outro de sequência. Os atores considerados para implementação do sistema foram: estudantes, administradores e facilitadores. Já para realizar a implementação foi utilizado da linguagem *Hypertext Preprocessor* (PHP) é um sistema de código aberto chamado *Booked Schedule* como base. O *e-Reserba Cardinal* foi o nome escolhido para o sistema desenvolvido, que possuiu uma camada de apresentação *web* para acesso às informações, dispondo de interfaces de *login* e um calendários de reservas. A partir desta tela é possível selecionar o horário e realizar a solicitação de reserva após preencher diversas informações, entre elas: horário e título de reserva, descrição e id (identidade) do aluno.

Segundo os autores o sistema ajudou a reduzir problemas relacionados aos processos, como passos desnecessários que não agregam valor à solicitação de reserva. Além disso, a utilização de papel foi reduzida, já que o sistema desenvolvido não necessita de papéis para operar.

[1] Escala de cinco ou quatro pontos inteiros variando de "discordo totalmente"ao "concordo totalmente". Monte (2020)

3.1.2 Classroom Allocation System with User Experience Design

No artigo de Jinjakam *et al.* (2021), é afirmado que a administração de salas pode ocasionar problemas, pois essas são limitadas, complicando o gerenciamento acadêmico. A estratégia ideal para esses casos seria a alocação sem possibilidade de desperdício de espaço ou excesso de alunos na sala. Para tal o sistema desenvolvido propôs minimizar os efeitos de congestionamento e superlotação nas salas de aula em três faculdades da Nigéria.

Ao analisar as salas de aula, os assentos foram categorizados conforme o tipo de assento, equipamento e capacidade, para serem analisados por um algoritmo que utiliza de programação linear e *Inteligência Artificial* (IA) desenvolvidos por um estudante canadense. Segundo o pesquisador devem ser considerados aspectos distintos para realizar a alocação das salas, como o tamanho da sala, recursos necessários, quantidade e categorias de vagas para alunos e equipamentos.

Assim como descrito no artigo anterior, o problema mais frequente da alocação de salas acontece quando o semestre inicia e é necessário adaptar as reservas a quantidade variável de disciplinas, número de alunos matriculados e salas de aula. Nesses momentos ocorrem os problemas descritos logo no início, sendo a superlotação e sub locação dos espaços de salas de aula.

Ainda sobre este artigo, o pesquisador afirma que foram estudadas alocações de sala em três faculdades na Tailândia, considerando que essas tem 12 departamentos operando em Bacharelado, Mestrado e Doutorado. Foi estimado que cerca de 11.310 alunos estão matriculados, e eles estão distribuídos entre 55 cursos de graduação, que pode chegar a ter 1.624 disciplinas distintas, com diversos alunos e quantidade diferente de alunos para cada um.

É descrito que antiga alocação de salas era feita manualmente pelos gestores responsáveis e essa dura em torno de duas semanas. Sobre o processo atual foi realizado uma pesquisa entre os usuários para descobrir as necessidades. Com os resultados obtidos foram selecionados pontos-chave para o desenvolvimento do sistema, entre eles a experiência do usuário, distância entre as salas e departamentos, equipamentos presentes nas salas, fixação de salas de acordo com necessidades, quantidade de alunos esperada para disciplina e exibição de informações importantes das salas. Entre essas informações é descrito a disponibilidade da sala, prédio e zona/departamento.

Assim como na aplicação citada anteriormente, for criado um diagrama de casos de uso para aplicação mostrando as ações que podem ser realizadas. Entre elas estão a criação,

edição e exclusão de salas de aula, relações de *status* e zona com a mesma. Também é descrito a possibilidade de visualizar informações como a quantidade de alunos suportada pela sala e *status* através de uma tabela que lista os horários e salas.

Segundo o pesquisador, com os resultados obtidos os professores ficaram satisfeitos, mas foram encontradas possibilidades de melhorias no sistema. Entre os problemas citados está descrita a necessidade de relacionar departamentos, ferramentas e professores para diminuir a distância necessária para locomoção. Os fatores humanos interferiram muito no desenvolvimento do sistema, pois estes são bem delicados e complicados, por tanto os resultados coletados devem ser utilizados para medir fatores relacionados a possíveis melhorias do sistema.

3.1.3 Intelligent Class Scheduler in Kathmandu University

O trabalho de Shrestha (2018), começa contextualizando os problemas enfrentados por uma universidade do Nepal, onde esta possui diversos departamentos e classes. A mesma tem um processo que apresenta as informações de alocação num quadro de avisos, ou seja, nenhum sistema de informações é utilizado para facilitar essa apresentação. O pesquisador afirma que existem problemas relacionados a comunicação entre administradores e alunos, ocasionando transtornos para os estudantes universitários. Outro problema abordado pelo autor é a falta de conhecimento dos locais da universidade por alunos e funcionários da mesma, implicando na dificuldade de busca de salas, alunos perdendo aula e problemas para realizar exames tanto por alunos quanto por professores. Também é mencionado que existem problemas relacionados a conflito de aulas em determinadas salas.

Para resolver os problemas mencionados no trabalho, o pesquisador propõe enfrentá-los aplicando uma solução provisória ao gerenciar as atividades da universidade com as informações conhecidas. Segundo o mesmo, a instituição carece de compartilhamento de conhecimento e então propôs uma aplicação para funcionar como portal de conhecimento desta instituição. O foco do sistema proposto é a apresentação de dados sobre blocos, departamentos e cronograma das salas de aula da universidade, além disso, a aplicação propõe proporcionar funcionalidade para realização de perguntas relacionadas a localização e horário da administração.

Foram realizadas pesquisas com perguntas sobre a programação das reservas e localização das salas, tendo a participação do total de 25 alunos de cursos de bacharelado e mestrado. Para realizar a coleta das informações foi utilizado o *Google Forms*, distribuídos em diversos grupos universitários no *Facebook*. Dos resultados da pesquisa foi identificado que

a maioria dos entrevistados observa o quadro de informações a cada duas semanas, 80% se sente frustrado ao precisar buscar por salas de aula no começo do semestre, apenas 48% dos entrevistados sabe a localização do seu bloco, cerca de 72% já faltou alguma aula por conta que estava buscando a sala de aula e 84% acham difícil encontrar a sala para seus exames.

A pesquisa realizada serviu como motivação para implementação de uma ferramenta de gerenciamento do conhecimento, essas informações devem ser armazenadas num banco de dados que poderão ser apresentados por *sites* e *tool-kits*[1]. Através do portal de conhecimento proposto, é possível a coleta de informações para serem realizadas tomadas de decisão pela administração, que aprova solicitações de alocação realizadas. O pesquisador prevê que a arquitetura *web* seja suficiente para suportar a implementação do sistema e ainda afirma que a universidade deve incentivar o compartilhamento de conhecimento entre o público da universidade, visando aumentar a quantidade de dados sobre localização e agendamento de salas. Através das informações disponibilizadas no artigo em questão, foi percebido que o sistema proposto não foi de fato implementado.

3.2 Comparativo de trabalhos

Entre os trabalhos citados acima diversas abordagens diferentes foram utilizadas para solução do problema de reserva de salas, os mesmos são comparados através da Tabela 1. O trabalho de Shrestha (2018) descreve a necessidade de criação de um sistema para coleta de dados da plataforma proposta, embora o mesmo tenha mencionado que uma aplicação *web* seria suficiente para consumir os dados e solucionar o problema, este não foi desenvolvido até o momento da postagem do artigo. Já no trabalho de German *et al.* (2021) além de apresentar os problemas, foi realizado o desenvolvimento de uma aplicação *web* em PHP baseada em um *software* de código aberto. Apesar de apresentar os problemas a serem solucionados e desenvolvimento de uma aplicação, o trabalho de Jinjakam *et al.* (2021) não menciona para qual plataforma o mesmo foi desenvolvido, nem detalhes técnicos como a linguagem utilizada no seu desenvolvimento.

Apesar dos trabalhos mencionados proporem aplicações para solucionar os problemas em seu contexto, apenas o trabalho de Jinjakam *et al.* (2021) desenvolve uma aplicação que pode ser utilizada como fonte de dados para demais sistemas que necessitem. Nesse aspecto o referido trabalho se assemelha bastante ao proposto, pois este é desenvolvido com uma API

[1] Conjunto de ferramentas para auxiliar o programador do desenvolvimento. IBM (2023)

Tabela 1 – Comparativo entre os trabalhos relacionados e o proposto

Trabalho	Aplicação *Web*	Plataforma de dados	Descrição das salas	Interface Gráfica	Ambiente Universitário	Foco em Qualidade
German *et al.* (2021)	Sim	Não	Não	Sim	Sim	Não
Shrestha (2018)	Não	Não	Não	Sim	Sim	Não
Jinjakam *et al.* (2021)	Não	Sim	Sim	Não	Sim	
Trabalho proposto	Sim	Sim	Sim	Sim	Sim	Sim

Fonte: Elaborado pelo autor (2022).

para fornecimento dos dados do sistema. Dessa forma as informações necessárias podem ser utilizadas para integração com outros sistemas, caso isso seja necessário posteriormente.

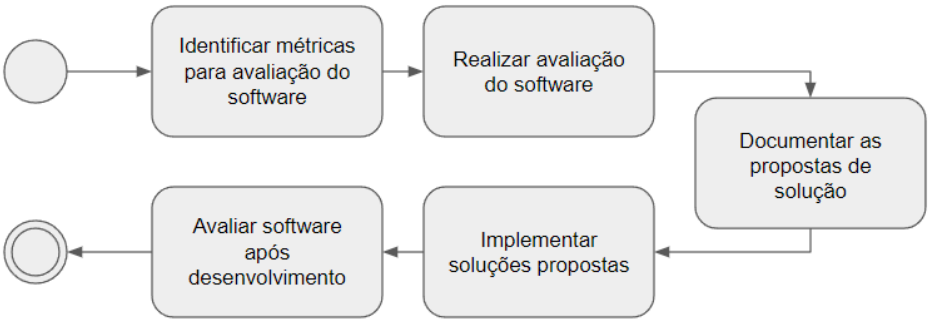
Como os sistemas apresentados são focados na utilização do usuário de forma que facilite a vida do mesmo, é interessante a utilização de interfaces gráficas como meio de acessar as informações necessárias e interagir com o sistema. Ao comparar os trabalhos mencionados foi percebido que apenas o trabalho de Jinjakam *et al.* (2021) não dispõe de uma aplicação em que os usuários principais possam utilizar, dessa forma o mesmo se torna dependente da implementação de outras aplicações para consiga atingir o objetivo de reserva de salas e visualização das informações pertinentes.

É perceptível que todos os trabalhos relacionados a este são destinados a solucionar problemas de reserva de salas no ambiente universitário. Contudo, a apresentação de informações das salas pode ser um diferencial para atribuição de valor aos trabalhos, assim como é descrito pelo trabalho de Jinjakam *et al.* (2021), é importante considerar informações sobre as salas, como os equipamentos presentes e capacidade de alunos da mesma. Apesar desse conceito ser mencionado no trabalho de Shrestha (2018), a aplicação desenvolvida e apresentada no artigo não deixou perceptível como o usuário consegue obter essas informações. Contudo, o trabalho de Jinjakam *et al.* (2021) apresenta apenas uma plataforma para disponibilidade dos dados, mas como o mesmo se compromete a compartilhar informações utilizadas na plataforma desenvolvida, supõe-se que tais dados significantes para reservas deverão estar exposto para qualquer aplicação que venha utilizá-lo.

4 PROCEDIMENTOS METODOLÓGICOS

Nesta seção é apresentada todas as etapas para a execução com foco em atingir os objetivos descritos. Através de cinco subseções, é apresentado todo conteúdo necessário para a execução do trabalho. Logo, essa seção está disposta da seguinte forma: na Seção 4.1 apresenta quais as áreas das métricas identificadas para análise do sistema; já a Seção 4.2 descreve como é realizada a avaliação do *software* antes devidas implementações; com os dados obtidos, na Seção 4.3 é escrita a documentação de propostas de solução para evolução do *software*; sabendo o que deve ser implementado, a Seção 4.4 descreve como é executado desenvolvimento do *software*; ao final é realizada a análise do *software* com as devidas métricas na Seção 4.5.

Figura 9 – Diagrama de Fluxo dos Procedimentos Metodológicos



Fonte: Elaborado pelo autor.

4.1 Identificar métricas para avaliação do *software*

A identificação das métricas considerará o contexto do sistema, *back-end*, e se baseará nas características de qualidade de *software* descritas nas ISOs da série 250nn. Nesse sentido são propostas métricas de qualidade para analisar o sistema, onde estas são representadas pela característica de manutenibilidade. A identificação das métricas também leva em consideração os dados disponibilizados nas ferramentas de análise.

Durante a identificação das métricas foi realizado um estudo com base em autores de referência da área como Fowler (2018) escritor de *Refactoring*, Martin e Coplien (2009) escritores do *Clean Code* e as próprias ISOs.

Após a identificação das métricas, será montado um quadro com o modelo que deve ser seguido para orientar a coleta de dados. Tal quadro consiste em informações sobre as métricas e como elas devem ser aplicadas no sistema.

4.2 Realizar avaliação do *software*

Para executar a avaliação do *software* conforme as métricas, foram selecionados *softwares* para realizar a inspeção do código do sistema, *back-end*. É importante perceber que tais *softwares* estão contidos no contexto das características de qualidade de manutenibilidade. Mesmo que esses *softwares* possam trazer, além de dados, métricas já prontas, essas últimas são desconsideradas, pois, no contexto do presente trabalho as métricas obtidas a partir ações realizadas na Seção 4.1 devem ser utilizadas. Apesar disso, os dados obtidos a partir das análises do *software* são utilizados para aplicação nas métricas selecionadas neste trabalho.

Os *softwares* considerados para realização dessa análise foram os seguintes: *Jest* e *Understand*. Tais *softwares* são utilizados para realização de análise de código estática e, respectivamente, fazem análise de cobertura de testes e linhas de código. Dessa forma foi realizada a análise do código através dos citados *softwares*, onde cada um analisa o código-fonte que apresenta dados necessários para o cálculo das métricas.

Os resultados obtidos são documentados em templates de mensuração disponibilizados nos documentos das ISOs, como é o caso do exemplo apresentado Seção 7 da ISO/IEC 25023 (2011), que define formatos para documentação de medidas de qualidade.

4.3 Documentar as propostas de solução

A documentação descrita nessa seção é referente as soluções encontradas para os problemas e necessidades conhecidas a partir das análises realizadas e resultados obtidos. Assim como foi descrito na Seção 4.2, foram identificadas métricas que podem ser utilizadas para realizar a análise do sistema, *back-end*, e assim possibilitando propor melhorias.

Essa seção trata da documentação proposta pelo autor para solucionar problemas e implementar melhorias necessárias para o sistema. Dessa forma, essa seção é um guia para etapa de desenvolvimento, descrita na Seção 4.4, e descreve com mais detalhes quais as restrições e estratégias utilizadas para evolução do sistema na totalidade. É possível perceber que ao descrever tais estratégias é possível definir ao nível arquitetural como devem ser realizadas as modificações, e quais padrões de código e arquitetura podem ser utilizados.

4.4 Implementar soluções propostas

Assim como descrito anteriormente, a documentação é utilizada como um guia para implementação do sistema, *back-end*. Com base nas necessidades descritas, é realizada a implementação conforme o resultado da análise coletada. Levando em consideração o prazo para realização deste trabalho, o sistema teve o código-fonte aprimorado de acordo com as melhoras documentadas. Dado as melhorias descritas, é realizado o desenvolvimento de cada uma delas respeitando suas restrições e focando no objetivo.

Para o desenvolvimento, é utilizado de *Javascript* para todo o sistema, *NodeJs* e parte de *Typescript* no *back-end* e *VueJs*, *Amazon Cognito* juntamente ao *Amazon Amplify* no *front-end*. Como ambas são aplicações distintas, mas pertencentes ao mesmo contexto, é utilizado do padrão de *API REST* para definição da interface do *back-end* e padrões de requisição impostas ao *front-end*.

4.5 Avaliar *software* após desenvolvimento

Essa etapa de avaliação é bem semelhante à descrita na Seção 4.2, tendo como diferença o objeto de análise das ferramentas, pois o mesmo é o sistema, *back-end*, analisado anteriormente, mas com melhorias orientadas aos dados coletados e documentados na Seção 4.3 antes das melhorias implementadas no sistema.

Após realização da análise dos sistemas na nova versão, é realizado um comparativo com o mesmo, antes e depois das melhorias implementadas. Com os dados gerados por essa análise é possível verificar a qualidade de implementação das técnicas escolhidas para solucionar os problemas e pontos de evolução identificados.

5 RESULTADOS

Neste capítulo será apresentado os resultados obtidos com as ações realizadas no projeto *web* referido neste trabalho. São descritas etapas de identificação de métricas, análise do *software* com ferramentas automatizadas, documentação de melhorias e análise após melhorias aplicadas no software. Os dados coletados nas análises são interpretados para produzir informações sobre a evolução do software conforme as métricas elaboradas.

5.1 Identificar métricas para avaliação do *software*

Assim como descrito na (ISO/IEC 25000, 2005), o desenvolvimento de softwares de qualidade é de primordial importância. Diversos autores escreveram livros sobre qualidade de software, e principalmente, sobre qualidade de código, como no caso de (MARTIN; COPLIEN, 2009), escritor de *Clean Code*. Ainda no mesmo livro, (MARTIN; COPLIEN, 2009) descreve que é ideal diminuir a qualidade de linhas para cada função, arquivo e classe, e até sugere 25 linhas como quantidade limite de linhas para cada função.

A avaliação do sistema utilizará do modelo do Quadro 1 para orientar a coleta de dados e ajudar na interpretação dos resultados coletados na análise sobre modularidade. Assim como é descrito neste quadro, será analisado a quantidade de linhas para funções, arquivos e classes do sistema, e como resultado da análise, será obtido um índice para descrever o quão bem o sistema está em cada situação de cada métrica analisada.

A quantidade máxima determinada para cada uma das medições levou em conta valores apresentados como exemplo por (MARTIN; COPLIEN, 2009) em *Clean Code*. Assim como já foi citado anteriormente, o referido autor sugere 25 linhas para tamanho de funções, outros valores não são explicitamente sugeridos para linhas em arquivos e classes, mas é sugerido diminuir a quantidade de linhas para eles. Por tanto, foi definido tamanho de 120 linhas como limite para cada arquivo e 100 linhas para cada classe.

Já o segundo Quadro 2 apresenta métricas utilizando um modelo parecido com o apresentado no Quadro 1, mas dessa vez com dados que orientam a análise dos testes no sistema. Para mensurar o quão bem o sistema está em relação aos testes, é realizado um cálculo sobre a quantidade de linhas, métodos e declarações testadas em cada classe. As funções de medição são elaboradas com baseada nos resultados obtidos pela ferramenta *Jest*, que entrega dados sobre linhas, métodos e declarações testadas.

Quadro 1 – Medidas de tamanho de funções, classes e arquivos.

Nome	Descrição	Função de medição	Método
Percentual de funções com tamanho acima do permitido.	Quantas funções passam do limite de linhas estipulado?	X = (NF / NT) * 100 NF = Número de funções que passam do limite de linhas por função. NT = Número total de funções.	Análise de código usando a ferramenta Understand SciTools.
Percentual de classes com tamanho acima do permitido.	Quantas classes passam do limite de linhas estipulado?	X = (NC / NT) * 100 NC = Número de classes que passam do limite de linhas por classes. NT = Número total de classes.	Análise de código usando a ferramenta Understand SciTools.
Percentual de arquivos com tamanho acima do permitido.	Quantos arquivos passam do limite de linhas estipulado?	X = (NA / NT) * 100 NA = Número de arquivos que passam do limite de linhas por arquivo. NT = Número total de arquivos.	Análise de código usando a ferramenta Understand SciTools.

Fonte: Elaborado pelo autor.

Quadro 2 – Medidas de cobertura de testes por linhas, métodos e declarações.

Nome	Descrição	Função de medição	Método
Percentual de linhas de código testadas.	Qual o percentual de linhas de código que foram chamadas pelo menos uma vez durante os testes unitários?	X = (LT/T) * 100 LT = Número de linhas testadas. T = Número total de linhas identificadas no teste.	Análise de código usando a ferramenta Jest.
Percentual de métodos testados.	Qual o percentual de métodos que foram chamadas pelo menos uma vez em uma classe durante os testes unitários?	X = (MT/T) * 100 MT = Número de métodos testados. T = Número total de métodos identificados no teste.	Análise de código usando a ferramenta Jest.
Percentual de declarações testadas.	Qual o percentual de declarações “If” que foram cumpridas pelo menos uma vez em uma classe durante os testes unitários?	X = (DT/T) * 100 DT = Número de declarações testadas. T = Número total de declarações identificadas no teste.	Análise de código usando a ferramenta Jest.

Fonte: Elaborado pelo autor.

5.2 Realizar avaliação do *software*

Essa seção é dedicada para descrever os resultados da avaliação do software em relação às métricas mencionadas na seção anterior. Por tanto, serão apresentadas resultados relacionados a modularidade do sistema, quantidade de linha de funções, classes e arquivos, e logo em seguida, será apresentado os resultados dos testes por linhas, métodos e declarações testadas.

A primeira análise sobre a modularidade do sistema envolve o cálculo do percentual de linhas por função acima do total permitido, que teve um resultado de 11.11%, e foi obtido por meio do cálculo descrito no quadro 3, onde é possível perceber que 2 das 18 funções totais dos arquivos passam do limite definido. Logo em seguida é apresentado o cálculo do máximo de classes acima do total definido, onde foi obtido o resultado de 0%, por tanto nenhuma das 11 classes passou do limite definido. O mesmo resultado ocorre para o resultado de arquivos acima do limite definido, onde nenhum dos 19 arquivos analisados passou do limite definido.

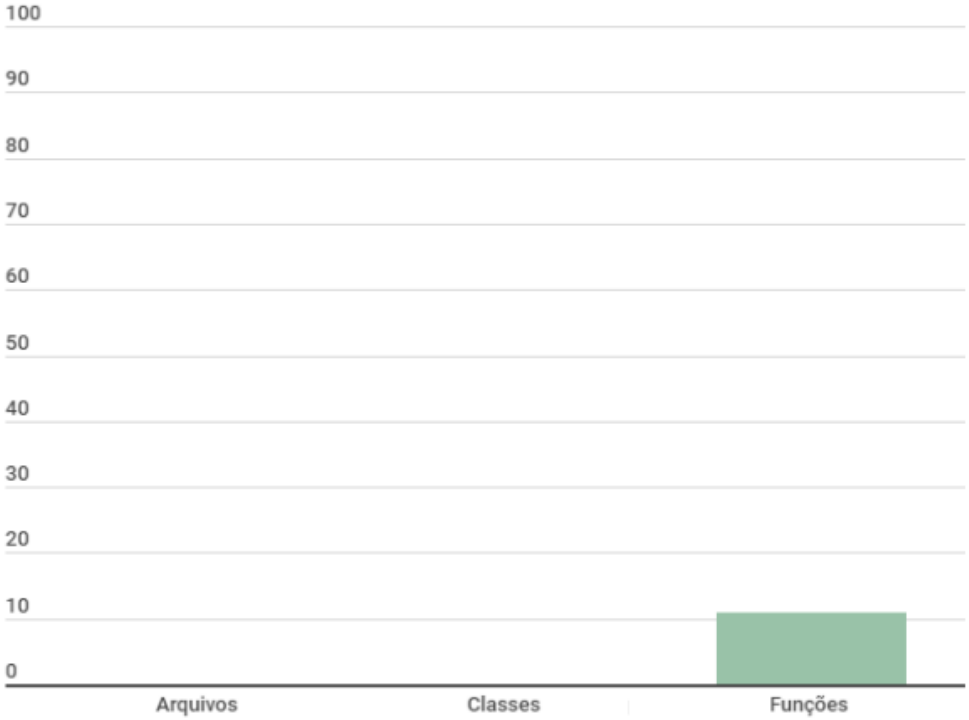
Quadro 3 – Avaliação do código-fonte antes de modificações.

Medida coletada	**Completude da coleta**
Taxa de funções com tamanho acima do permitido.	Número de funções que passam do limite de linhas por função (NF) = 2 funções. Número total de funções (NT) = 18 funções. X = (2 / 18) * 100 = 11.11% de funções acima do permitido.
Taxa de classes com tamanho acima do permitido.	Número de classes que passam do limite de linhas por classes (NC) = 0 classes. Número total de classes (NT) = 11 classes. X = (0 / 11) * 100 = 0% de classes acima do permitido.
Taxa de arquivos com tamanho acima do permitido.	Número de arquivos que passam do limite de linhas por arquivo (NA) = 0 arquivos. Número total de arquivos (NT) = 19 arquivos. X = (0 / 19) * 100 = 0% de arquivos acima do permitido.

Fonte: Elaborado pelo autor.

Na Figura 10 é apresentado os resultados de forma visual, onde, numa escala de 0 a 100% dos objetos analisados, é possível perceber que houveram somente resultados que afetassem a modularidade do sistema em relação à quantidade de linhas por funções.

Figura 10 – Gráfico de percentual de linhas do código-fonte por funções, classes e arquivos acima do valor determinado.



Fonte: Elaborado pelo autor.

Seguindo a ordem de apresentação no Quadro 4, e a Figura 11, para a primeira análise da cobertura de testes do sistema, 33 das 59 linhas de código as possuíam testes automatizados representado 55.93% de cobertura de testes por linhas de código.

É interessante perceber que a baixa quantidade de linhas de código identificada se da

Figura 11 – Percentual de linhas cobertas por testes antes das modificações.

Fonte: Elaborado pelo autor.

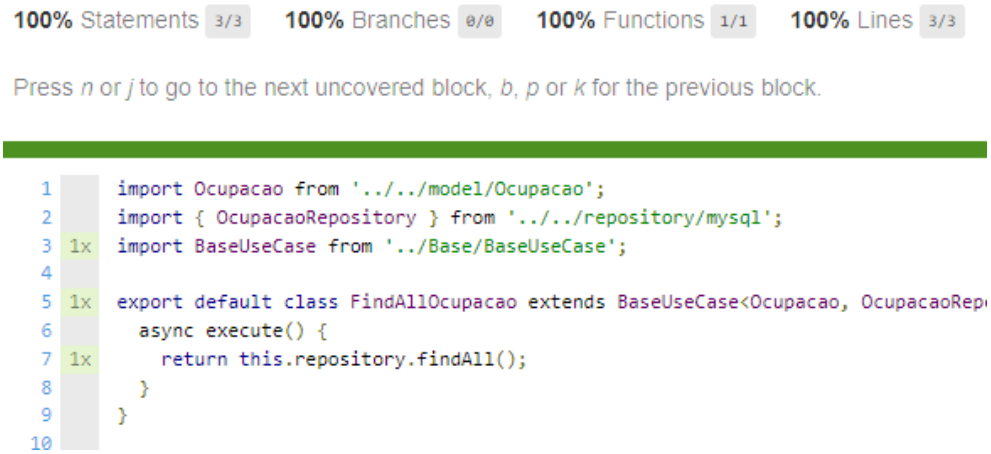
Quadro 4 – Avaliação dos testes antes de modificações.

Medida coletada	**Completude da coleta**
Percentual de linhas de código testadas.	Número de linhas testadas (LT) = 33 linhas. Número total de linhas identificadas no teste (T) = 59 linhas. X = (33/59) * 100 = 55.93% de linhas cobertas por testes.
Percentual de métodos testados.	Número de métodos testados (MT) = 1 método. Número total de métodos identificados no teste (T) = 5 métodos. X = (1/5) * 100 = 20% de métodos cobertos por testes.
Percentual de declarações testadas.	Número de declarações testadas (DT) = 37 declarações. Número total de declarações identificadas no teste (T) = 63 declarações. X = (63/37) * 100 = 58.73% de declarações cobertas por testes.

Fonte: Elaborado pelo autor.

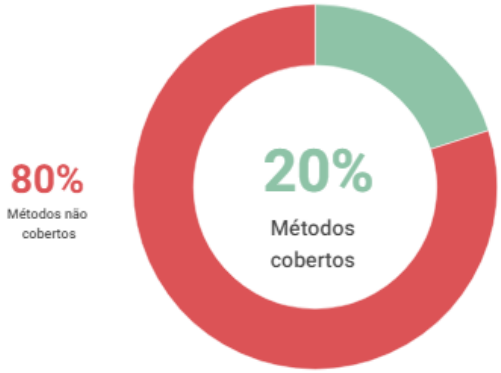
por conta da forma como a ferramenta *Jest* analisa o código. A mesma adiciona a somatória de linhas de código testadas apenas algumas linhas que o mesmo considera testável para geração de resultados. Por tanto, comentários, linhas em branco, *tokens* para fechamento de escopo como chaves e entre outros não são considerados na contagem de linhas testadas. Assim como é possível perceber na Figura 12 um arquivo com tamanho de 10 linhas tem apenas 3 linhas contabilizadas no teste, pois apenas essas são testáveis pela ferramenta.

Figura 12 – Resultado de cobertura de teste da ferramenta *Jest* em um arquivo.



Fonte: Elaborado pelo autor.

Figura 13 – Percentual de métodos cobertos por testes antes das modificações.



Fonte: Elaborado pelo autor.

Para a segunda análise de cobertura de testes foi verificado a quantidade de métodos testados. Como é possível ver na Figura 13, 20% dos métodos foram testados, sendo que apenas 1 dos 5 métodos identificados pela ferramenta foi coberto por testes. Assim como descrito na análise anterior, a cobertura de testes de métodos da ferramenta levou em conta os métodos analisados nos arquivos que possuem ao menos um teste configurado, por conta disso apenas 5 métodos foram identificados como testáveis.

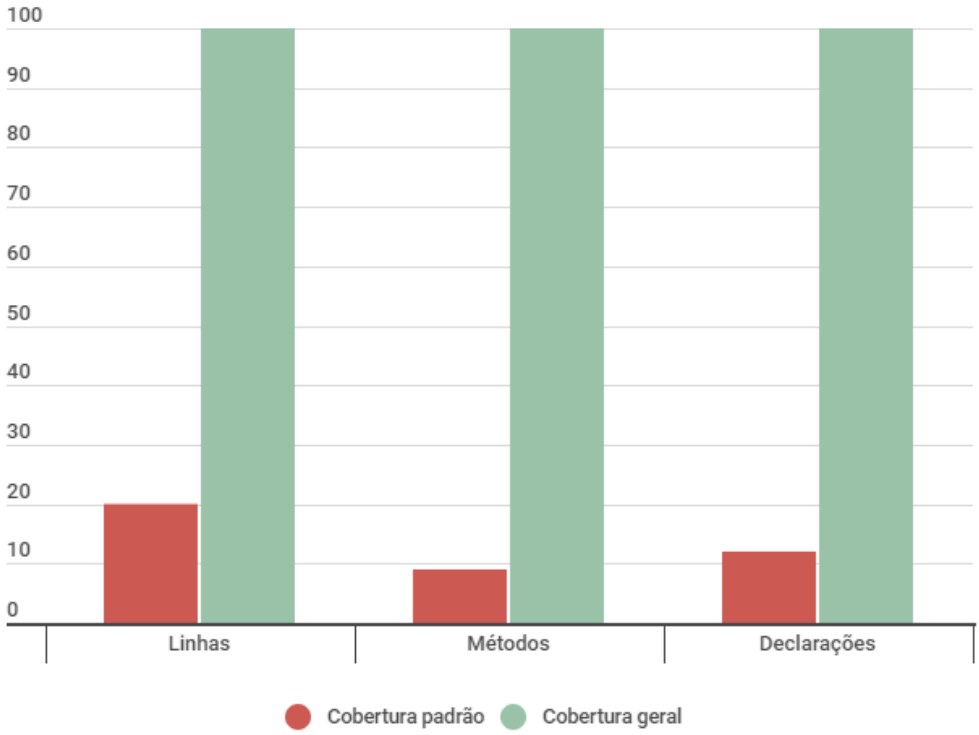
Figura 14 – Percentual de declarações cobertas por testes antes de modificações.

Fonte: Elaborado pelo autor.

Já para a terceira análise de testes, foi identificado a qualidade de declarações testadas, onde é possível perceber através da Figura 14 que foi obtido o resultado de 58.73% de cobertura por testes, sendo 37 declarações cobertas por testes, de um total de 63. Esse teste também é afetado pela configuração da ferramenta para identificação somente das declarações em arquivos explicitamente testados.

Para uma demonstração mais clara do quanto o sistema possui de cobertura de teste, foi realizada a análise da quantidade de linhas, métodos e declarações testáveis em todo o código, incluindo arquivos que não são explicitamente testados. Como é possível ver na Figura 15 a

Figura 15 – Resultado do total quantidade de linhas, funções e declarações testáveis variando por configuração de cobertura de testes.



Fonte: Elaborado pelo autor.

quantidade testável de linhas, métodos e declarações na ferramenta respectivamente, 282, 54, 289, são valores bem acima do total obtido na configuração de cobertura padrão de testes. Dessa forma é possível perceber que, o total de linhas, métodos e declarações dos testes acima não estavam contando com o total existente no projeto.

5.3 Documentação de propostas de melhoria

Com base nos resultados obtidos na primeira análise, essa seção dispões da documentação de estratégias para melhora dos resultados a fim de agregar valor ao sistema em analizado.

Os resultados sobre os testes do sistema apresentaram que a maioria do sistema não possui cobertura de testes. Pois, do total de 289 declarações do sistema, apenas 63 foram identificadas nos testes e destas, 37 estavam cobertas por testes. O mesmo ocorre com linhas e métodos testados, com um total de 282 e 54, respectivamente, a ferramenta identificou 59 linhas e 5 métodos no total dos arquivos testados. Por tanto se faz necessário aumentar a cobertura de testes do sistema, com a configuração de cobertura geral, para ser possível garantir que todas as funcionalidades do mesmo estejam funcionando corretamente, independente das alterações.

Como o restante da aplicação não coberta por testes também está escrita em *Javascript*, continuará sendo realizado o desenvolvimento dos testes usando a ferramenta *Jest*. Os testes seguirão com objetivo de aumentar a cobertura de testes o máximo possível, ainda seguindo na implementação de testes unitários.

Os resultados obtidos sobre tamanho de funções, arquivos e classes apresentaram somente duas funções que passaram do limite definido, sendo essas funções construtoras de objetos das classes. Por tanto, não se torna necessário a refatoração das mesma, tendo em vista que estas cumpre sua responsabilidade mantendo a alta coesão e baixo acoplamento. Como não foi identificado nenhum arquivo ou classe com tamanho acima do determinado, não será realizada nenhuma ação em relação quanto ao tamanho dessas.

O resultado de baixa quantidade de funções e nenhum arquivo classes acima do limite pode ser compreendido por conta do tamanho do sistema e da forma como o mesmo foi projetado. O sistema de alocação e reserva de salas foi implementado levando em consideração a arquitetura de estrutura de código baseada nos conceitos de *Clean Architecture* de (MARTIN; COPLIEN, 2009). Seguindo esses conceitos foi realizada a separação a de camadas de aplicação, domínio e interface, que ajudaram na criação de um código mais coeso e desacoplado, e que não passam do tamanho máximo definido anteriormente.

5.4 Avaliação do *software* após desenvolvimento

Essa seção é dedicada para descrever os resultados da avaliação do software com base nas métricas selecionadas e nas decisões de propostas de melhoria da seção anterior. Assim como foi descrito na seção anterior, dados sobre modularidade do sistema não serão analisados por conta do baixo percentual de modificações necessárias no código identificado. Logo, será realizada novamente a análise dos testes do sistema em relação ao percentual de linhas, métodos e declarações testadas.

É importante perceber que a quantidade de linhas, métodos e declarações analisadas conta com a configuração de cobertura de testes de todos os arquivos, incluindo arquivos testados e não testados. Dessa forma, os valores de total para cada contexto diferem dos valores da primeira análise, onde apenas dados de arquivos testados eram totalizados.

O Quadro 5 acima contém dados sobre a análise de cobertura de testes após a implementação dos testes, onde foram considerados os três contextos de cobertura de testes por linhas, métodos e declarações. Logo a baixo é descrito o percentual de cada contexto, assim

Quadro 5 – Avaliação dos testes após modificações.

Medida coletada	**Completude da coleta**
Percentual de linhas de código testadas.	Número de linhas testadas (LT) = 281 linhas. Número total de linhas identificadas no teste (T) = 282 linhas. X = (281/282) * 100 = 99.64% de linhas cobertas por testes.
Percentual de métodos testados.	Número de métodos testados (MT) = 53 método. Número total de métodos identificados no teste (T) = 54 métodos. X = (53/54) * 100 = 98.14% de métodos cobertos por testes.
Percentual de declarações testadas.	Número de declarações testadas (DT) = 288 declarações. Número total de declarações identificadas no teste (T) = 289 declarações. X = (288/289) * 100 = 99.65% de declarações cobertas por testes.

Fonte: Elaborado pelo autor.

como sua respectiva figura para apresentação visual da evolução.

Figura 16 – Percentual de linhas cobertas por testes após modificações.

Fonte: Elaborado pelo autor.

Seguindo a ordem de apresentação no Quadro 5, a Figura 16 contém dados da primeira análise de cobertura de testes do sistema, onde 281 das 282 linhas de código possuíam testes automatizados representando 99.64% de cobertura de testes por linhas de código.

Figura 17 – Percentual de métodos cobertos por testes após modificações.

Fonte: Elaborado pelo autor.

Para a segunda análise de cobertura de testes foi verificado a quantidade de métodos testados. Como é possível ver na Figura 17, 98.14% dos métodos foram testados, sendo que 53 dos 54 métodos identificados pela ferramenta foram cobertos por testes.

Figura 18 – Percentual de declarações cobertas por testes após modificações.

Fonte: Elaborado pelo autor.

Figura 19 – Código de modelo de código não coberto por testes.

```
26
27        @ManyToOne(() => Sala)
28        @JoinColumn({ name: "sala_id"})
29   4x   sala: Sala;
30
```

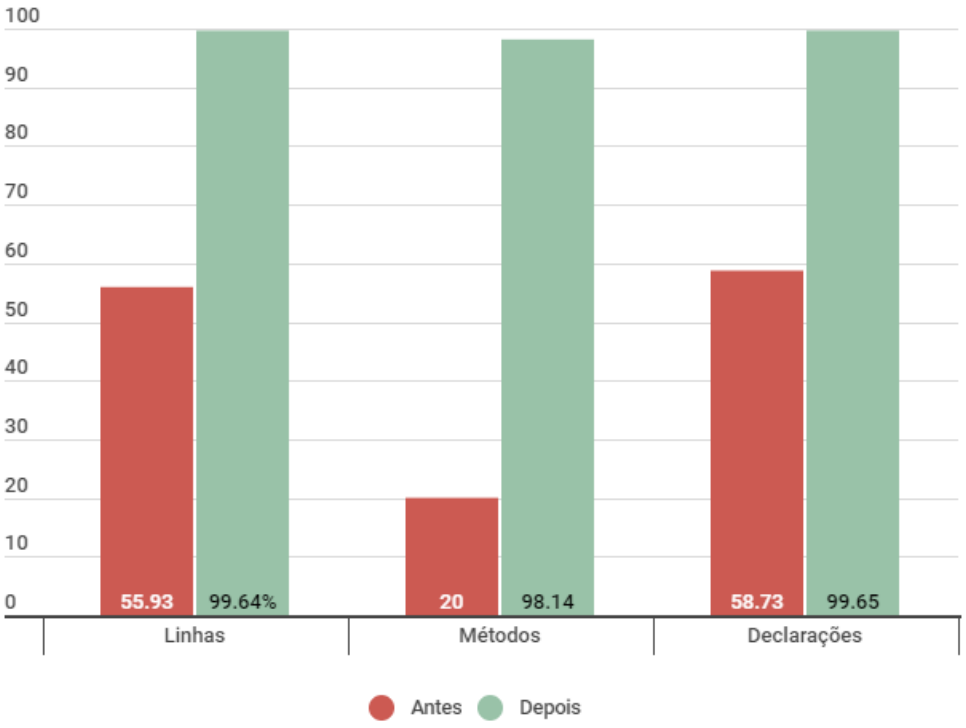
Fonte: Elaborado pelo autor.

Já para a terceira análise de testes, assim como é possível ver na Figura 18, foi identificado a qualidade de declarações testadas, onde esse resultado ficou descrito como 99.65% de cobertura de testes, sendo 288 declarações cobertas por testes de um total de 289.

Os trechos de código não cobertos por testes pelo sistema necessitam de teste de integração para conferir a execução dos mesmos. Assim como é possível ver na Figura 19, a função da linha 27 não foi coberta por testes, mas a mesma é caracterizada como uma função executada quando a biblioteca interna do sistema executa chamadas no banco de dados. Por tanto, os fluxos possíveis para testes unitários do sistema foram realizados conforme as limitações apresentadas.

Com a análise dos testes após o desenvolvimento é possível perceber que a quantidade de linhas, métodos e declarações foram, em sua maioria, testadas. Assim como é possível ver na Figura 20, o aumento da cobertura de testes chegou próximo a 100% em todos os contextos analisados. Por tanto, foi obtido uma cobertura de testes mais completa, evitando possíveis problemas por implementações incorretas.

Figura 20 – Gráfico comparativo de cobertura de testes antes e depois das alterações.



Fonte: Elaborado pelo autor.

6 CONCLUSÕES E TRABALHOS FUTUROS

O desenvolvimento de um *software* pode, muitas vezes, passar por um processo que exija início rápido para atender as demandas de negócio. Durante esse processo alguns *softwares* acabam passando por problemas em sua evolução, sejam esses, aumento na complexidade de um sistema mal modularizado ou implementação de *bug* ao tentar trabalhar na evolução de um sistema sem testes.

Os conceitos abordados durante o presente trabalho fazem parte e ajudam na evolução do *software*, garantindo a consistência do que está sendo trabalhado, de forma que o usuário final possa receber o produto desenvolvido que consiga seguir em constantes melhorias sem grandes problemas.

Com esses conceitos bem aplicados no sistema é possível que futuros desenvolvimentos possam ser realizados de forma menos problemática e sem enfrentar grandes problemas. Tais evoluções do sistema abordado neste trabalho são referentes a integração do sistema em nuvem para disponibilizar a aplicação para o usuário. Dessa forma será possível coletar dados sobre possíveis melhorias e correções.

Com objetivo de continuar garantindo a consistência do sistema, também é possível que o presente sistema tenha o desenvolvimento de testes de integração e ponta a ponta, a fim de garantir que todo sistema está sendo desenvolvido corretamente com todas os componentes internos e externos.

REFERÊNCIAS

AIRBNB. **O que é o Airbnb e como ele funciona? - Central de Ajuda do Airbnb**. 2022. Disponível em: https://www.airbnb.com.br/help/article/2503/o-que-%C3%A9-o-airbnb-e-como-ele-funciona. Acesso em: 15 mai. 2022.

AWS Docs. **Amazon Cognito – Cadastramento e login de usuários com simplicidade e segurança | Amazon Web Services (AWS)**. 2022. Disponível em: https://aws.amazon.com/pt/cognito/. Acesso em: 05 jun. 2022.

AWS Docs. **O que são microsserviços? | AWS**. 2022. Disponível em: https://aws.amazon.com/pt/microservices/. Acesso em: 05 jun. 2022.

AWS, G. **O que é uma API? – Guia de APIs para iniciantes – AWS**. 2022. Disponível em: https://aws.amazon.com/pt/what-is/api/. Acesso em: 05 jun. 2022.

BERNERS-LEE, T.; CONNOLLY, D. W. **Hypertext Markup Language - 2.0**. [*S. l.*], 1995. Disponível em: https://datatracker.ietf.org/doc/rfc1866/. Acesso em: 04 jun. 2022.

FENG, X.; SHEN, J.; FAN, Y. Rest: An alternative to rpc for web services architecture. In: **2009 First International Conference on Future Information Networks**. [*S. l.*: *s. n.*], 2009. p. 7–10.

FIELDING, R. T.; RESCHKE, J. **Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content**. [*S. l.*], 2014. Disponível em: https://datatracker.ietf.org/doc/rfc7231/. Acesso em: 05 jun. 2022.

FLANAGAN, H. **Cascading Style Sheets (CSS) Requirements for RFCs**. [*S. l.*], 2016. Disponível em: https://datatracker.ietf.org/doc/rfc7993/. Acesso em: 04 jun. 2022.

FOWLER, M. **Refactoring**: Improving the design of existing code. 2. ed. Boston, MA: Addison-Wesley, 2018. (Addison-Wesley Signature Series (Fowler)). ISBN 978-0-13-475759-9.

GELLERSEN, H.-W.; GAEDKE, M. Object-oriented Web application development. **IEEE Internet Computing**, v. 3, n. 1, p. 60–68, fev. 1999. ISSN 10897801. Disponível em: http://ieeexplore.ieee.org/document/747323/.

GERMAN, J. D.; YAP, D. C. G.; BINOYA, G. O. Design and Development of an Integrated Room Reservation System for Higher Education Institutions. In: **2021 IEEE 8th International Conference on Industrial Engineering and Applications (ICIEA)**. Chengdu, China: IEEE, 2021. p. 231–236. ISBN 9781665428958. Disponível em: https://ieeexplore.ieee.org/document/9436766/.

IBM. **Programmer's Toolkit**. 2023. Disponível em: https://www.ibm.com/docs/en/i/7.4?topic=overview-programmers-toolkit.

ISO/IEC. **ISO/IEC 25030:2007(en) Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Quality requirements**. 2007. Disponível em: https://www.iso.org/obp/ui/fr/#iso:std:iso-iec:25030:ed-1:v1:en.

ISO/IEC 25000. **ISO/IEC 25000: Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaREe**. Japan, 2005.

ISO/IEC 25010. **ISO/IEC 25010: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models**. [*S. l.*], 2011.

ISO/IEC 25022. **ISO/IEC 25022: Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of quality in use**. Japan, 2012.

ISO/IEC 25023. **ISO/IEC 25023: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality**. Japan, 2011.

ISO/IEC JTC 1/SC 7. **ISO 25000 SQuaRE series**. 2015. Disponível em: https://committee.iso.org/sites/jtc1sc7/home/projects/flagship-standards/iso-25000-square-series.html. Acesso em: 2023-04-27.

ISO.org. **ISO standards are internationally agreed by experts**. 2023. Disponível em: https://www.iso.org/standards.html. Acesso em: 26 abr. 2023.

JINJAKAM, C.; PHOTHONG, P.; PEARODWONG, P.; HONGSUWAN, T. Classroom Allocation System with User Experience Design. In: **2021 7th International Conference on Engineering, Applied Sciences and Technology (ICEAST)**. Pattaya, Thailand: IEEE, 2021. p. 190–193. ISBN 9781665441223. Disponível em: https://ieeexplore.ieee.org/document/9426279/.

MARTIN, R. C.; COPLIEN, J. O. **Clean code: a handbook of agile software craftsmanship**. Upper Saddle River, NJ [etc.]: Prentice Hall, 2009. ISBN 9780132350884 0132350882. Disponível em: https://www.amazon.de/gp/product/0132350882/ref=oh_details_o00_s00_i00.

MDN. **REST - Glossário | MDN**. 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Glossary/REST. Acesso em: 05 jun. 2022.

MDN. **JavaScript | MDN**. 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 05 jun. 2022.

Microsoft Azure. **Estilo de arquitetura de microsserviço - Azure Architecture Center**. 2022. Disponível em: https://docs.microsoft.com/pt-br/azure/architecture/guide/architecture-styles/microservices. Acesso em: 05 jun. 2022.

MONTE, L. G. **Escala Likert difusa**: um estudo sobre diferentes abordagens. 2020. 41 f. – Trabalho de Conclusão de Curso (Graduação em Matemática Industrial) – Universidade Federal do Ceará, Centro de Ciências, Fortaleza, 2020. Disponível em: http://www.repositoriobib.ufc.br/0000b4/0000b4c1.pdf. Acesso em: 05 jun. 2022.

Nodejs.Org. **About Node.js**. 2022. Disponível em: https://nodejs.org/en/about/. Acesso em: 05 jun. 2022.

OKTRIWINA, A. S. **Single Page Application vs Multi Page Application**. 2020. Disponível em: https://glints.com/id/lowongan/single-page-application-vs-multi-page-application/. Acesso em: 05 jun. 2022.

REDHAT. **O que é API REST?** 2020. Disponível em: https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api. Acesso em: 05 jun. 2022.

SANCHEZ, L. M.; DIAZ-OREIRO, I.; QUESADA, L.; GUERRERO, L. A.; LOPEZ, G. Smart Meeting Room Management System Based on Real-Time Occupancy. In: **2019 IV Jornadas Costarricenses de Investigación en Computación e Informática (JoCICI)**. San Pedro, Costa Rica: IEEE, 2019. p. 1–6. ISBN 9781728147871. Disponível em: https://ieeexplore.ieee.org/document/9105174/.

SANTOS, L. **Node.js Under The Hood #3 - Deep Dive Into the Event Loop**. 2019. Disponível em: https://dev.to/_staticvoid/node-js-under-the-hood-3-deep-dive-into-the-event-loop-135d. Acesso em: 05 jun. 2022.

SHRESTHA, G. S. R. Intelligent Class Scheduler in Kathmandu University. **Journal of Management Research**, v. 10, n. 4, p. 17, set. 2018. ISSN 1941-899X. Disponível em: http://www.macrothink.org/journal/index.php/jmr/article/view/13060.

SINGH, H.; SHAH, R. R. BOOKiiIT - Designing a Venue Booking System (Technical Demo). In: **2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM)**. New Delhi, India: IEEE, 2020. p. 287–291. ISBN 9781728193250. Disponível em: https://ieeexplore.ieee.org/document/9232604/.

VUEJS.ORG. **Introdução — Vue.js**. 2022. Disponível em: https://br.vuejs.org/v2/guide/index.html. Acesso em: 05 jun. 2022.

VUEJS.ORG. **Reactivity in Depth — Vue.js**. 2022. Disponível em: https://v2.vuejs.org/v2/guide/reactivity.html#How-Changes-Are-Tracked. Acesso em: 05 jun. 2022.

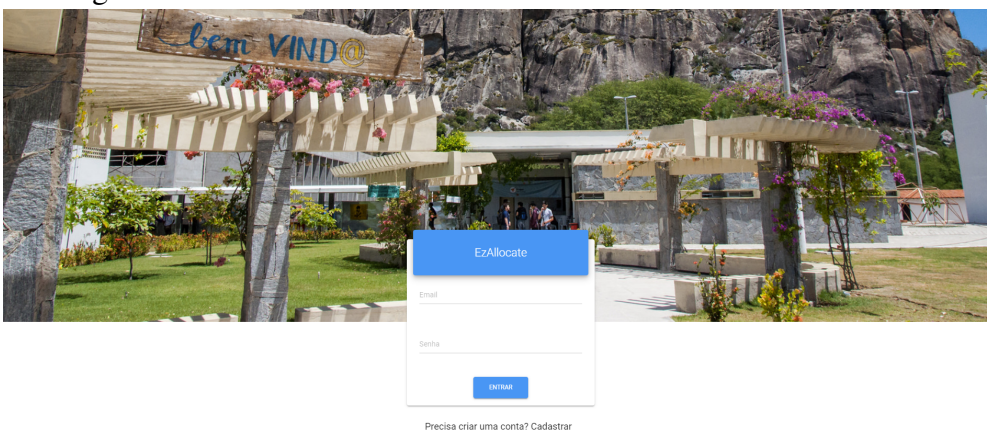
VUEJS.ORG. **Ways of Using Vue | Vue.js**. 2022. Disponível em: https://vuejs.org/guide/extras/ways-of-using-vue.html#single-page-application-spa. Acesso em: 05 jun. 2022.

WONG, S.; CAI, Y.; KIM, M.; DALTON, M. Detecting software modularity violations. In: **Proceedings of the 33rd International Conference on Software Engineering**. Waikiki, Honolulu HI USA: ACM, 2011. p. 411–420. ISBN 9781450304450. Disponível em: https://dl.acm.org/doi/10.1145/1985793.1985850.

APÊNDICE A – TELAS DO SISTEMA EZALLOCATE

Abaixo estão as capturas de tela realizadas do sistema *web* descrito neste documento anteriormente. Através as figuras abaixo apresentadas, é possível descrever e ter melhor conhecimento das funcionalidades do sistema.

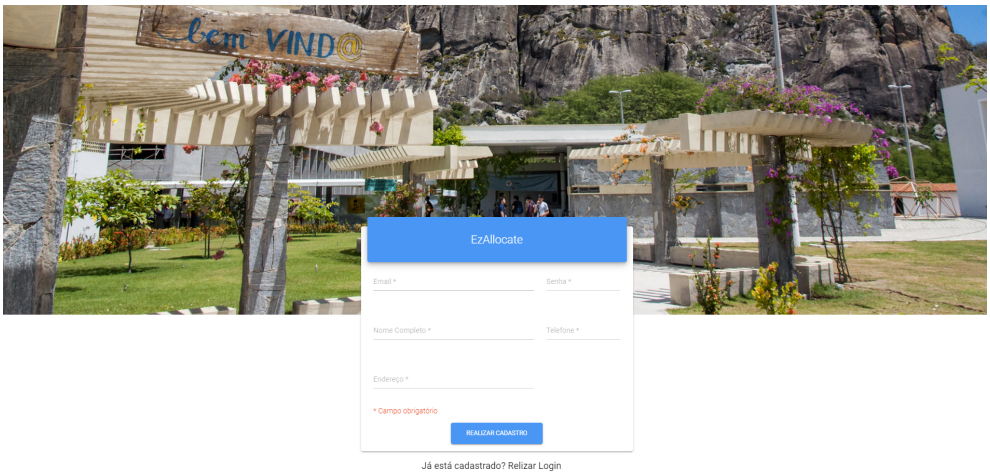
Figura 21 – Login



Fonte: Elaborado pelo autor.

Na Figura 21 é apresentada a tela inicial do sistema, onde é possível realizar *login* usando *email* e senha ou realizando o cadastro de um novo usuário. Caso o usuário digite as credenciais corretamente, será aberta a página da Figura 24. No caso do usuário desejar realizar cadastro no sistema, é aberta a página da Figura 22.

Figura 22 – Cadastro

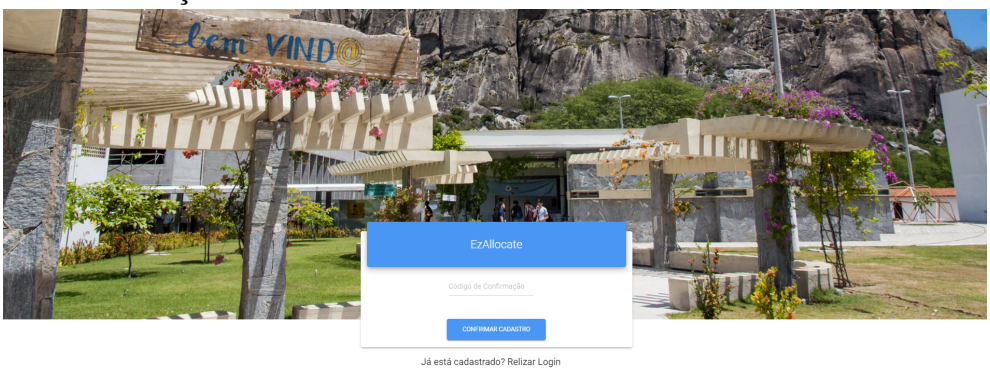


Fonte: Elaborado pelo autor.

Na Figura 22 é possível ver a tela com as informações necessárias para realizar cadastro no sistema. Caso o usuário digito todos os dados é corretamente aberta a página da

Figura 23. Também é possível retornar à página de login da Figura 21 através do botão "Já está cadastrado? Realizar login".

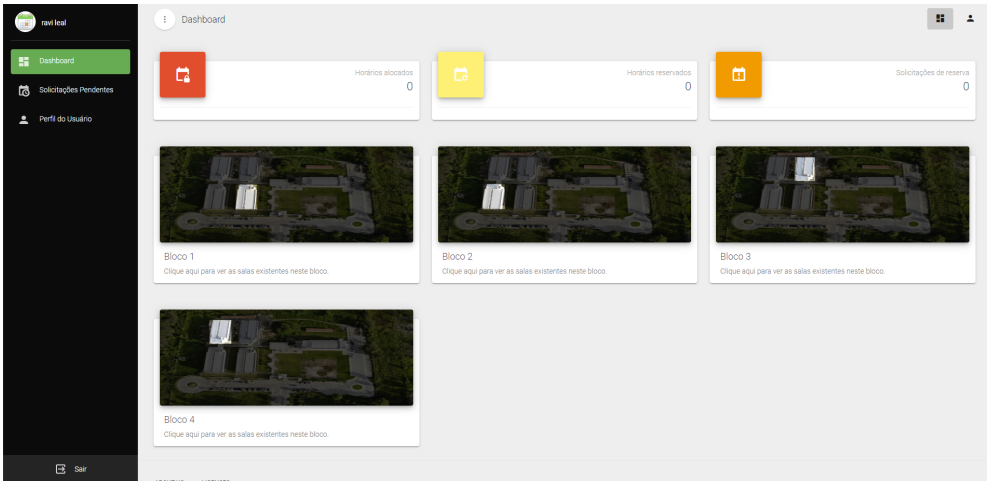
Figura 23 – Confirmação de cadastro



Fonte: Elaborado pelo autor.

Nessa etapa, apresentada na Figura 23 é realizada a confirmação do cadastro do usuário por meio de um código enviado para o *email* digitado na página da Figura 22. Caso o código esteja correto, é aberta a página inicial da Figura 21.

Figura 24 – Visão geral - Blocos

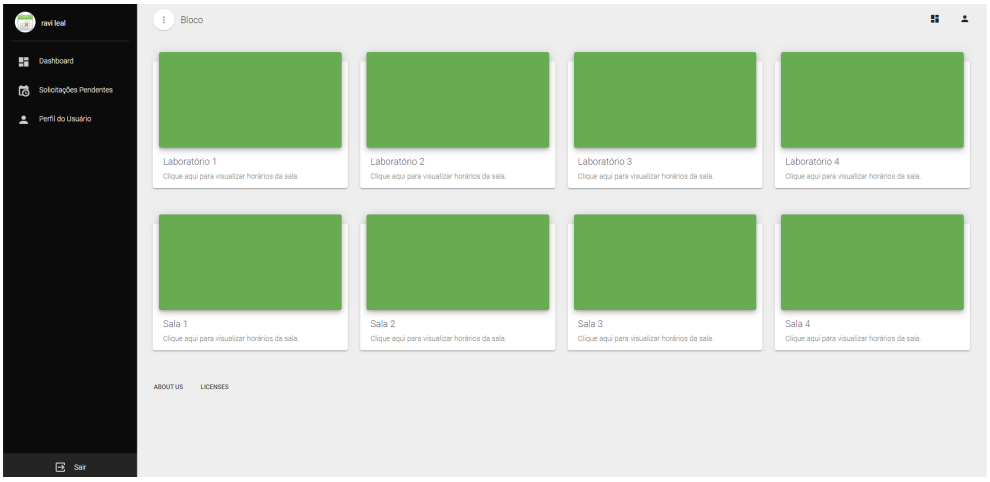


Fonte: Elaborado pelo autor.

Na página de apresentação dos blocos, da Figura 24 é disposto todos os blocos cadastrados previamente no sistema, assim como informações sobre total de horários alocados, reservados e solicitações de reserva. Cada bloco tem uma imagem ilustrativa dos prédios da UFC - Campus Quixadá, com destaque para o respectivo prédio do bloco. No lado direto é apresentado um menu lateral onde estão dispostas as opções principais de acesso do sistema e botão para

encerrar a sessão. Caso o usuário clique em algum bloco será aberta página da Figura 25, já caso o mesmo clique em "Perfil de usuário", será aberta a página da Figura 28. O botão "Solicitações Pendentes", abre a página da Figura 32, e aparece para usuários professores e administradores, que são previamente configurados no sistema. Caso o usuário seja aluno, será apresentado o botão "Minhas Solicitações", que abre a página da Figura 30. O botão do *Dashboard* abre a página principal, descrita na Figura 24.

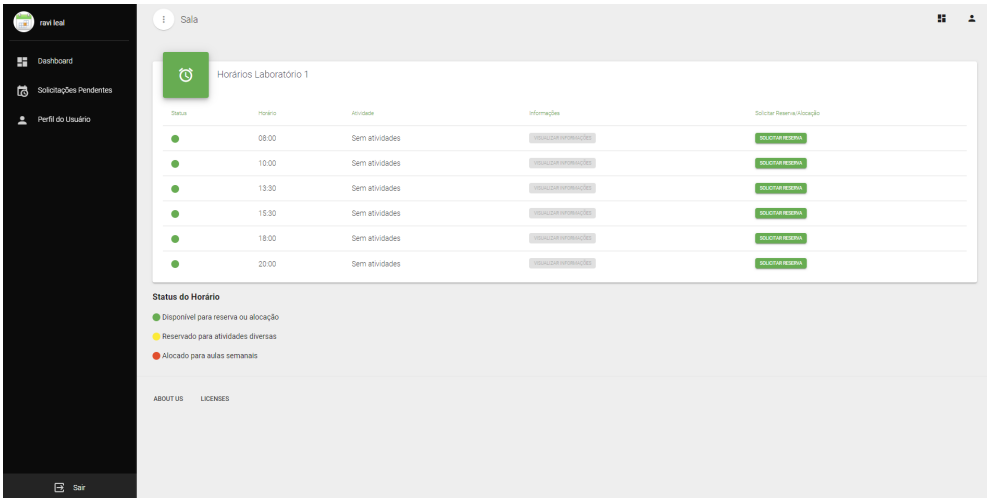
Figura 25 – Visão geral - Salas por bloco



Fonte: Elaborado pelo autor.

Nessa tela da Figura 25, é possível visualizar todas as salas e laboratório cadastrados previamente no sistema conforme o respectivo bloco selecionado. Caso o usuário clique em alguma sala ou bloco, é aberta a página da Figura 26.

Figura 26 – Visão geral - Horários por sala

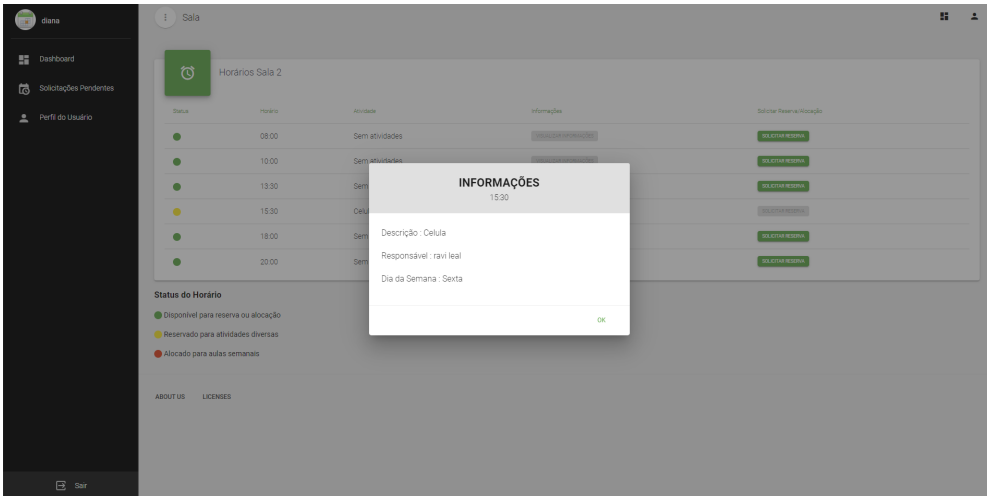


Fonte: Elaborado pelo autor.

A tela da Figura 26 apresenta todos os horários, com suas respectivas informações

de *status*, horário, resumo da atividade, botão para visualizar informações completas do horário e botão para abrir a solicitação de reserva. Os dois últimos botões, respectivamente, abrem o *modal* da Figura 27 e o *modal* da Figura 29. O botão para visualização de informações do horário só é habilitado se houver alguma alocação ou reserva efetuada. Já o *status* do horário pode variar conforme a legenda apresentada na tela. Onde a esfera verde, amarela e vermelha representam, respectivamente, horário livre, reservado para atividades diversas e alocado para aulas.

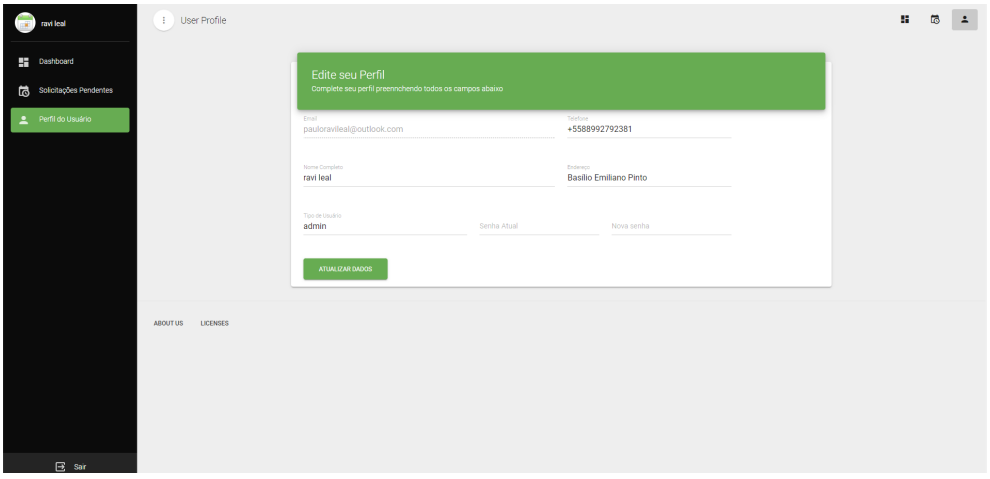
Figura 27 – Visão geral - Informação sobre horários alocados ou reservados



Fonte: Elaborado pelo autor.

As informações apresentadas na Figura 27 tem dados sobre a atividade que será realizada no horário, o responsável e o dia da semana que ocorre.

Figura 28 – Visão geral - Editar Perfil

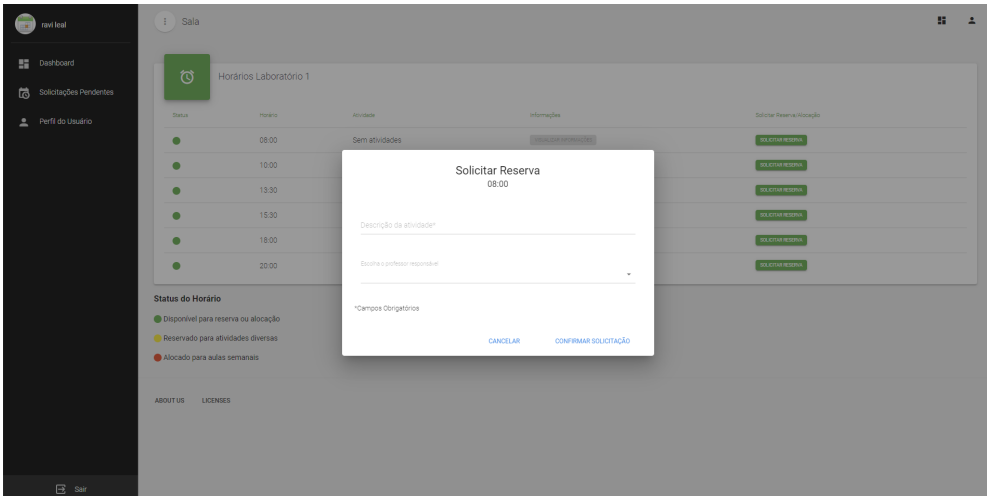


Fonte: Elaborado pelo autor.

A página da Figura 28 possibilita a visualização de informações do usuário da sessão, mesmos dados previamente cadastrados. Também é possível editar informações pessoais, assim

como a senha, que caso não digitada permanece a mesma, clicando no botão de editar.

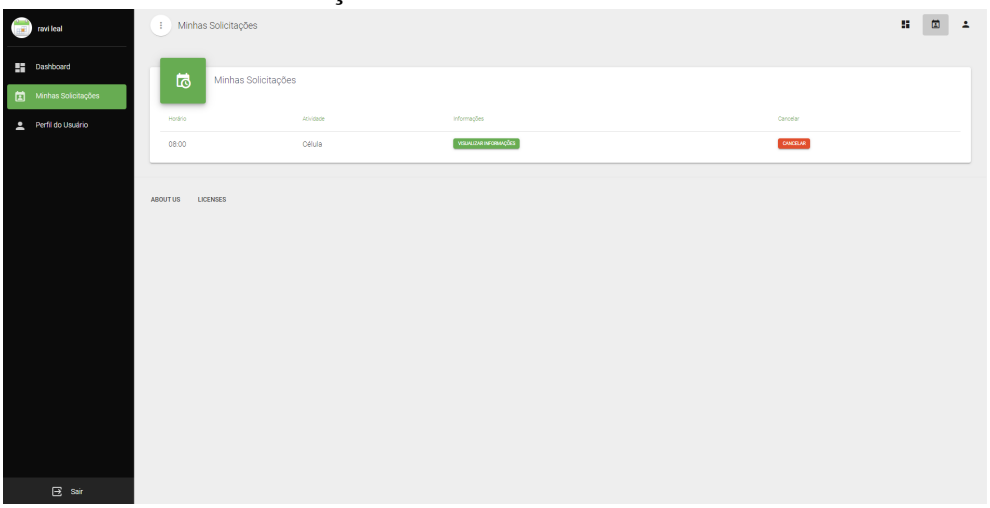
Figura 29 – Visão geral - Solicitar de reserva do horário



Fonte: Elaborado pelo autor.

A solicitação de reserva de horário da Figura 29 precisa de informações de que tipo de atividade será realizada e qual o professor responsável. Caso desejar o usuário confirma a solicitação ou cancelar o processo.

Figura 30 – Visão aluno - Solicitações de reserva

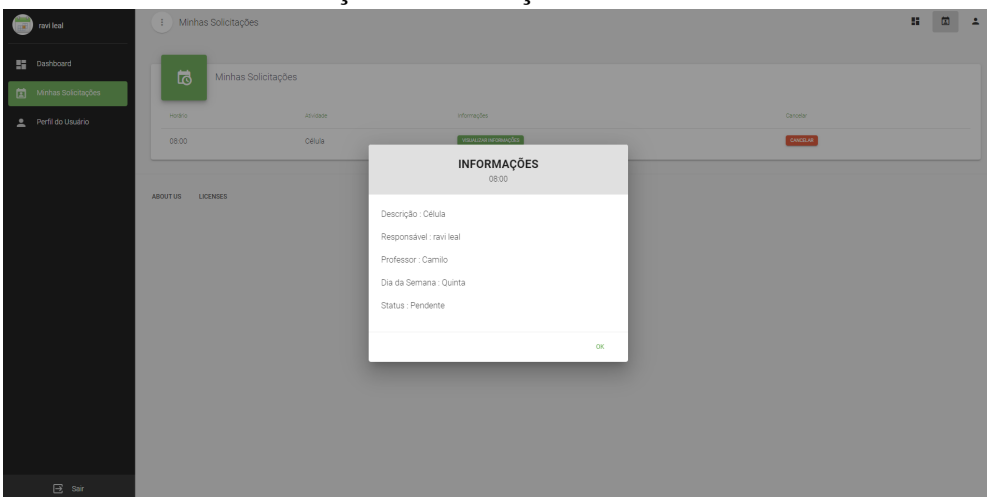


Fonte: Elaborado pelo autor.

Para o aluno é possível ter acesso às solicitações de reserva efetuadas, assim como é apresentado na Figura 30, através do menu lateral descrito anteriormente. Além de informações de horário e atividade solicitada é possível cancelar a solicitação clicando no botão vermelho de cancelar e visualizar todas as informações da solicitação, que abre o *modal* da Figura 31.

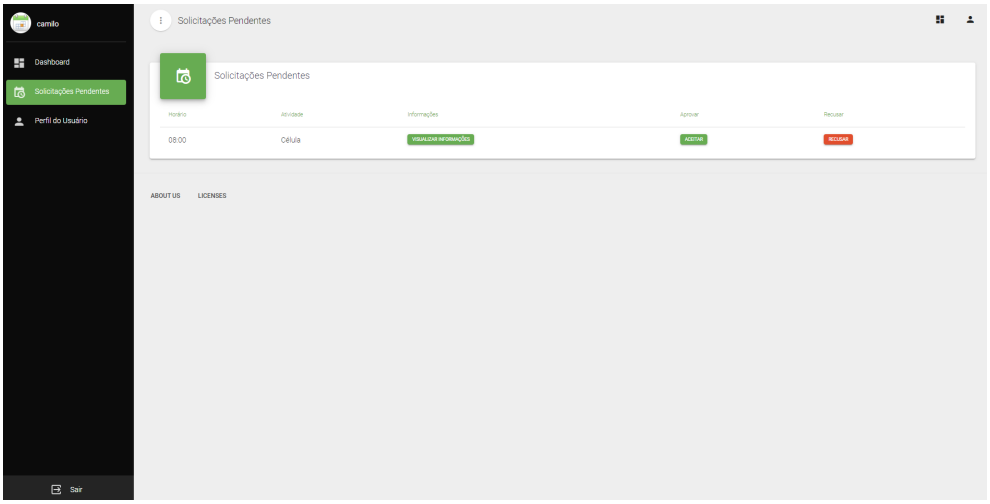
O *modal*, da Figura 31, apresenta as informações sobre a solicitação de reserva como: horário; descrição da atividade, responsável, professor responsável, dia da semana e situação.

Figura 31 – Visão aluno - Informação de solicitação de reserva



Fonte: Elaborado pelo autor.

Figura 32 – Visão professor/administrador - Solicitações pendentes



Fonte: Elaborado pelo autor.

Na Figura 32 é apresentada todas as solicitações de reserva pendentes para professores selecionados e administradores analisarem. A página é bem semelhante a da Figura 30, mas a maior diferença está no botão de recusar, que está em vermelho e serve para encerrar a solicitação com recusa e o botão verde, para aceitar a solicitação. O botão de informações sobre a solicitação apresenta um *modal* semelhante ao da Figura 31.