UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ-CE

CURSO DE GRADUAÇÃO EM REDES DE COMPUTADORES

FRANCISCO LUCAS PINTO DE CASTRO

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB ACADÊMICA UTILIZANDO FERRAMENTAS E PRÁTICAS DEVOPS NA UNIVERSIDADE FEDERAL DO CEARÁ, CAMPUS QUIXADÁ-CE

QUIXADÁ

2025

FRANCISCO LUCAS PINTO DE CASTRO

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB ACADÊMICA UTILIZANDO FERRAMENTAS E PRÁTICAS DEVOPS NA UNIVERSIDADE FEDERAL DO CEARÁ, CAMPUS QUIXADÁ-CE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Redes de Computadores do Campus Quixadá-CE da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de Tecnólogo em Redes de Computadores.

Orientador: Prof. Dr. Antonio Rafael Braga.

QUIXADÁ

2025

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)

C351d Castro, Francisco Lucas Pinto de.
Desenvolvimento de uma aplicação web acadêmica utilizando ferramentas e práticas DevOps na Universidade Federal do Ceará, Campus Quixadá-CE / Francisco Lucas Pinto de Castro. – 2025.
66 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2025.
Orientação: Prof. Dr. Antonio Rafael Braga.

1. Aplicação Web. 2. DevOps. 3. Automação. 4. Integração Contínua. 5. Entrega Contínua. I. Título.
CDD 004.6

FRANCISCO LUCAS PINTO DE CASTRO

DESENVOLVIMENTO DE UMA APLICAÇÃO WEB ACADÊMICA UTILIZANDO FERRAMENTAS E PRÁTICAS DEVOPS NA UNIVERSIDADE FEDERAL DO CEARÁ, CAMPUS QUIXADÁ-CE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Redes de Computadores do Campus Quixadá-CE da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de Tecnólogo em Redes de Computadores.

Aprovada em: 30/07/2025.

BANCA EXAMINADORA

Prof. Dr. Antonio Rafael Braga (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Alisson Barbosa de Souza
Universidade Federal do Ceará - UFC

Prof. Dr. João Marcelo Uchôa de Alencar
Universidade Federal do Ceará - UFC

À minha família, por sua capacidade de acreditar e investir em mim. Mãe, Vó, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação web acadêmica utilizando ferramentas e práticas DevOps, aplicado ao contexto da Universidade Federal do Ceará (UFC), Campus Quixadá-CE. Inicialmente, o projeto surgiu no âmbito do Programa de Acolhimento e Incentivo à Permanência (PAIP) do curso de Redes de Computadores, com o objetivo de criar um sistema eficiente para gerenciamento de monitorias acadêmicas. O desenvolvimento contemplou a definição clara de requisitos funcionais e técnicos, com implementação do *front-end* em *React* e *TailwindCSS*, e *back-end* em *NestJS* com banco de dados *PostgreSQL* utilizando *Prisma ORM*. Para garantir entregas rápidas e estáveis, foi configurado um *pipeline* de integração e entrega contínua (*Continuous Integration* (CI)/*Continuous Delivery* (CD)) com *GitHub Actions*, automatizando processos de *build*, teste e *deploy* das aplicações nas plataformas *Vercel* e *Render*. Além disso, a aplicação foi submetida a um teste de integração entre as ferramentas e um questionário de usabilidade por meio do *User Experience Questionnaire* (UEQ), apresentando resultados positivos relacionados à eficiência, atratividade e confiabilidade. O estudo reconhece explicitamente limitações quanto à mensuração da colaboração e sugere, como trabalho futuro, a utilização de métricas quantitativas mais específicas e ferramentas avançadas de monitoramento contínuo.

Palavras-chave: Aplicação Web; DevOps; Integração Contínua; Entrega Contínua; Automação.

ABSTRACT

This work presents the development of an academic web application using DevOps tools and practices, applied to the context of the UFC, Quixadá Campus. Initially, the project emerged within the scope of the PAIP of the Computer Networks course, with the objective of creating an efficient system for managing academic tutoring sessions. The development included the clear definition of functional and technical requirements, with implementation of the *front-end* in *React* and *TailwindCSS*, and *back-end* in *NestJS* with a *PostgreSQL* database using *Prisma ORM*. To ensure fast and stable deliveries, a *pipeline* of continuous integration and delivery (CI/CD) was configured with *GitHub Actions*, automating *build*, test, and *deploy* processes of the applications on the *Vercel* and *Render* platforms. Additionally, the application was subjected to an integration test between the tools and a usability questionnaire through the UEQ, presenting positive results related to efficiency, attractiveness, and reliability. The study explicitly acknowledges limitations regarding the measurement of collaboration and suggests, as future work, the use of more specific quantitative metrics and advanced continuous monitoring tools.

Keywords: Web Application; DevOps; Continuous Integration; Continuous Delivery; Automation.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AWS	Amazon Web Services
CD	*Continuous Delivery*
CDN	*Content Delivery Network*
CI	*Continuous Integration*
Dev	Desenvolvimento
DevOps	Desenvolvimento e Operações
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IaC	Infraestrutura como Código
MTTR	*Mean Time To Repair*
OPS	Operações
PAIP	Programa de Acolhimento e Incentivo à Permanência
REST	*Representational State Transfer*
SPA	*Single Page Application*
SSL	*Secure Sockets Layer*
TI	Tecnologia da Informação
UEQ	*User Experience Questionnaire*
UFC	Universidade Federal do Ceará
WIP	*Work in Progress*

LISTA DE CÓDIGOS-FONTE

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

A indústria de *Software* é um setor em constante evolução, impulsionado pela necessidade de atender às demandas cada vez mais complexas dos clientes e pela busca por inovação e qualidade em seus produtos. Essa dinâmica exige que as empresas não apenas desenvolvam soluções eficientes, mas também garantam a melhoria contínua de seus processos e serviços (Guerrero *et al.*, 2020). Esse cenário de evolução e demanda por qualidade contrasta com as dificuldades enfrentadas, e com isso as empresas de *software* enfrentam o desafio da entrega rápida e contínua, juntamente com as necessidades de sustentabilidade a longo prazo (Alarcon *et al.*, 2024).

Essa complexidade se torna ainda mais evidente ao considerar que, historicamente, o desenvolvimento de *software* e a administração de infraestrutura de Tecnologia da Informação (TI), eram tratados como áreas distintas, resultando em falhas de comunicação, atrasos na entrega de sistemas e dificuldades na implementação de mudanças. Nesse contexto, a metodologia DevOps surge como uma abordagem inovadora para integrar o desenvolvimento de *software* e as operações de TI, promovendo a colaboração e a automação de processos (Pavan *et al.*, 2018). Essa abordagem tem se tornado essencial para organizações que buscam maior competitividade no mercado, sendo amplamente adotada por empresas de tecnologia e, mais recentemente, por sua eficiêcia nas instituições acadêmicas.

O DevOps chegou ao *mainstream*, em parte, por propor uma solução para a chamada "última milha" do processo de entrega de software. Ele trouxe para a indústria respostas concretas a preocupações frequentemente ignoradas pela engenharia de software tradicional e pela academia. Seu sucesso chamou a atenção do meio acadêmico, levando algumas universidades a incluírem conteúdos relacionados em seus programas de ensino (Paez; Fontela, 2023). No entanto, sua implementação no ambiente universitário ainda representa um desafio, o que pode comprometer a formação de futuros profissionais de TI, especialmente quanto às competências exigidas por um mercado cada vez mais dinâmico.

Diante desse panorama, este trabalho tem como objetivo demonstrar o desenvolvimento de uma aplicação web acadêmica utilizando ferramentas e práticas da cultura DevOps. Este estudo foi realizado através da experiência prática de desenvolvimento de uma aplicação web para o gerenciamento de monitorias no curso de Redes de Computadores. A análise considerará a aplicação dos princípios de integração e entrega contínua, automação e monitoramento, observando os desafios enfrentados, os resultados obtidos e a percepção sobre a efetividade das

práticas DevOps neste contexto.

1.1 Objetivos

Esta seção apresenta o objetivo geral e os objetivos específicos do trabalho.

1.1.1 Objetivo Geral

Demonstrar o desenvolvimento de uma aplicação web acadêmica utilizando ferramentas DevOps para automação e integração contínua no contexto da Universidade Federal do Ceará, Campus Quixadá-Ce.

1.1.2 Objetivos Específicos

- Compreender as práticas DevOps mais relevantes para o contexto acadêmico;
- Analisar ferramentas para gestão e centralização das atividades de desenvolvimento de *software*;
- Desenvolver uma aplicação web utilizando a cultura DevOps.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, são apresentados os conceitos fundamentais que embasam as práticas e metodologias adotadas neste trabalho. Inicia-se com a definição do processo de software, essencial para compreender a estrutura e organização do desenvolvimento de sistemas. Em seguida, discorrem-se os métodos ágeis, que desempenham papel crucial na promoção da flexibilidade e eficiência das equipes de desenvolvimento.

Posteriormente, explora-se o conceito de DevOps, que integra as áreas de desenvolvimento e operações com o objetivo de otimizar o ciclo de vida do software. Em continuidade, aborda-se a prática de CI, responsável pela automação da integração contínua do código, permitindo a detecção rápida de problemas. Por fim, apresenta-se a Entrega Contínua (CD), que complementa a CI ao possibilitar a entrega constante e confiável de software aos usuários finais.

Esses conceitos formam a base para a aplicação das práticas DevOps no desenvolvimento do sistema proposto neste trabalho, com foco na melhoria contínua, automação e qualidade nas entregas.

2.1 Fundamentos do Desenvolvimento de Software

2.1.1 Processo de Software

Existem muitas definições de processo de *software* na literatura, refletindo a complexidade e abrangência do conceito. Segundo Pressman (2011), o processo de software é um conjunto de atividades, métodos e práticas estruturadas para orientar a criação e manutenção de software de maneira eficaz e eficiente. Isso inclui etapas como definição de requisitos, *design*, implementação, testes e manutenção, com o objetivo de garantir que o software atenda às necessidades dos usuários e seja desenvolvido dentro dos padrões de qualidade e prazo estabelecidos.

Em uma perspectiva complementar, Sommerville (2004) define o processo de software como um conjunto de atividades organizadas para desenvolver e manter sistemas, enfatizando a importância de uma estrutura clara e de práticas que assegurem a qualidade e a consistência do produto. Enquanto Pressman (2011) destaca a aplicação prática e os métodos específicos envolvidos, Sommerville (2004) foca na estrutura e organização das atividades necessárias para um desenvolvimento eficaz.

A convergência entre as definições apresentadas revela a natureza dual de um processo de software eficaz. Ele deve ser tanto uma coleção de métodos práticos quanto uma estrutura organizacional coesa. Essa abordagem integrada é o que permite gerenciar a complexidade inerente aos projetos de software e assegurar que a qualidade do produto final não seja uma consequência acidental, mas sim um resultado planejado.

2.1.2 Arquitetura de Aplicações Web

A arquitetura de uma aplicação web diz respeito à estruturação de seus componentes e às interações entre cliente e servidor, sendo crucial para garantir atributos como escalabilidade, manutenibilidade e desempenho. Um dos modelos mais consolidados para esse tipo de arquitetura é o modelo *cliente-servidor*, no qual o cliente, normalmente um navegador web, é responsável pela apresentação da interface ao usuário, enquanto o servidor executa a lógica de negócio e manipula os dados de forma centralizada (Fowler, 2002).

No lado do cliente, uma evolução marcante foi o surgimento das Aplicações de Página Única (*Single Page Application* (SPA)), que substituem o modelo tradicional de navegação por páginas completas. Nessas aplicações, uma única página (*HyperText Markup Language* (HTML)), é carregada inicialmente e, a partir dela, os dados são atualizados dinamicamente por meio de chamadas a *Application Programming Interface* (API)s, proporcionando uma experiência de uso mais fluida e interativa (Haverbeke, 2018). *Frameworks* como o *React*, utilizado neste projeto, são amplamente adotados para viabilizar esse modelo, favorecendo o desenvolvimento modular, reativo e eficiente da interface.

No servidor, a comunicação com o *front-end* é realizada por meio de uma Interface de Programação de Aplicações (*Application Programming Interface* - API). Entre os estilos arquiteturais disponíveis, o *Representational State Transfer* (REST), proposto por Fielding (2000), destaca-se por sua simplicidade e adesão a princípios da web. O estilo arquitetural REST estabelece um conjunto de restrições para o desenvolvimento de sistemas distribuídos baseados em comunicação via protocolo *Hypertext Transfer Protocol* (HTTP), como a separação entre cliente e servidor, a ausência de estado entre as requisições (*statelessness*) e a utilização de uma interface uniforme, baseada em métodos padronizados do protocolo HTTP, o que facilita a manipulação de recursos e contribui para a escalabilidade e previsibilidade das interações.

Dentre os métodos HTTP mais utilizados em API RESTFul, destacam-se:

– **GET**: método destinado à obtenção de informações ou à leitura de um recurso específico,

sem causar alterações no estado do servidor;

- **POST**: utilizado para o envio de dados com o objetivo de criar um novo recurso no servidor;
- **PUT**: empregado para a substituição completa de um recurso existente, atualizando-o com os dados enviados na requisição;
- **DELETE**: responsável por excluir um recurso previamente existente no servidor;
- **PATCH**: utilizado em operações de atualização parcial de um recurso, modificando apenas os campos informados.

No presente trabalho, foi adotado o *framework NestJS* para o desenvolvimento da API RESTful no ambiente de *back-end*. A escolha do *NestJS* justifica-se por sua estrutura robusta, que favorece a organização do código por meio de módulos e a adoção de boas práticas, como a injeção de dependência. Ademais, sua integração nativa com a linguagem *TypeScript* proporciona maior segurança, legibilidade e manutenibilidade ao código, características desejáveis no desenvolvimento de aplicações escaláveis e de longo prazo.

Outro aspecto importante da arquitetura é a organização do código-fonte. Em projetos com múltiplos componentes, como *front-end*, *back-end* e *scripts* de automação, a abordagem de monorepositório (*monorepo*) tem se mostrado vantajosa. Nessa estratégia, todo o código relacionado ao projeto é armazenado em um único repositório, o que facilita a padronização, a automação de tarefas e o gerenciamento de dependências entre os módulos (Potvin; Levenberg, 2016). Essa escolha arquitetural também favorece a aplicação de práticas *DevOps*, pois centraliza os processos de *build*, teste e *deploy* em um único fluxo de integração contínua.

Assim, a arquitetura adotada neste trabalho — composta por um *front-end* baseado em SPA, um *back-end* exposto por meio de API RESTful e uma estrutura unificada em *monorepo* — estabelece as bases necessárias para a automação do ciclo de vida da aplicação. Isso permite a implementação eficiente de práticas como integração e entrega contínuas (CI/CD), automatizando desde a compilação e testes até o *deploy* final dos serviços. O estudo dessa arquitetura, sob a perspectiva conceitual, fornece o embasamento para as decisões técnicas detalhadas ao longo do projeto.

Compreendida a base técnica da infraestrutura, torna-se essencial analisar como as metodologias ágeis se inserem nesse contexto. Essas metodologias trazem flexibilidade e colaboração contínua, indispensáveis para a adaptação rápida e o alinhamento das equipes. A seguir, aprofunda-se a fundamentação teórica desses métodos, explorando seus princípios e a

integração com abordagens como DevOps.

2.2 Métodos Ágeis

Como resposta à rigidez dos modelos tradicionais de desenvolvimento, como o modelo cascata, surgiram os métodos ágeis. Essas abordagens representam uma mudança significativa na condução de projetos de software, priorizando a colaboração contínua com o cliente, a adaptabilidade a mudanças e a entrega de valor de maneira incremental e frequente. Nesta seção, apresentam-se os princípios que fundamentam o movimento ágil, seguidos da descrição de dois frameworks amplamente utilizados: *Scrum* e *Kanban*.

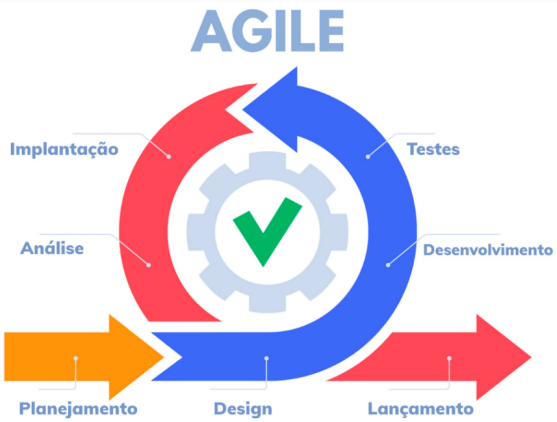
2.2.1 Princípios Ágeis

De acordo com Schwaber e Beedle (2001), os métodos ágeis enfatizam a rápida adaptação às necessidades emergentes e promovem uma comunicação eficaz entre os membros da equipe. Já Fowler (2002) complementa essa visão ao afirmar que as práticas ágeis valorizam a comunicação contínua, iteração curta e *feedback* constante como elementos centrais do processo. Para os autores, o desenvolvimento iterativo, aliado à integração contínua e testes frequentes, é essencial para responder a mudanças de requisitos e garantir entregas de software com qualidade e alinhadas às expectativas do cliente.

Esses princípios foram formalizados no Manifesto Ágil Beck *et al.* (2001), publicado em 2001 por um grupo de especialistas em desenvolvimento de software. O documento estabelece valores fundamentais, como a valorização de indivíduos e interações acima de processos e ferramentas, e a colaboração com o cliente acima da negociação de contratos.

A imagem apresentada na Figura 1 ilustra o fluxo prático dos métodos ágeis na gestão de projetos, conforme discutido por Brandao (2023). O processo inicia com o planejamento, onde são definidos os objetivos do projeto, identificados os requisitos e priorizadas as tarefas de forma iterativa. Em seguida, ocorre a análise, que examina riscos, viabilidade e necessidades do cliente. Na etapa de design, elabora-se a arquitetura, criam-se protótipos e organiza-se a experiência do usuário. A fase de desenvolvimento consiste na implementação incremental das funcionalidades, acompanhada de testes contínuos. Por fim, ocorrem o lançamento e a implantação da solução.

Figura 1 – Fluxo prático de métodos ágeis



Fonte: Brandao (2023)

2.2.2 Scrum

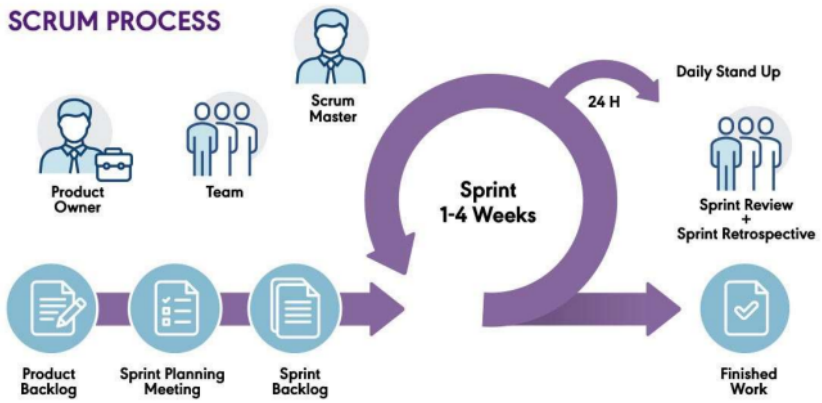
Entre os frameworks ágeis existentes, o *Scrum* destaca-se por sua ampla adoção em projetos que lidam com requisitos voláteis e ambientes dinâmicos. Segundo Sutherland e Schwaber (2020), o *Scrum* é uma estrutura iterativa e incremental voltada à entrega contínua de valor. Seu funcionamento baseia-se em ciclos curtos de desenvolvimento, chamados *sprints*, que geralmente duram de uma a quatro semanas.

O processo inicia com a definição de um *Product Backlog*, uma lista priorizada de funcionalidades mantida pelo *Product Owner*. A equipe de desenvolvimento seleciona, durante a *Sprint Planning*, os itens que serão implementados no ciclo. Ao longo da sprint, o progresso é acompanhado por reuniões diárias chamadas *Daily Stand-ups*, facilitadas pelo *Scrum Master*, que atua removendo impedimentos e zelando pela aplicação da metodologia. Ao final, realizam-se a *Sprint Review*, para apresentação do incremento gerado, e a *Sprint Retrospective*, voltada à identificação de melhorias no processo (Silva, 2023).

A Figura 2 ilustra esse fluxo de atividades, destacando os papéis e artefatos dos passos do *Scrum*, proporcionando uma visualização clara da dinâmica do processo.

Essa abordagem possibilita maior transparência, colaboração e eficiência, garantindo que o produto final atenda melhor às expectativas dos stakeholders.

Figura 2 – Processo e fases do framework ágil Scrum



Fonte: Galvo (2023)

2.2.3 Kanban

Outra metodologia amplamente utilizada é o *Kanban*, cujo foco está na visualização e otimização do fluxo de trabalho. Conforme Ahmad *et al.* (2013), o *Kanban* utiliza quadros visuais para representar as etapas do processo e controlar o *Work in Progress* (WIP), limitando a quantidade de tarefas em execução simultaneamente. Essa limitação ajuda a identificar gargalos e promover um fluxo de trabalho contínuo e sustentável.

O objetivo principal do *Kanban* é melhorar a eficiência do processo por meio de ajustes incrementais e baseados em dados. Ao contrário de frameworks mais estruturados, o *Kanban* permite uma adoção gradual, sendo ideal para equipes que buscam evoluir seus processos de forma orgânica.

Em síntese, os métodos ágeis oferecem uma base sólida para a construção de soluções flexíveis, iterativas e voltadas à geração de valor. Sua ênfase na entrega incremental, na colaboração entre *stakeholders* e na melhoria contínua contribui diretamente para práticas mais integradas e automatizadas no desenvolvimento de *software*, fundamentos que serão aprofundados na próxima seção, dedicada ao DevOps.

2.3 DevOps

A cultura DevOps surge como uma evolução natural dos princípios ágeis, estendendo-os para além da equipe de desenvolvimento e abrangendo todo o ciclo de vida do *software*. Trata-se de uma filosofia cultural e um conjunto de práticas que visam unificar o Desenvolvimento

(Dev), focado na criação de novas funcionalidades, e as Operações (OPS), responsáveis pela estabilidade e entrega do ambiente produtivo (Kim *et al.*, 2016). Essa integração busca superar as barreiras tradicionais entre essas áreas, promovendo um fluxo de trabalho mais coeso e eficiente.

Reforçando o caráter prático e cultural dessa união, Doernenburg (2018) define DevOps como uma prática que integra o desenvolvimento de *software* e as operações de TI para melhorar a colaboração e a comunicação entre essas equipes. Mais do que um conjunto de ferramentas, representa uma mudança cultural que busca quebrar barreiras tradicionais entre os times, promovendo uma abordagem mais ágil e responsiva. O objetivo primordial é automatizar e monitorar os processos de construção, teste e implantação de aplicações, permitindo entregas mais rápidas, frequentes e confiáveis, diminuindo assim a lacuna entre Desenvolvimento e Operações e resultando em maior qualidade, velocidade e confiabilidade das entregas de sistemas.

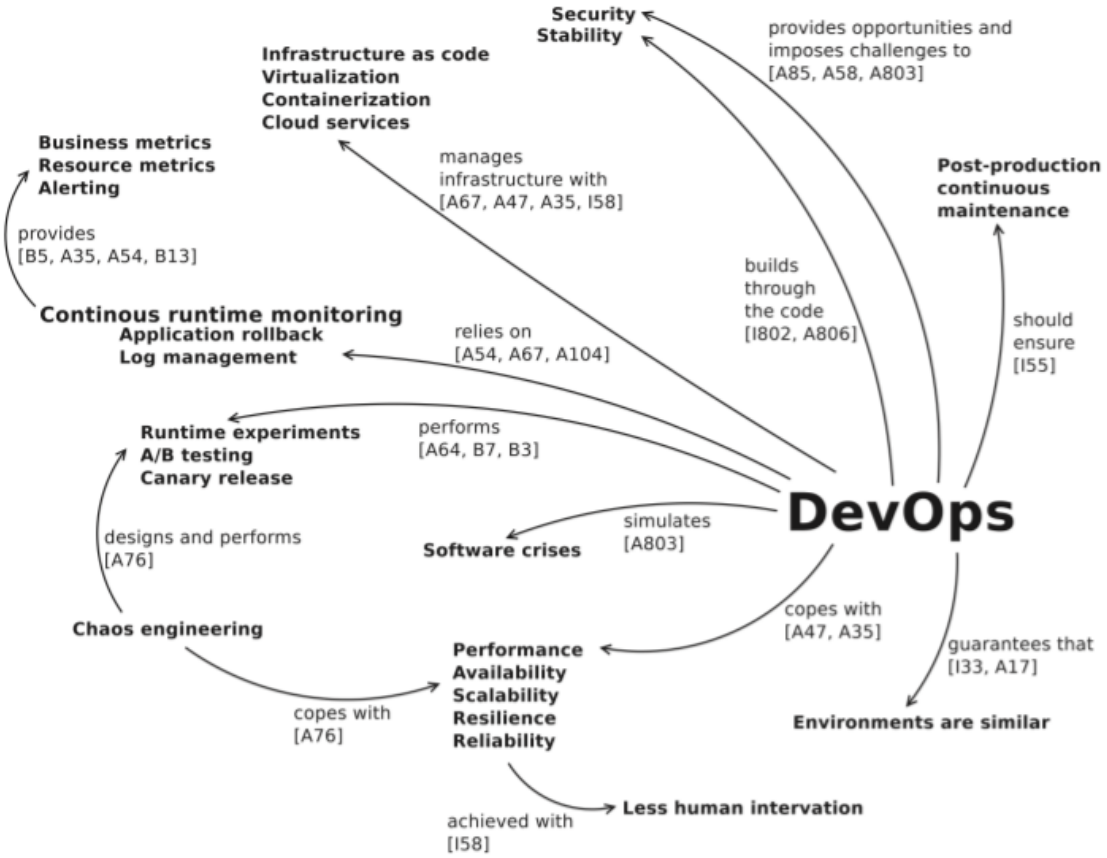
Muitos estudos abordam essa união e tentam sistematizar os conceitos e práticas que compõem o ecossistema DevOps. Um desses trabalhos é o de Leite *et al.* (2020), que realizou um mapeamento sistemático da literatura para identificar, agrupar e categorizar os principais elementos presentes na cultura DevOps, com ênfase nos desafios operacionais enfrentados durante a execução de sistemas em produção.

A Figura 3 apresenta a estrutura resultante dessa análise, organizada em torno da categoria de tempo de execução (*runtime*), no qual os conceitos estão agrupados em oito categorias principais, como o monitoramento contínuo, que enfatiza a coleta de métricas em tempo real para auxiliar no diagnóstico de falhas e no *feedback* contínuo; resiliência, que trata de estratégias para tolerância a falhas e recuperação rápida; *rollback*, referente à capacidade de desfazer implantações com segurança; observabilidade, voltada à visibilidade completa do comportamento do sistema; *feature toggles*, utilizadas para controle de funcionalidades em produção; e Infraestrutura como Código (IaC), que permite o provisionamento automatizado e reprodutível de ambientes computacionais.

A partir dessa compreensão integrada das práticas e fundamentos que formam a cultura DevOps, é possível observar que ela vai além da utilização de ferramentas, consolidando-se como uma abordagem estratégica de desenvolvimento e operação de sistemas. Neste trabalho, essa cultura foi aplicada tanto no desenvolvimento da aplicação quanto na gestão do projeto, influenciando a seleção de ferramentas, a organização do repositório e os processos de entrega e automação.

Nas próximas subseções, abordam-se os aspectos práticos da adoção das principais

Figura 3 – Categorias de conceitos associados ao ambiente de execução na cultura DevOps, conforme sistematizado por Leite et al. (2020).



Fonte: Adaptado de Leite *et al.* (2020)

técnicas de DevOps discutidas neste estudo, com ênfase em Integração Contínua (CI), Entrega Contínua (CD), Monitoramento, *Feedback* e Infraestrutura como Código (IaC), todos alinhados com a proposta de um ciclo de desenvolvimento ágil, confiável e automatizado.

2.3.1 Integração Contínua (CI)

A Integração Contínua é uma prática na qual os desenvolvedores integram seu trabalho ao repositório principal com frequência, muitas vezes várias vezes ao dia (Shahin *et al.*, 2017). Cada integração é verificada por um processo de construção e teste automatizado, permitindo que as equipes detectem problemas o mais cedo possível. No contexto prático deste trabalho, a CI foi implementada por meio do *GitHub Actions*. A cada *commit* enviado ao repositório, um fluxo de trabalho automatizado é acionado para compilar a aplicação e executar uma suíte de testes, garantindo que novas alterações não introduzam regressões no código-fonte.

2.3.2 Entrega Contínua (CD)

Complementar à CI, a Entrega Contínua (CD) é uma prática que visa garantir que novas versões do *software* possam ser disponibilizadas de forma frequente e automatizada, minimizando a intervenção manual e os riscos associados a implantações manuais. De acordo com Shahin *et al.* (2017), a CD estende a automação da CI até os ambientes de homologação e produção, possibilitando que, uma vez aprovadas todas as etapas de teste, o software seja implantado de forma automática e segura.

Essa abordagem reduz o tempo de entrega, melhora a rastreabilidade das mudanças e está alinhada aos princípios da cultura DevOps, ao favorecer ciclos de desenvolvimento rápidos e confiáveis. Além disso, permite maior controle de versões, *feedback* contínuo e rápida reversão em caso de falhas.

No contexto deste projeto, a CD foi configurada para operar imediatamente após a validação bem-sucedida dos testes automatizados no fluxo de CI. Para isso, foram utilizados *workflows* do *GitHub Actions*, os quais realizam o *deploy* automatizado do *front-end* na plataforma Vercel e do *back-end* na Render, que oferece suporte à execução da API e ao gerenciamento do banco de dados relacional PostgreSQL. Essa arquitetura é compatível com os principais conceitos de nuvem e automação contínua discutidos nesta seção, sendo adequada para o ambiente acadêmico adotado.

Essa configuração assegura que uma versão funcional, testada e atualizada da aplicação esteja sempre disponível em produção, reforçando os princípios de automação e confiabilidade essenciais à engenharia moderna de software.

2.3.3 Feedback Contínuo e Monitoramento

O ciclo DevOps não termina na entrega. O monitoramento contínuo é a prática de observar a aplicação em produção, coletando dados sobre seu desempenho, erros e utilização. Esse monitoramento alimenta um ciclo de *feedback* contínuo, onde as informações coletadas são usadas para guiar o próximo ciclo de desenvolvimento, permitindo que as equipes respondam rapidamente a problemas e tomem decisões informadas sobre novas funcionalidades (Kim *et al.*, 2021). Para este trabalho, a análise das percepções e vantagens da implementação do DevOps dependeu intrinsecamente do monitoramento dos *deploys* e do *feedback* da equipe sobre a estabilidade e a velocidade proporcionadas pela automação.

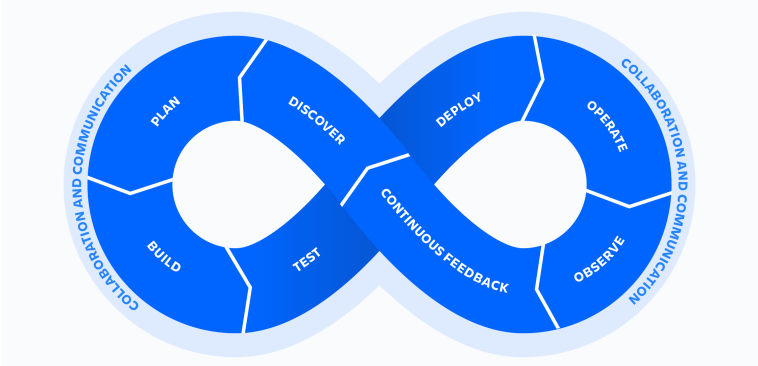
2.3.4 Relação do Projeto com as Etapas do Ciclo DevOps

A Figura 4 apresenta o ciclo DevOps em formato de infinito, representando o modelo contínuo e integrado utilizado neste projeto. Cada etapa desse ciclo foi claramente contemplada:

- **Planejamento**: definição clara de requisitos da aplicação.
- **Construção**: implementação das camadas *front-end* e *back-end*.
- **Teste**: execução automatizada de testes no pipeline CI.
- **Feedback Contínuo**: utilização de logs automáticos e avaliação de usabilidade com usuários (UEQ).
- **Deploy**: implantação automática nas plataformas Vercel e Render.
- **Operação**: gerenciamento dos ambientes produtivos Vercel e Render.
- **Observação**: monitoramento básico contínuo fornecido pelas plataformas escolhidas.
- **Descoberta**: identificação explícita de melhorias futuras com base no feedback coletado.

Reconheceu-se explicitamente a limitação em relação à colaboração contínua, recomendando-se que em trabalhos futuros sejam adotadas métricas mais objetivas e ferramentas específicas para mensurar e fomentar esta colaboração.

Figura 4 – Ciclo clássico DevOps representado em formato de infinito.



Fonte: https://www.atlassian.com/br/devops

2.3.5 Infraestrutura como Código (IaC)

A Infraestrutura como Código (IaC) é uma prática que permite a definição e o gerenciamento de recursos computacionais por meio de arquivos versionáveis, substituindo a configuração manual por automações (Rani; Karthika, 2024). Segundo Spinellis (2018), a IaC representa uma mudança de paradigma na administração de sistemas, ao aplicar princípios da engenharia de software ao provisionamento de infraestrutura.

De acordo com Humble e Farley (2010), a adoção da IaC promove consistência, rastreabilidade e reprodutibilidade na entrega de ambientes, sendo um dos pilares da integração e entrega contínuas em projetos modernos. Tais características estão alinhadas aos princípios da cultura DevOps, frequentemente aplicada em contextos de desenvolvimento ágil em TI.

Dentre as ferramentas que viabilizam a IaC, destacam-se o *Terraform*[1], *AWS CloudFormation*[2], *Ansible*[3] e *Pulumi*[4]. Essas ferramentas permitem a definição de servidores, redes, bancos de dados e serviços em nuvem de forma declarativa, permitindo versionamento, *rollback* e replicação de ambientes com confiabilidade.

Neste trabalho, a aplicação foi estruturada em um monorepositório que integra o *front-end* e o *back-end*. O *front-end*, desenvolvido em *React*, é implantado automaticamente pela plataforma Vercel por meio de *pipelines* de *Continuous Deployment*. Já o *back-end*, desenvolvido com *NestJS*, é hospedado na plataforma Render, que permite a automação do *deploy* contínuo via integrações com o *GitHub Actions* e oferece suporte nativo a bancos de dados PostgreSQL. Esse arranjo mantém os princípios de automação, escalabilidade e rastreabilidade da infraestrutura, conforme preconizado pelas práticas modernas de DevOps.

Essa abordagem garante escalabilidade, confiabilidade e rastreabilidade ao ciclo de vida da infraestrutura, reduzindo falhas humanas e otimizando a produtividade da equipe. Como apontado por Morris (2020), a IaC se tornou um elemento essencial para o gerenciamento seguro e eficiente de sistemas distribuídos em nuvem.

Após a apresentação dos conceitos e fundamentos que embasam o desenvolvimento deste trabalho, torna-se relevante analisar como a literatura tem tratado a implementação da cultura DevOps em diferentes contextos. O próximo capítulo, portanto, apresenta e discute estudos que exploram desafios, estratégias e resultados relacionados à adoção de DevOps.

[1] https://www.terraform.io/
[2] https://aws.amazon.com/cloudformation/
[3] https://www.ansible.com/
[4] https://www.pulumi.com/

3 TRABALHOS RELACIONADOS

Esta seção apresenta estudos que discutem a implementação da cultura DevOps em diferentes contextos, com destaque para desafios enfrentados, abordagens adotadas e resultados obtidos. Ao final, uma tabela comparativa sintetiza os principais elementos de cada estudo.

3.1 Da teoria à prática: Desafios da Implantação da cultura DevOps na rotina de empresa de desenvolvimento de sistemas

No trabalho de Beppler (2023), foi investigada a aplicação da cultura DevOps em uma empresa de desenvolvimento de sistemas, com o objetivo de compreender os desafios enfrentados durante a adoção dessa abordagem. A pesquisa foi conduzida em três fases: revisão bibliográfica, coleta de dados e análise dos resultados obtidos por meio de questionários aplicados aos profissionais das áreas de desenvolvimento web e suporte técnico da empresa.

A análise dos dados revelou diversos obstáculos, como a centralização de informações, divergências de comunicação entre as equipes e dificuldades na visualização dos processos e resultados. Apesar da adoção das práticas DevOps, a integração entre os setores ainda era limitada, o que comprometia a eficiência esperada. O estudo conclui que, para uma implementação mais eficaz da cultura DevOps, é essencial investir em estratégias que favoreçam a comunicação e o compartilhamento de informações entre as áreas envolvidas.

3.2 Os desafios na adoção de DevOps em uma empresa brasileira de tecnologia da informação no setor bancário

O estudo de Correa (2017) analisou o processo de adoção da cultura DevOps em uma empresa brasileira de tecnologia da informação que atua no setor bancário. A pesquisa buscou identificar os benefícios, desafios e limitações da aplicação do DevOps em um ambiente caracterizado por forte regulamentação, sistemas legados e alta demanda por segurança e estabilidade.

Durante a implementação, foram enfrentadas diversas dificuldades, como resistência à mudança por parte das equipes, barreiras na integração entre setores e a complexidade dos sistemas já existentes. A pesquisa destaca que, embora o DevOps ofereça ganhos em agilidade e colaboração, sua adoção em contextos altamente regulados exige um esforço adicional de adaptação.

A principal conclusão do estudo é que o sucesso na implantação da cultura DevOps depende não apenas da escolha de ferramentas e práticas técnicas adequadas, mas também de mudanças estruturais e culturais dentro da organização. O estudo reforça a importância do alinhamento entre áreas técnicas e estratégicas para que a abordagem seja eficaz em ambientes corporativos complexos.

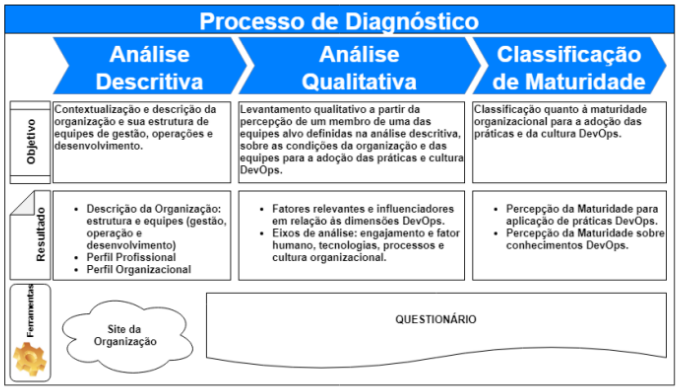
3.3 *DevOps Maturity Diagnosis - A Case Study in Two Public Organizations*

O trabalho de Gimenez e Santos (2020) teve como objetivo diagnosticar o nível de maturidade das práticas DevOps em duas organizações públicas brasileiras. Para isso, os autores utilizaram entrevistas, questionários e análise documental, com base em um modelo de maturidade que avaliou aspectos como automação, integração contínua e colaboração entre equipes.

A pesquisa revelou que as duas organizações se encontravam em estágios diferentes de adoção da cultura DevOps. Enquanto uma apresentava práticas mais consolidadas, a outra estava em fase inicial, com pouca automatização e integração entre as áreas. As lacunas identificadas foram relacionadas principalmente à falta de padronização de processos, baixa adesão a ferramentas de integração e resistência à mudança organizacional.

A Figura 5 ilustra o processo proposto pelos autores para avaliar a maturidade DevOps nas organizações analisadas, detalhando as etapas de coleta de dados, avaliação dos indicadores e identificação de oportunidades de melhoria.

Figura 5 – Processo de Diagnóstico para adoção DevOps



Fonte: (Gimenez; Santos, 2020)

O estudo conclui que a aplicação de modelos de maturidade é uma estratégia eficaz para mapear o estágio atual da organização, orientar ações de melhoria e facilitar a transição para uma cultura DevOps mais estruturada e eficiente.

3.4 Industrial DevOps

O artigo de Hasselbring *et al.* (2019) explora a aplicação de práticas DevOps em ambientes industriais, com o objetivo de analisar como a automação, a integração contínua e a entrega contínua podem transformar os processos de desenvolvimento e operação. O estudo destaca a necessidade de adaptação contínua dessas práticas para atender às demandas específicas de organizações industriais, que frequentemente operam em ambientes críticos e altamente estruturados.

A Figura 6 apresenta o processo contínuo de adaptação e melhoria descrito pelos autores. Essa figura representa a dinâmica evolutiva da adoção do DevOps nas organizações, evidenciando as interações entre os ciclos de planejamento, execução, monitoramento e ajuste, que buscam garantir maior agilidade e confiabilidade nas entregas.

Figura 6 – Processo contínuo de adaptação e melhoria do DevOps industrial

Fonte: (Hasselbring *et al.*, 2019)

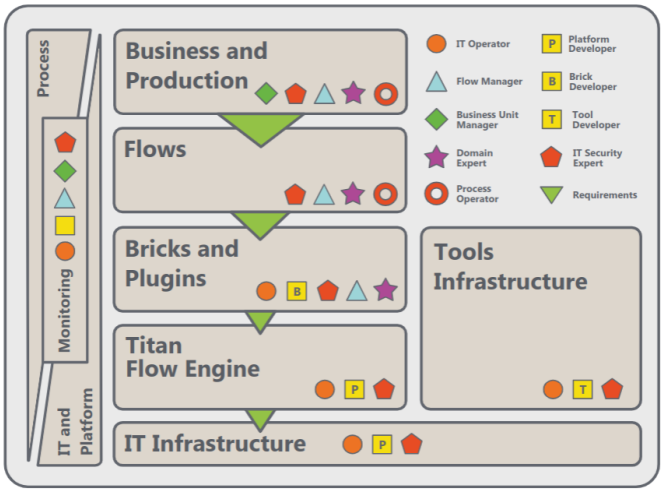
Além disso, o estudo propõe o modelo de referência denominado TITAN, um acrônimo formado por cinco pilares fundamentais para a adoção eficaz da cultura DevOps em ambientes industriais: *Testing*, *Integration*, *Tracking*, *Automation* e *Networking*. Esses pilares representam as dimensões técnicas e organizacionais que devem ser desenvolvidas de forma integrada para que o DevOps alcance maturidade e gere resultados consistentes.

A Figura 7 apresenta a estrutura visual do modelo TITAN. Cada componente do

acrônimo possui uma função específica:

- **Testing** – refere-se à automação dos testes em diferentes estágios do ciclo de desenvolvimento, garantindo que falhas sejam detectadas precocemente e que o código entregue seja confiável;
- **Integration** – trata da integração contínua entre os diferentes módulos do sistema, promovendo um fluxo de trabalho colaborativo e sincronizado;
- **Tracking** – diz respeito ao monitoramento e rastreamento contínuo das versões, mudanças e desempenho das aplicações em tempo real;
- **Automation** – abrange a automação de tarefas recorrentes, como builds, testes e *deploys*, reduzindo o esforço manual e o risco de erros;
- **Networking** – representa a colaboração e a comunicação entre equipes e sistemas, essencial para garantir a fluidez e a visibilidade das operações.

Figura 7 – Modelo TITAN – Estrutura de práticas DevOps para ambientes industriais



Fonte: (Hasselbring *et al.*, 2019)

Essa organização em pilares torna o modelo aplicável em diversos níveis de maturidade, funcionando como um guia para orientar decisões estratégicas e operacionais. O uso do TITAN permite identificar pontos de melhoria e alinhar práticas DevOps com os objetivos de eficiência, rastreabilidade e estabilidade exigidos pela indústria.

Os autores concluíram que a adoção de DevOps no setor industrial pode gerar melhorias significativas na colaboração entre equipes, na automação de processos e na redução do tempo de ciclo do desenvolvimento. Práticas como automação de testes, integração contínua e entrega contínua são destacadas como fundamentais para elevar a qualidade dos produtos

entregues. Além disso, o estudo ressalta que a cultura DevOps, quando bem implementada, contribui para um ambiente de trabalho mais coeso e adaptável às constantes mudanças do setor industrial.

3.5 Comparativo

Os estudos analisados apresentam diversas abordagens e contextos para a implementação da cultura DevOps, cada um com suas particularidades. O trabalho de Beppler (2023) adotou uma abordagem prática e metodológica, identificando desafios como a comunicação deficiente e a centralização excessiva de informações em uma empresa de desenvolvimento de sistemas. Correa (2017), por sua vez, explorou a introdução do DevOps no setor bancário, evidenciando obstáculos como resistência cultural e a presença de sistemas legados.

O estudo de Gimenez e Santos (2020) analisou o nível de maturidade DevOps em organizações públicas, utilizando um modelo de avaliação que permitiu mapear pontos de melhoria e a aderência às boas práticas. Já Hasselbring *et al.* (2019) propuseram o modelo TITAN para aplicação de DevOps em ambientes industriais, enfatizando pilares como automação, integração e rastreabilidade.

O Quadro 1 sintetiza as principais características de cada estudo, comparando os contextos, abordagens, desafios, tecnologias adotadas e objetivos.

Quadro 1 – Comparativo dos Estudos sobre Práticas DevOps em Diferentes Contextos

Trabalho	**Contexto**	**Abordagem**	**Desafios Principais**	**Tecnologias / Ferramentas**	**Objetivo do Estudo**
Beppler (2023)	Empresa de TI	Metodológica prática	Comunicação deficiente, centralização	Ferramentas internas	Melhorar a colaboração entre equipes
Correa (2017)	Setor bancário	Adoção e análise de obstáculos	Resistência cultural, sistemas legados	Integrações legadas	Superar barreiras à adoção do DevOps
Gimenez e Santos (2020)	Órgãos públicos	Diagnóstico de maturidade	Baixa adesão, necessidade de melhoria	Modelo de Maturidade DevOps	Avaliar estágio e propor melhorias
Hasselbring et al. (2019)	Indústria	Aplicação estruturada	Integração, automação de processos	Modelo TITAN	Aumentar eficiência e integração entre equipes
Trabalho proposto	Academia (UFC)	Implementação prática em projeto real	Adaptação ao contexto estudantil, colaboração	*GitHub Actions*, CI/CD	Avaliar a aplicação da cultura DevOps em um projeto acadêmico real

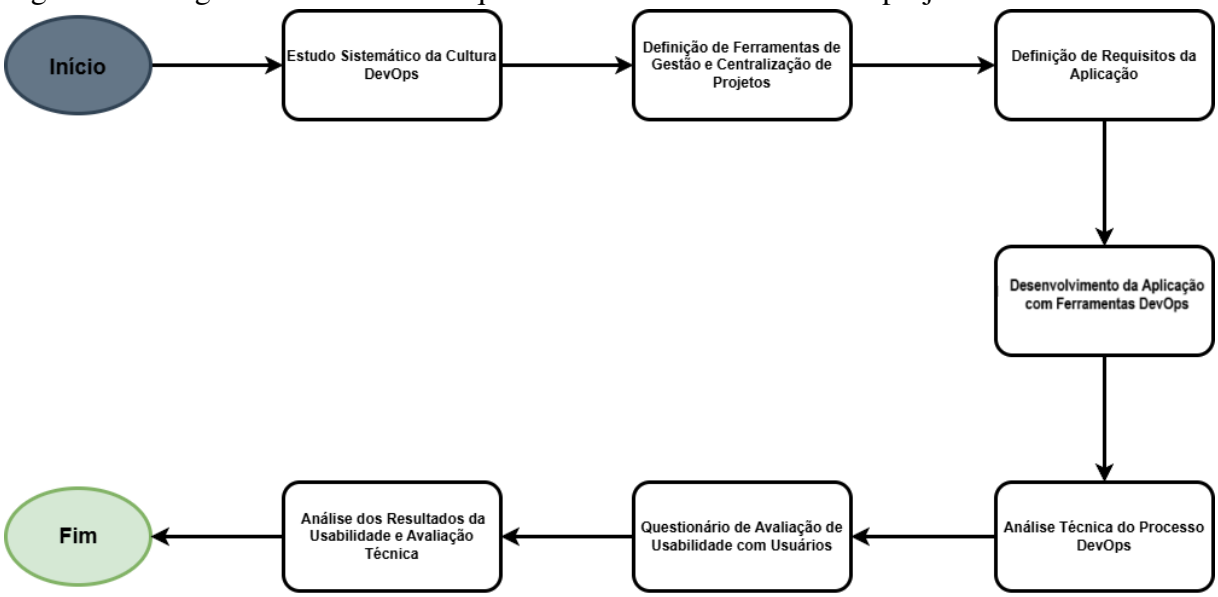
Fonte: Elaborado pelo autor (2024).

Com base na análise, observa-se que os principais desafios enfrentados nos diferentes contextos giram em torno da integração entre equipes, resistência à mudança e necessidade de automatização de processos. Embora o foco e os ambientes de aplicação variem — como o setor bancário, industrial, público e acadêmico — os objetivos convergem para a melhoria da eficiência e da colaboração nos ciclos de entrega de software.

4 METODOLOGIA

Esta seção descreve o procedimento metodológico empregado na realização deste trabalho. Inicialmente, começou-se um levantamento bibliográfico da cultura DevOps, seguido pela definição das ferramentas utilizadas e dos requisitos da aplicação. Em seguida, procedeu-se ao desenvolvimento da aplicação utilizando as ferramentas DevOps. A metodologia também incluiu um teste técnico de integração das ferramentas, bem como a avaliação de usabilidade com usuários. A Figura 8 apresenta, de forma resumida, as etapas que compõem o processo metodológico adotado.

Figura 8 – Diagrama das atividades que foram executadas durante o projeto.



Fonte: Elaborado pelo autor (2025)

4.1 Estudo Sistemático da Cultura DevOps

A primeira etapa deste trabalho consistiu em um levantamento bibliográfico sobre a cultura *DevOps*, com o objetivo de compreender as práticas fundamentais dessa abordagem, como a Integração Contínua (CI), a Entrega Contínua (CD), a automação de processos e a colaboração entre equipes de desenvolvimento e operações. Este estudo envolveu a análise de fontes acadêmicas, como artigos e estudos de caso publicados, além da experimentação prática das ferramentas e processos associados a *DevOps*.

A pesquisa foi conduzida com base nos conceitos apresentados na fundamentação teórica deste trabalho, que explora a aplicação de *DevOps* em diferentes contextos e suas

vantagens no ciclo de vida do desenvolvimento de *software*. Durante este estudo, foram analisadas ferramentas de CI/CD e métodos para integração e automação, buscando uma aplicação prática no projeto de desenvolvimento da aplicação web proposta.

Os principais aspectos abordados neste estudo incluem:

- Compreensão dos conceitos de *DevOps*, CI e CD;
- Adoção de ferramentas de automação, como *GitHub Actions*, para integrar e automatizar o processo de desenvolvimento e *deploy*;
- Avaliação dos desafios de integração entre equipes de desenvolvimento e operações;
- Implementação prática das metodologias *DevOps* no contexto acadêmico, com foco no gerenciamento de monitorias no curso de Redes de Computadores.

Esse estudo visa alinhar os conceitos teóricos de *DevOps* com a aplicação prática no desenvolvimento da aplicação, explorando como as práticas de automação, integração e entrega contínua podem ser implementadas de maneira eficaz em um ambiente acadêmico. Os resultados desta etapa fundamentam as decisões técnicas detalhadas nos tópicos seguintes da metodologia.

4.2 Definição de Ferramentas de Gestão e Centralização de Projetos

Para atender às necessidades deste projeto, foram selecionadas ferramentas específicas para controle de versão, integração contínua, entrega contínua e hospedagem da aplicação. A escolha dessas ferramentas foi pautada principalmente pela aderência aos princípios da cultura Desenvolvimento e Operações (DevOps), ampla documentação disponível, facilidade de integração entre si, gratuidade e adequação ao ambiente acadêmico. As ferramentas escolhidas foram: *GitHub*, *GitHub Actions*, *Vercel* e *Render*.

O *GitHub* foi selecionado como plataforma de controle de versão distribuído utilizando o sistema *Git*. Destaca-se pela popularidade no ambiente acadêmico, gratuidade para repositórios públicos e pela facilidade no gerenciamento colaborativo de código-fonte. Além disso, oferece uma interface intuitiva para gerenciamento de *branches*, *pull requests*, revisões de código e resolução de conflitos, aspectos fundamentais para estudantes e equipes em projetos acadêmicos.

Complementando o uso do *GitHub*, foi adotado o *GitHub Actions* como ferramenta de automação de CI/CD. Essa escolha foi motivada pela integração nativa com o *GitHub*, simplicidade na definição de *pipelines* via arquivos declarativos YAML e gratuidade para projetos públicos, características ideais para a aprendizagem prática dos conceitos de automação e entrega

contínua.

Para a hospedagem e publicação contínua do *front-end*, foi utilizada a plataforma *Vercel*. Essa ferramenta oferece integração direta com o *GitHub*, automatizando o processo de *deploy* a cada modificação aprovada no repositório. Destacam-se entre seus benefícios a facilidade de configuração inicial, a escalabilidade automática, o suporte a domínios personalizados, a certificação *Secure Sockets Layer* (SSL) automática e a distribuição global de conteúdo via *Content Delivery Network* (CDN). Essas funcionalidades simplificam significativamente o processo de publicação e gerenciamento do *front-end*, sendo muito adequadas ao contexto educacional.

A plataforma *Render* foi escolhida para o *deploy* do *back-end* e do banco de dados relacional. Optou-se por essa ferramenta devido à facilidade de uso, integração simples com o *GitHub Actions*, configuração rápida via interface web e plano gratuito inicial, que possibilita acesso completo aos recursos necessários por um período adequado para projetos acadêmicos. Comparativamente às alternativas como Amazon Web Services (AWS) (que apresenta limitações de uso no plano educacional), *Railway* e *Heroku*, a Render demonstrou-se a mais acessível e menos complexa para alunos em formação inicial.

O Quadro 2 apresenta uma comparação resumida das ferramentas selecionadas e suas alternativas comuns, evidenciando os principais critérios considerados na escolha.

Quadro 2 – Comparativo das ferramentas de gestão e centralização utilizadas no projeto

Categoria	Ferramenta Utilizada	Alternativas Comuns	Justificativa da Escolha
Controle de Versão	Git e GitHub	GitLab, Bitbucket	Gratuito para estudantes, amplamente utilizado e integrado diretamente às ferramentas de automação.
Integração e Entrega Contínua (CI/CD)	*GitHub Actions*	GitLab CI/CD, *Travis CI*, CircleCI	Integrado nativamente ao GitHub, facilidade de configuração e gratuidade para repositórios públicos.
deploy front-end	Vercel	*Netlify*, *Firebase Hosting*	Integração direta com GitHub, simplicidade no *deploy*, escalabilidade automática e plano gratuito robusto.
deploy Back-end e Banco de Dados	Render	Railway, Heroku, AWS Free Tier	Facilidade de configuração, integração nativa com *GitHub Actions*, suporte completo a PostgreSQL e plano gratuito inicial acessível para uso acadêmico.

Fonte: Elaborado pelo autor (2025).

A definição e integração dessas ferramentas possibilitaram a automação efetiva das etapas fundamentais do ciclo de desenvolvimento, alinhando-se diretamente com os princípios da cultura DevOps. Dessa forma, proporcionaram um ambiente acadêmico realista, permitindo aos alunos a prática ativa de versionamento, integração contínua, entrega contínua e gerenciamento

simplificado da infraestrutura em nuvem. Além disso, a escolha dessas ferramentas levou em conta critérios de acessibilidade, facilidade de uso inicial e potencial educativo, garantindo que os estudantes possam replicar e aprofundar os conhecimentos adquiridos em contextos futuros, tanto acadêmicos quanto profissionais.

4.3 Definição dos Requisitos da Aplicação

Após a definição das ferramentas de gestão e centralização do projeto, realizou-se o levantamento e documentação dos requisitos necessários para o desenvolvimento da aplicação. O sistema foi planejado especificamente para o gerenciamento das monitorias no curso de Redes de Computadores, visando atender às necessidades específicas de administradores, monitores e estudantes. Os requisitos foram divididos em funcionais e não-funcionais para proporcionar maior clareza e facilitar o desenvolvimento, garantindo assim escalabilidade, facilidade de manutenção e qualidade.

4.3.1 Requisitos Funcionais

Os requisitos funcionais definem as operações e funcionalidades essenciais que a aplicação deve oferecer aos usuários:

- **Cadastro de monitorias**: possibilitar que administradores realizem o cadastro das monitorias, incluindo informações essenciais como data, horário, localização (sala ou bloco) e disciplina;
- **Visualização das monitorias**: permitir que usuários visualizem as monitorias cadastradas, com filtros que possibilitem buscas por dia da semana, disciplina e localização;
- **Detalhamento das monitorias**: exibir informações detalhadas sobre cada monitoria, destacando dados como horário específico, nome do monitor responsável e local exato;
- **Responsividade**: assegurar que a interface da aplicação se adapte automaticamente a diferentes tipos de dispositivos (computadores, tablets e smartphones), proporcionando uma experiência de usuário uniforme e intuitiva.

4.3.2 Requisitos Técnicos

Além das funcionalidades mencionadas, foram definidos requisitos não-funcionais para garantir a segurança, desempenho e qualidade geral da aplicação:

- **Escalabilidade**: garantir que a arquitetura da aplicação permita facilmente o aumento no número de usuários simultâneos e volume de dados, sem prejudicar o desempenho;
- **Segurança**: aplicar mecanismos robustos de autenticação e autorização, protegendo as informações sensíveis dos usuários, além de implementar boas práticas recomendadas pela cultura DevOps e uso adequado de criptografia;
- **Desempenho**: otimizar consultas ao banco de dados e assegurar baixas latências durante as interações do usuário com a interface;
- **Automação**: integrar rotinas automatizadas por meio das práticas de integração contínua (CI) e entrega contínua (CD), assegurando maior confiabilidade e agilidade nas entregas.

4.3.3 Resumo das Funcionalidades e Requisitos Técnicos

O Quadro 3 sintetiza de forma clara as funcionalidades principais da aplicação, relacionando cada uma aos requisitos técnicos necessários para sua correta implementação.

Quadro 3 – Resumo das funcionalidades e requisitos técnicos da aplicação

Funcionalidade	**Requisitos técnicos associados**
Cadastro de monitorias	Segurança, escalabilidade e eficiência no processamento
Visualização de monitorias	Responsividade, baixa latência e otimização das consultas
Detalhamento de monitorias	Clareza das informações, desempenho e usabilidade
Responsividade	Adaptabilidade completa da interface aos diferentes dispositivos

Fonte: Elaborado pelo autor (2025).

4.3.4 Conclusão da Definição dos Requisitos

A definição detalhada dos requisitos funcionais e não-funcionais desempenhou um papel fundamental na orientação clara e objetiva das etapas subsequentes do desenvolvimento da aplicação. Essa organização assegurou alinhamento com os objetivos iniciais do projeto, cumprindo as necessidades práticas do gerenciamento de monitorias e adotando efetivamente os princípios essenciais da cultura DevOps. Dessa forma, foi possível estabelecer uma base sólida para a criação de um sistema robusto, escalável e seguro, plenamente preparado para a automação dos processos de integração e entrega contínua.

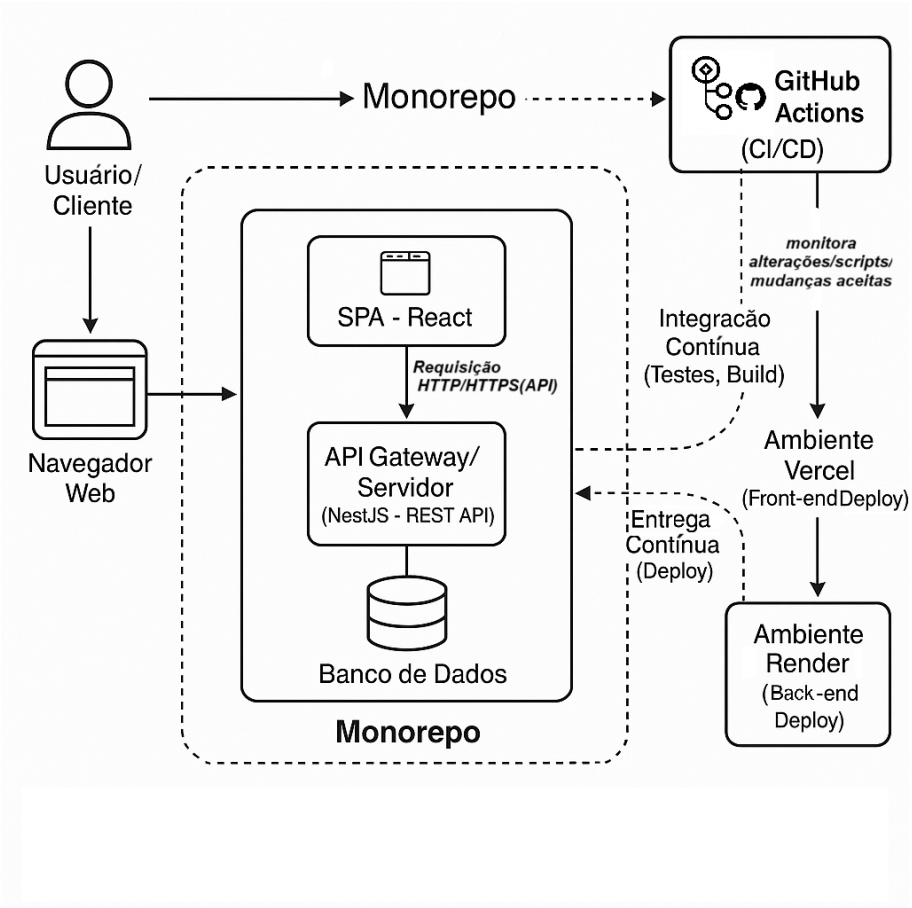
4.4 Desenvolvimento da Aplicação com Ferramentas DevOps

Esta seção detalha as etapas metodológicas seguidas para o desenvolvimento da aplicação web, com foco na aplicação prática dos conceitos e ferramentas da cultura DevOps. A metodologia adotada inclui a definição de uma arquitetura conceitual clara, o uso de ferramentas e técnicas modernas para desenvolvimento *front-end* e *back-end*, além da implementação de processos de automação (CI/CD), *feedback* contínuo e monitoramento.

4.4.1 Arquitetura da Aplicação e Fluxo de CI/CD

A Figura 9 apresenta a arquitetura conceitual da aplicação e o fluxo proposto para integração e entrega contínua (CI/CD). A arquitetura é estruturada em três componentes principais: *front-end*, *back-end* e banco de dados, integrados através de processos automatizados de testes, validação e implantação contínua com ferramentas específicas.

Figura 9 – Arquitetura conceitual da aplicação web e fluxo CI/CD adotado.



Fonte: Elaborado pelo autor (2025).

Nesse fluxo, as alterações submetidas ao repositório GitHub acionam *pipelines* automatizados utilizando o *GitHub Actions*, que realizam validações e testes nas duas camadas principais (*front-end* e *back-end*). Após aprovação, o *front-end* é automaticamente publicado na Vercel, enquanto o *back-end* e o banco de dados são implantados na plataforma Render. Este processo busca garantir entregas rápidas, consistentes e rastreáveis, fundamentais para uma cultura DevOps efetiva.

4.4.2 Implementação do front-end

Para o desenvolvimento do *front-end*, foram escolhidas tecnologias amplamente adotadas e com sólida documentação, favorecendo performance, manutenção e escalabilidade:

- *React*: framework JavaScript para construção modular e dinâmica de interfaces, permitindo uma experiência fluida ao usuário com a arquitetura SPA (*Single Page Application*).
- *Vite*: ferramenta de *build* e servidor de desenvolvimento rápido, otimizada para o desenvolvimento iterativo, reduzindo o tempo entre atualizações e testes durante o processo.
- *TailwindCSS*: framework CSS com uma abordagem utilitária que facilita a construção de interfaces responsivas e com design consistente, otimizando produtividade e manutenção do código.

As funcionalidades implementadas no *front-end* visaram proporcionar uma experiência intuitiva e acessível, incluindo visualização organizada das monitorias, interfaces responsivas adaptadas a múltiplos dispositivos e *feedback* visual imediato das operações realizadas. O *deploy* contínuo realizado na Vercel garante que cada mudança validada esteja imediatamente disponível no ambiente de produção, alinhado aos princípios DevOps de automação e integração contínua.

4.4.3 Implementação do Back-end

O desenvolvimento do *back-end* seguiu uma abordagem modular e fortemente tipada utilizando o framework *NestJS*, baseado em Node.js e TypeScript. Essa escolha técnica foi fundamentada pela facilidade de manutenção, escalabilidade e integração com outras ferramentas e tecnologias.

Para persistência de dados, adotou-se o *Prisma ORM* pela sua integração nativa com TypeScript, sistema robusto de migrações, tipagem forte e agilidade no desenvolvimento. O banco de dados escolhido foi o PostgreSQL, hospedado na plataforma Render devido à facilidade de configuração, integração com os processos automatizados e plano gratuito acessível para

projetos acadêmicos.

O arquivo `schema.prisma`, mostrado na Código-fonte 1, define a estrutura do banco de dados e configurações essenciais para o funcionamento do *Prisma ORM*. O trecho apresentado contém três partes principais:

1. `generator client`: indica ao Prisma que deve gerar automaticamente um cliente JavaScript tipado para comunicação segura e eficiente com o banco de dados.
2. `datasource db`: configura o banco de dados, neste caso `PostgreSQL`, informando a URL de conexão por meio da variável de ambiente `DATABASE_URL`. Esse mecanismo permite proteger informações sensíveis, mantendo-as fora do código fonte.
3. `model Monitoria`: define explicitamente a estrutura de uma tabela chamada `Monitoria`, com campos e seus tipos:
   - `id`: campo identificador único que utiliza números inteiros com auto incremento.
   - `titulo, professor, horario, local`: campos textuais para armazenar informações das monitorias cadastradas.

Dessa forma, o Prisma gera automaticamente comandos SQL para criação e atualização do banco, facilitando o gerenciamento dos dados e assegurando consistência e segurança.

Código-fonte 1 – Trecho do arquivo `schema.prisma`

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Monitoria {
  id         Int     @id @default(autoincrement())
  titulo     String
  professor  String
  horario    String
  local      String
}
```

4.4.4 Automação de Integração Contínua e Entrega Contínua (CI/CD)

O ciclo de automação foi implementado utilizando o G*GitHub Actions* através do arquivo de configuração `.yml`(YAML), responsável por orquestrar *workflows* para *build*, testes e *deploy* contínuo da aplicação. A integração do repositório com as plataformas Vercel (*front-end*) e Render (*back-end* e banco) permite que toda atualização aprovada seja imediatamente disponibilizada em produção, promovendo entregas ágeis e rastreáveis. Ademais, tanto a Vercel quanto a Render realizam o processo de *build* e exibem nos logs possíveis erros que impedem o *deploy*, embora testes automatizados devam ser definidos no *pipeline* de CI, como no *GitHub Actions*.

O arquivo YAML mostrado no Código-fonte 2 representa a automação do *pipeline* de *Continuous Integration/Continuous Delivery* (CI/CD) utilizado, nele é detalhado dois fluxos principais, um para o *back-end* e outro para o *front-end*, que são acionados automaticamente sempre que há um `push` ou `pull request` para a branch principal (`main`). A execução segue as etapas abaixo:

- **Job Backend**:
  1. `actions/checkout@v4`: obtém o código mais recente do repositório.
  2. `actions/setup-node@v4`: configura o ambiente Node.js com a versão especificada (neste caso, versão 18).
  3. `npm ci`: instala as dependências do projeto de maneira limpa e segura.
  4. `npx prisma generate`: gera o cliente Prisma necessário para comunicação com o banco de dados.
  5. `npm run build`: compila o código do *back-end*.
  6. `npm test`: executa testes automatizados para validar o código.
- **Job Frontend** (executado após sucesso do *back-end*):
  1. `actions/checkout@v4`: também obtém o código mais recente do repositório.
  2. `actions/setup-node@v4`: configura Node.js versão 18.
  3. `npm ci`: instala dependências do *front-end*.
  4. `npm run build`: compila a aplicação *front-end*.
  5. `npm test`: realiza testes, sendo permitido continuar em caso de falhas para revisão posterior.

Este *pipeline* assegura que cada atualização seja automaticamente testada e validada antes da implantação, garantindo estabilidade, segurança e eficiência no desenvolvimento

contínuo da aplicação.

Código-fonte 2 – Arquivo YAML conceitual para automação do *pipeline* CI/CD.

```yaml
name: CI/CD Monorepo

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  backend:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: monitoria-backend
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 18
      - run: npm ci
      - run: npx prisma generate
      - run: npm run build
      - run: npm test

  frontend:
    needs: backend
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: monitoria-frontend
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 18
      - run: npm ci
```

```
37      - run: npm run build
38      - name: Testar backend
39        run: npm test
40        continue-on-error: true
```

4.4.5 Feedback Contínuo e Monitoramento

A abordagem DevOps adotada enfatiza o feedback contínuo através de automações no *GitHub Actions*, permitindo a rápida detecção e resolução de problemas no código antes do *deploy* em produção. Esse fluxo, combinado com estratégias de monitoramento contínuo e registro de operações, garantiram maior confiabilidade e eficiência na entrega de novas versões do sistema.

4.4.6 Conclusão do Desenvolvimento com DevOps

A aplicação prática das metodologias DevOps proporcionou benefícios concretos ao projeto, tais como automação e arquitetura robusta de fácil manutenção. Essas práticas garantiram entregas frequentes, estáveis e de alta qualidade, fortalecendo a conexão entre teoria acadêmica e implementação prática. A experiência obtida demonstra claramente a viabilidade do DevOps em contextos acadêmicos, preparando futuros profissionais para demandas reais do mercado de tecnologia.

4.5 Análise Técnica do Processo DevOps

Esta seção apresenta uma avaliação técnica do processo de desenvolvimento adotado, com ênfase na automação, integração contínua, entrega contínua e monitoramento. A análise é fundamentada em métricas extraídas do ambiente de desenvolvimento e nas boas práticas associadas à cultura DevOps.

4.5.1 Métricas de Execução do Pipeline

Durante a implementação dos fluxos de integração e entrega contínua, foram observadas métricas relevantes, como o tempo médio de execução dos *jobs*, tempo de espera em fila, taxa de sucesso e falhas de execução. Essas métricas foram disponibilizadas automaticamente

pelas plataformas utilizadas (*GitHub Actions*, Vercel e Render), e permitiram acompanhar a estabilidade e a eficiência do processo de CI/CD.

A média de tempo de execução dos *pipelines* variou entre 1 e 2 minutos, considerando desde a verificação do repositório até o término dos testes automatizados e *build*. O tempo de fila foi praticamente inexistente, e a taxa de falha em etapas críticas como testes e *builds* manteve-se próxima de zero, com exceção de instabilidades pontuais durante a fase inicial de configuração do ambiente.

4.5.2 Histórico de Execução e Frequência de Atualizações

A análise do histórico de commits e execuções ao longo do tempo indicou uma frequência constante de entregas e melhorias incrementais. Essa regularidade reforça a aderência aos princípios da integração contínua, promovendo versionamento transparente, rastreabilidade das alterações e validações frequentes.

Os registros do *GitHub Actions* mostraram que, a cada `push` na `main`, os testes e *deploys* foram disparados automaticamente. Isso evidencia um ciclo automatizado de validação e publicação, favorecendo entregas rápidas e com menor risco de regressões.

4.5.3 Gestão de Falhas e Logs de Execução

Durante o desenvolvimento, alguns erros foram identificados no processo de *build*, principalmente relacionados à sintaxe incorreta em arquivos de configuração ou falhas de tipagem em funções específicas. No entanto, esses problemas foram rapidamente solucionados graças à visibilidade proporcionada pelos logs de execução e mensagens claras emitidas pelas ferramentas de integração.

A gestão contínua desses incidentes, combinada com a automação de testes e validações, permitiu manter a estabilidade do sistema ao longo do ciclo de vida da aplicação.

4.5.4 Monitoramento Básico em Produção

Após o *deploy*, as plataformas utilizadas forneceram registros automáticos de status e logs dos ambientes de produção. Essas informações possibilitaram identificar falhas, latências ou interrupções com maior agilidade, reforçando a rastreabilidade e a confiabilidade do sistema.

Ainda que não tenham sido utilizados serviços externos de observabilidade avançada,

como Prometheus ou Grafana, os recursos nativos das plataformas foram suficientes para garantir o acompanhamento básico do funcionamento da aplicação em tempo real.

4.5.5 Considerações Finais da Análise Técnica

A análise técnica do processo de desenvolvimento evidenciou a efetividade da aplicação dos princípios da cultura DevOps, especialmente nos aspectos de automação, controle de qualidade e monitoramento.

A integração entre as ferramentas utilizadas viabilizou um fluxo de entrega contínua enxuto e confiável, promovendo ciclos curtos de desenvolvimento, correção ágil de falhas e maior controle sobre os artefatos em produção. Esse conjunto de práticas resultou em um processo de desenvolvimento alinhado às exigências atuais da engenharia de software moderna.

No entanto reconhece-se a limitação relacionada à colaboração, considerando que todo o desenvolvimento da aplicação foi conduzido de forma individual, o que acabou restringido a diversidade de perspectivas no projeto nesse quesito, além de não utilizar também, métricas objetivas como o Tempo Médio para Reparo (*Mean Time To Repair* (MTTR)), frequência exata de entregas ou tempo médio de *deploy*, destacando oportunidades para trabalhos futuros.

Dando continuidade ao processo metodológico, a próxima etapa consistiu na avaliação da experiência dos usuários finais com a aplicação desenvolvida. Para isso, foi utilizado um questionário de usabilidade, conforme detalhado na seção a seguir.

4.6 Questionário de Avaliação de Usabilidade com Usuários

Como parte da metodologia, foi realizada uma avaliação de usabilidade da aplicação web, com o objetivo de validar a experiência de uso a partir da perspectiva do usuário final. Para isso, adotou-se um questionário baseado no método UEQ, traduzido como Questionário de Experiência do Usuário, conforme descrito por Schrepp e Thomaschewski (2017). O UEQ é um instrumento reconhecido internacionalmente, amplamente utilizado em pesquisas acadêmicas para mensuração da experiência do usuário em produtos digitais, permitindo avaliar atributos como atratividade, perspicuidade, eficiência, dependabilidade, estimulação e originalidade.

O procedimento adotado seguiu os seguintes passos:

1. **Seleção dos Participantes:** A aplicação foi disponibilizada a alunos do curso de Redes de Computadores da UFC, escolhidos por conveniência e proximidade com o contexto de uso

do sistema.

2. **Execução dos Testes:** Os participantes foram convidados a realizar testes livres nas funcionalidades principais da aplicação, tais como cadastro, edição, remoção e busca de monitorias.
3. **Aplicação do Questionário:** Após a interação com o sistema, cada usuário respondeu a um questionário estruturado contendo assertivas relacionadas aos fatores do UEQ, avaliadas em escala Likert de 1 (discordo totalmente) a 5 (concordo totalmente).
4. **Coleta de Dados:** As respostas foram coletadas de forma anônima, garantindo o consentimento e a privacidade dos participantes, conforme o termo ilustrado na Figura 18.
5. **Análise das Respostas:** Os dados coletados foram organizados para análise quantitativa (frequência, médias e percentuais) e qualitativa (comentários e sugestões dos usuários).

O questionário aplicado avaliou os seguintes fatores de experiência do usuário:

- Atratividade (satisfação geral e prazer de uso)
- Perspicuidade (facilidade de entendimento e aprendizado)
- Eficiência (rapidez e facilidade na realização das tarefas)
- Dependabilidade (previsibilidade e segurança das operações)
- Estimulação (interesse e motivação do uso)
- Originalidade (inovação em relação às alternativas tradicionais)

O instrumento completo encontra-se disponível no Apêndice A. Os resultados desta avaliação, incluindo gráficos e análises das respostas dos participantes, serão apresentados no capítulo seguinte deste trabalho.

5 RESULTADOS

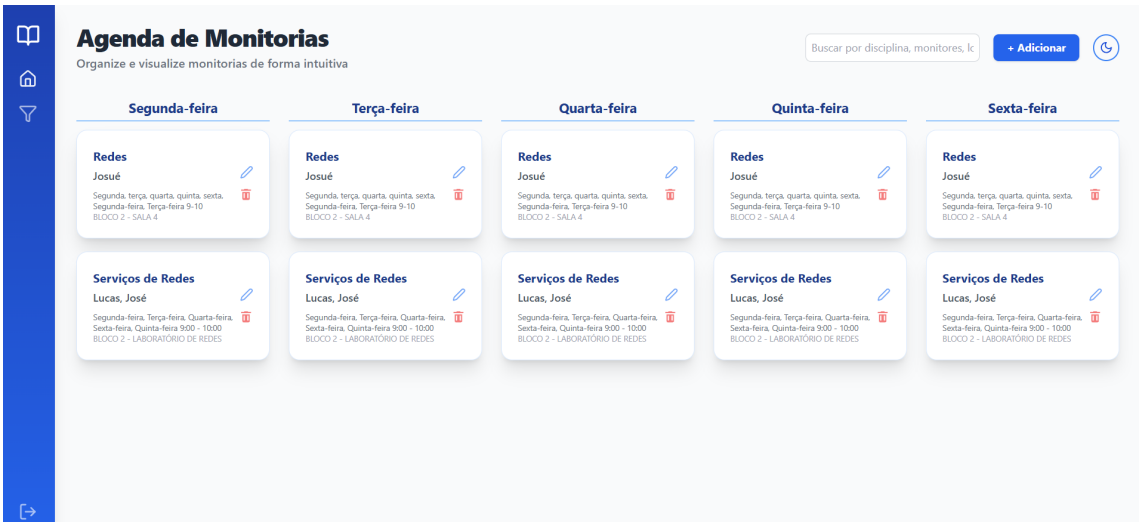
Neste capítulo são apresentados e discutidos os principais resultados obtidos durante o desenvolvimento e validação da aplicação de gerenciamento de monitorias baseada em práticas com ferramentas DevOps. Os resultados incluem evidências visuais da aplicação, métricas dos *pipelines* automatizados, registros das execuções, além da análise da avaliação de usabilidade realizada com usuários reais. Ao final, é apresentada uma discussão crítica relacionando os resultados aos objetivos, à literatura e às limitações encontradas.

5.1 Evidências do Funcionamento da Aplicação Web

A implementação da aplicação web contemplou a interface intuitiva, responsiva e com recursos visuais de fácil navegação. As imagens a seguir apresentam as principais telas do sistema e suas funcionalidades.

O painel inicial exibe todas as monitorias cadastradas, organizadas por dia da semana e acessível a diferentes perfis de usuário como pode ser visualizado na Figura 10. O design privilegia a clareza das informações e uma experiência positiva tanto em modo claro quanto escuro.

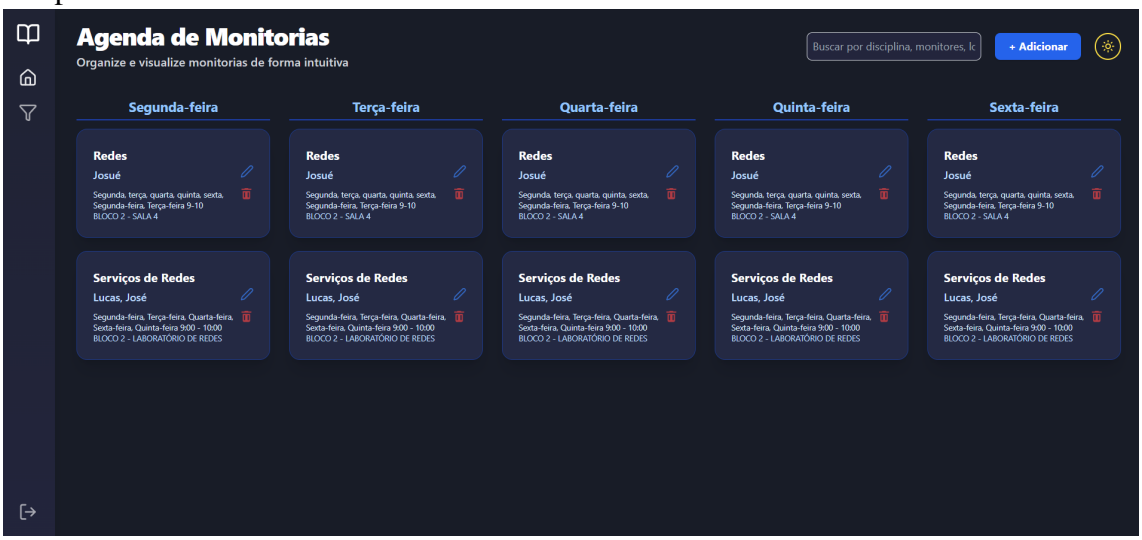
Figura 10 – Painel inicial da aplicação web (modo claro), apresentando as monitorias cadastradas.



Fonte: Elaborado pelo autor (2025).

A interface também oferece suporte a tema escuro como mostrado na Figura 11, sendo importante para acessibilidade e conforto visual do usuário durante longos períodos de uso.

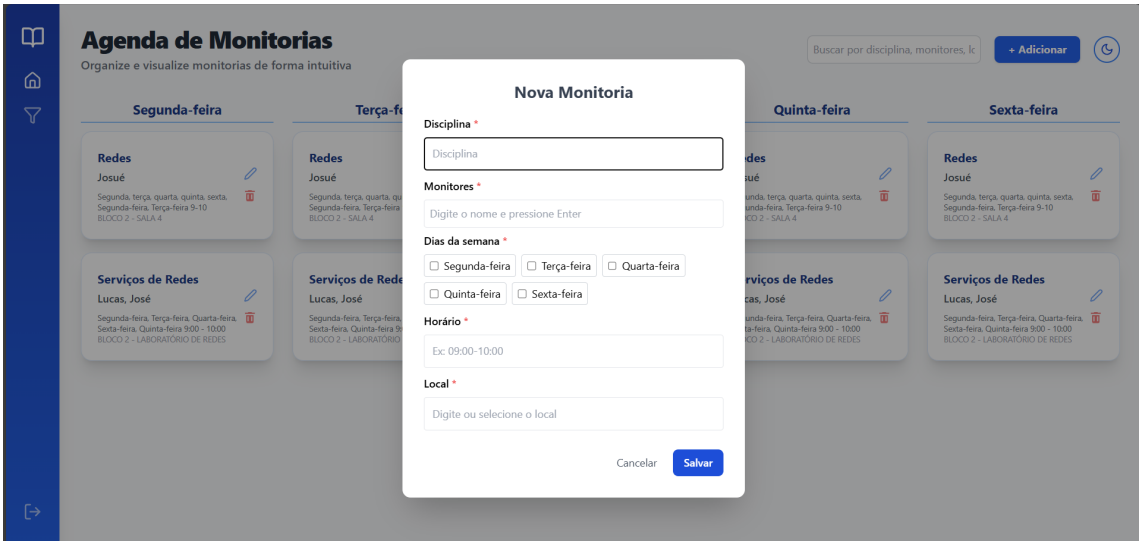
Figura 11 – Painel inicial da aplicação web (modo escuro), demonstrando acessibilidade e adaptabilidade visual.



Fonte: Elaborado pelo autor (2025).

Na Figura 12 pode-se visualizar o cadastro de uma nova monitoria, que é realizado por meio de um modal específico, que exige o preenchimento de campos obrigatórios e validação instantânea. Essa abordagem reduz a possibilidade de erros e torna o processo mais eficiente.

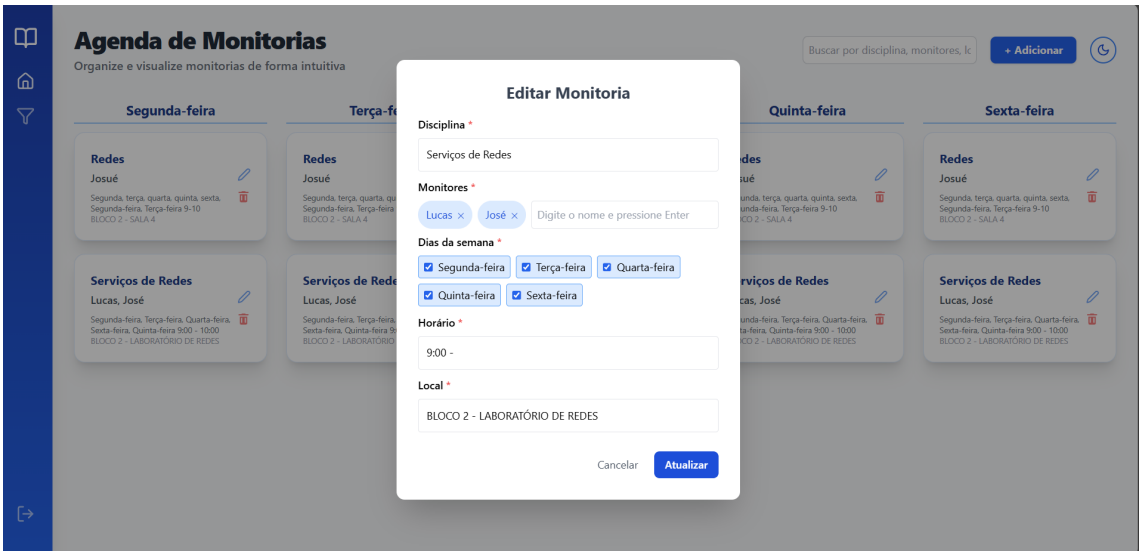
Figura 12 – Modal de cadastro de nova monitoria, com validação dos campos obrigatórios.



Fonte: Elaborado pelo autor (2025).

A edição de monitorias pode ser feita com facilidade, permitindo que o usuário altere informações relevantes a qualquer momento, garantindo flexibilidade à gestão do sistema. Na Figura 13 pode ser visualizado o modal de edição.

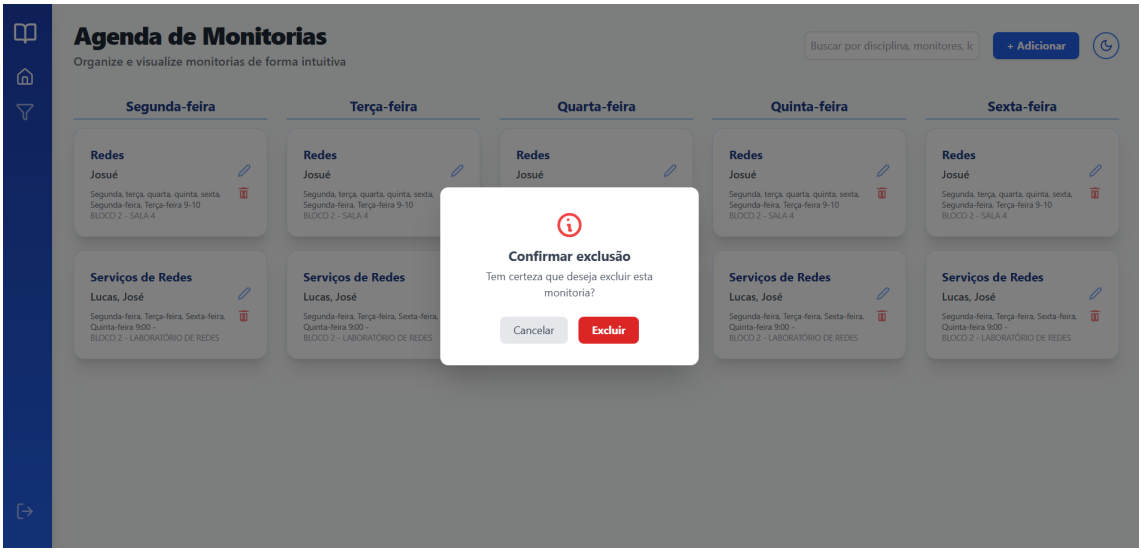
Figura 13 – Modal de edição de monitoria, com seleção dinâmica de campos e atualização direta.



Fonte: Elaborado pelo autor (2025).

Na Figura 14 é mostrado a remoção de monitorias, que exige confirmação antes de realmente deletar, reforçando práticas de segurança e evitando exclusão acidental de dados importantes.

Figura 14 – Modal de confirmação para remoção de monitoria, reforçando boas práticas de usabilidade.



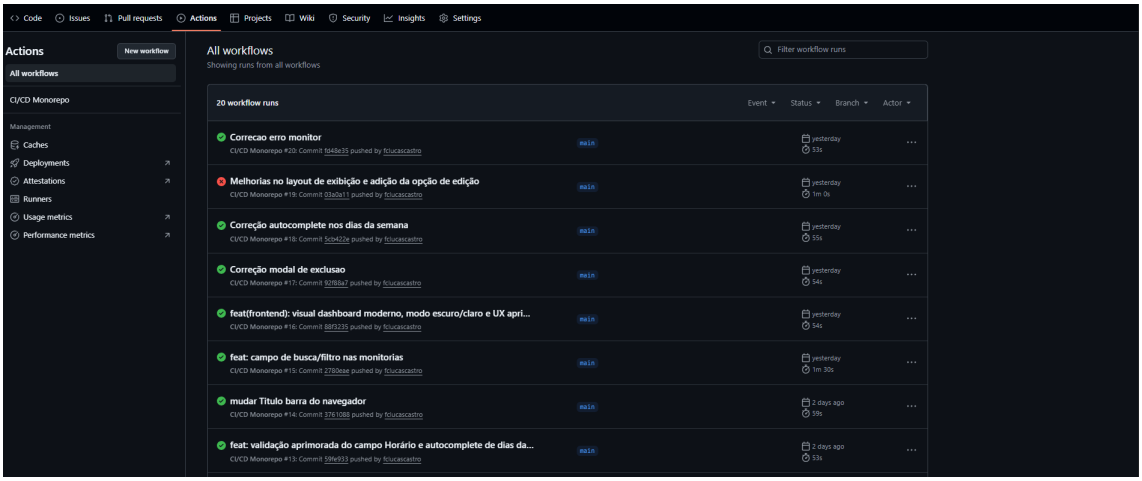
Fonte: Elaborado pelo autor (2025).

5.2 Teste de Integração das Ferramentas

A validação da integração entre as ferramentas utilizadas no *pipeline* DevOps foi realizada por meio de um ciclo completo de automação. O objetivo foi assegurar que cada etapa, desde a submissão do código até a publicação do sistema, ocorresse de maneira coordenada e rastreável.

O teste se inicia a partir de um novo *commit* ou *merge* na *branch* principal do repositório *GitHub*. Esse evento dispara automaticamente o *workflow* configurado no *GitHub Actions*, responsável por executar tarefas de *build*, teste e *deploy*. Esse fluxo automatizado pode ser visualizado no painel de execuções do *GitHub Actions*, conforme mostra a Figura 15, que ilustra o status das etapas do *pipeline*, com registros claros de execuções bem-sucedidas e eventuais falhas.

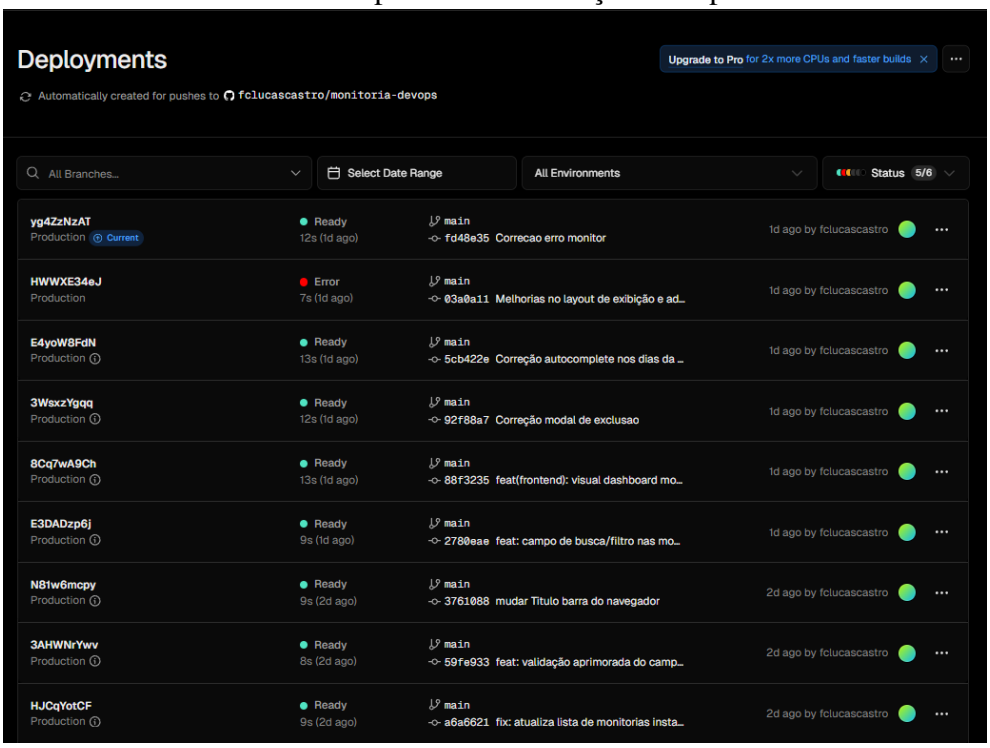
Figura 15 – Execução do *pipeline* CI/CD no *GitHub Actions*, mostrando status das etapas de build e testes.



Fonte: Elaborado pelo autor (2025).

Após a conclusão dos testes e do build, o processo de *deploy* contínuo é iniciado. Para o front-end, a publicação ocorre automaticamente na plataforma *Vercel*. O painel da Vercel registra o histórico de builds, como ilustrado na Figura 16, onde é possível acompanhar o sucesso de cada publicação e eventuais mensagens de erro.

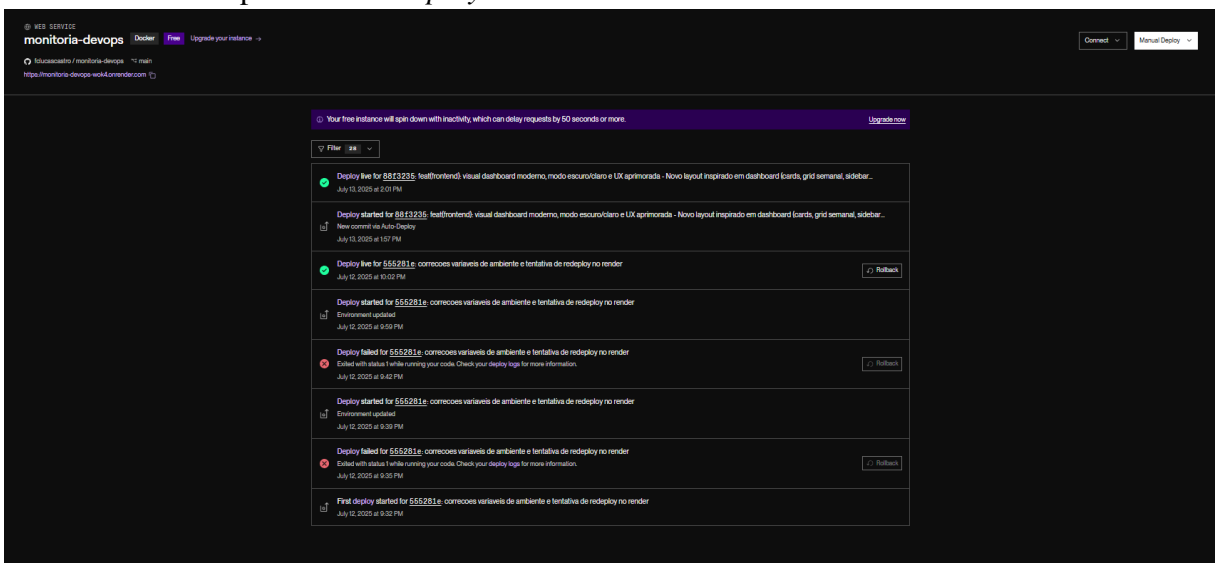
Figura 16 – Log do processo de build e publicação do *front-end* na Vercel, indicando sucesso ou falha após cada atualização do repositório.



Fonte: Elaborado pelo autor (2025).

De maneira semelhante, o *deploy* do *back-end* e banco de dados é realizado pela plataforma *Render*. A Figura 17 apresenta o painel de *builds* do Render, que mostra o histórico das publicações e possibilita a verificação detalhada de cada execução.

Figura 17 – Histórico de *builds* do *back-end* e banco de dados na Render, evidenciando a rastreabilidade do processo de *deploy*.



Fonte: Elaborado pelo autor (2025).

Por fim, a verificação final da integração é realizada acessando a aplicação online e

conferindo se as alterações feitas no código estão refletidas no ambiente de produção, garantindo que o ciclo completo de automação e entrega contínua foi efetivamente realizado.

O Quadro 4 sintetiza as principais etapas desse teste de integração entre ferramentas e as evidências observadas ao longo do processo.

Quadro 4 – Resumo dos testes de integração entre ferramentas no pipeline DevOps

Etapa Testada	**Evidência Observada**
Commit ou merge no *GitHub*	*workflow* do *GitHub Actions* iniciado automaticamente, registro dos eventos nos logs da plataforma
Execução dos jobs (build/testes)	Status e histórico do *pipeline* registrados no painel do *GitHub Actions*
deploy no *Vercel* (*front-end*) e *Render* (back-end)	Novos *builds* gerados nas plataformas, exibição dos logs e atualização do ambiente online
Verificação final da aplicação	Acesso à aplicação publicada na web, conferindo se as mudanças realizadas no código estavam refletidas conforme o esperado

Fonte: Elaborado pelo autor (2025).

Esse processo demonstrou, de forma prática e transparente, a eficácia da integração entre as ferramentas do *pipeline* DevOps adotado. Sempre que uma alteração era realizada no código, o ciclo de automação respondia de forma coordenada e rastreável, proporcionando maior segurança e agilidade às entregas, além de facilitar a identificação de eventuais falhas em qualquer etapa do fluxo.

5.3 Análise dos Resultados do Questionário de Usabilidade

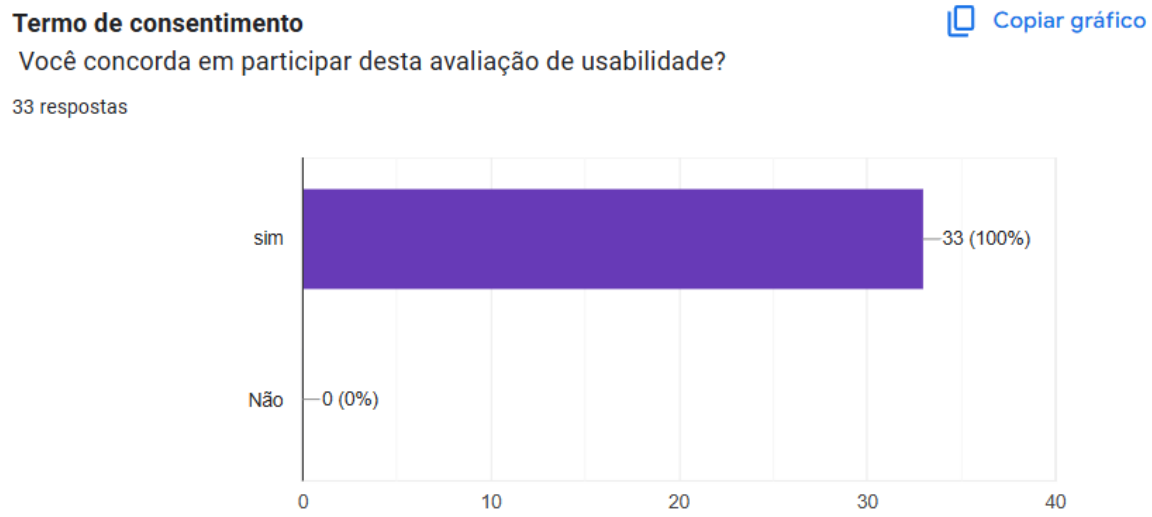
Conforme detalhado na Metodologia deste trabalho, a avaliação da experiência de uso da aplicação foi conduzida por meio do *User Experience Questionnaire* (UEQ), um instrumento amplamente reconhecido para mensuração da usabilidade em sistemas interativos (Schrepp; Thomaschewski, 2017). O questionário foi baseado nos seis fatores principais definidos pelo método: Atratividade, Perspicuidade, Eficiência, Dependabilidade, Estimulação e Originalidade. Cada um desses fatores busca capturar aspectos fundamentais da experiência do usuário, permitindo uma análise quantitativa e qualitativa do sistema desenvolvido.

Com o numero de 33 respostas dos alunos, foi feita uma análise sistemática da experiência de uso da aplicação. Os resultados a seguir são apresentados de forma estruturada para cada fator avaliado, acompanhados de discussões que relacionam as percepções dos usuários ao contexto do trabalho, conforme descrito na Metodologia. O objetivo é oferecer uma visão

sistemática dos pontos fortes, limitações e oportunidades de melhoria identificadas durante a pesquisa.

Antes do início da pesquisa, todos os participantes consentiram em contribuir concordando com o termo de consentimento, garante a participação voluntária e consciente dos usuários no processo de avaliação, conforme ilustrado na Figura 18.

Figura 18 – Termo de consentimento dos participantes da avaliação de usabilidade.

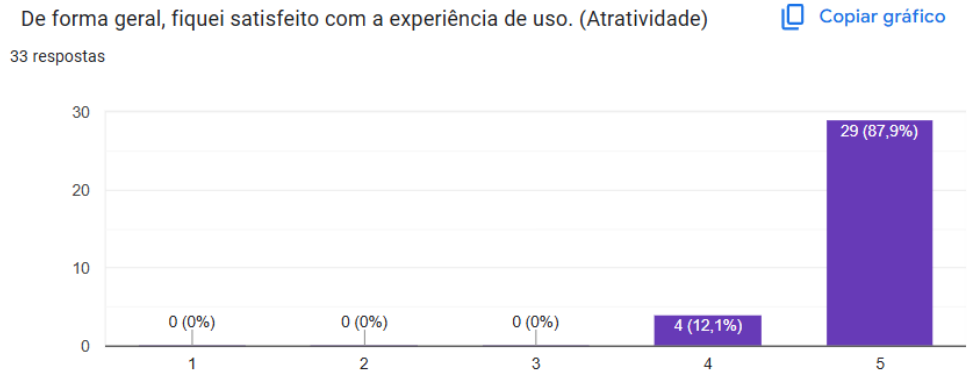


Fonte: Elaborado pelo autor (2025).

5.3.1 Atratividade

A Figura 19 mostra que satisfação geral dos usuários com a aplicação foi alta: 87,9% dos participantes atribuíram nota máxima à experiência geral. Isso evidencia que o sistema atendeu as expectativas de um produto agradável e amigável ao usuário.

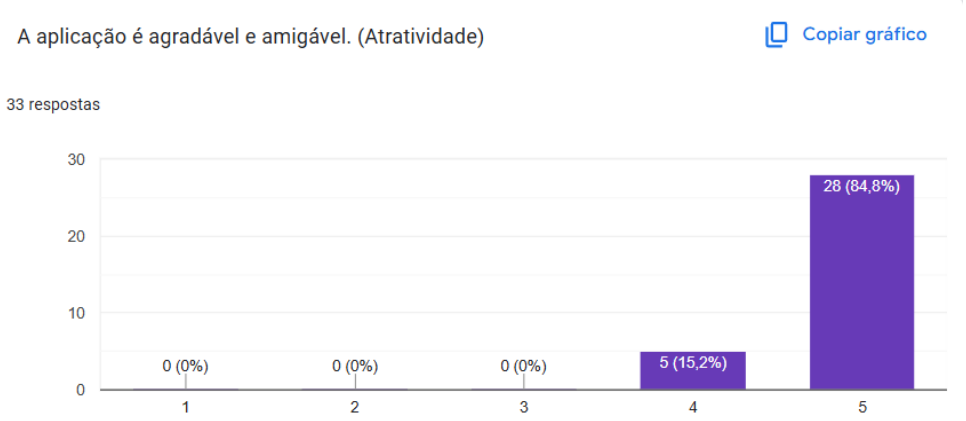
Figura 19 – Satisfação geral dos usuários em relação à experiência de uso (Atratividade).



Fonte: Elaborado pelo autor (2025).

A percepção positiva também se reflete na avaliação sobre o quanto a aplicação é considerada agradável e simpática para uso no dia a dia acadêmico, tal percepção é visualizada ao analizar a Figura 20.

Figura 20 – Avaliação da aplicação quanto à agradabilidade e simpatia (Atratividade).

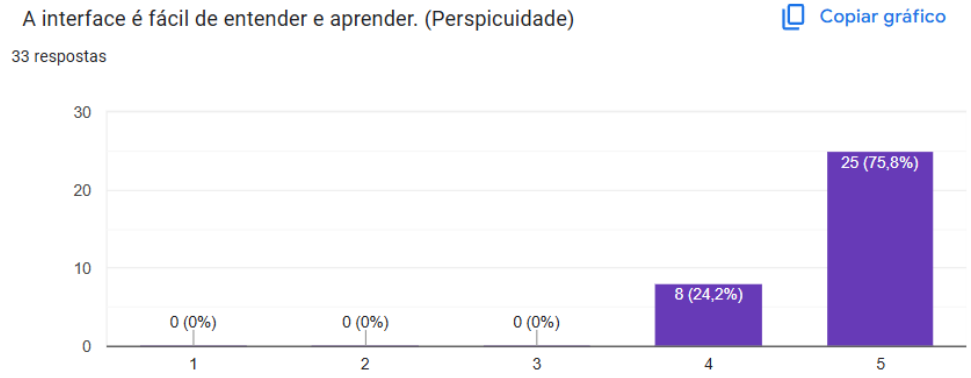


Fonte: Elaborado pelo autor (2025).

5.3.2 Perspicuidade

A facilidade de entendimento e aprendizado também foi bem avaliada: 75,8% dos usuários deram nota máxima para a clareza da interface, como pode ser visualizado na Figura 21, O que indica uma assimilação rápida mesmo que por novos usuários.

Figura 21 – Avaliação da facilidade de entendimento e aprendizado (Perspicuidade).

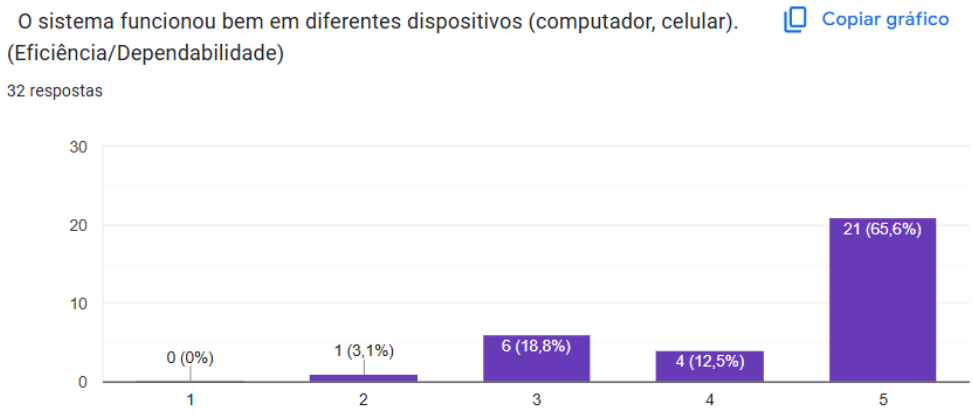


Fonte: Elaborado pelo autor (2025).

5.3.3 Eficiência

A eficiência do sistema foi avaliada tanto pelo funcionamento em diferentes dispositivos, como pelo desempenho nas tarefas principais. Figura 22 apresentada a seguir indica boa aceitação quanto à performance no uso em computadores e dispositivos móveis, apesar de alguns relatos de dificuldade pontual para cadastro em smartphones (necessidade de pressionar “Enter” para concluir o cadastro de monitorias).

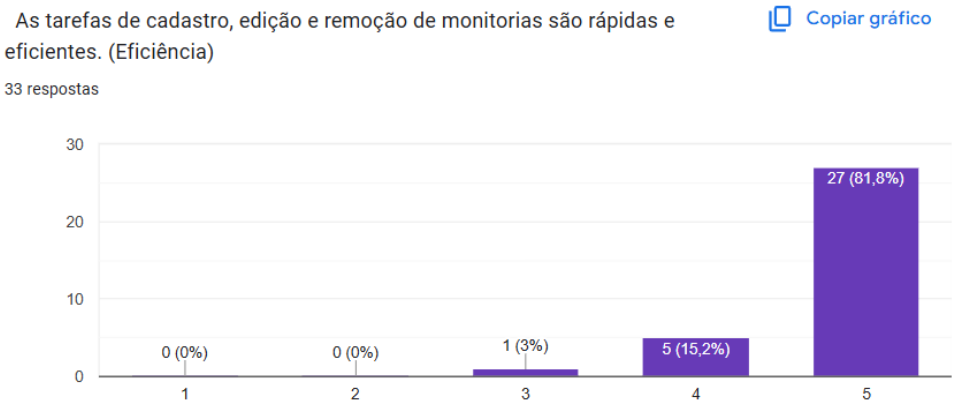
Figura 22 – Funcionamento do sistema em dispositivos diversos (Eficiência/-Dependabilidade).



Fonte: Elaborado pelo autor (2025).

A rapidez e eficiência no cadastro, edição e remoção de monitorias são aspectos valorizados pelos usuários, como pode ser visto na Figura 23 a seguir.

Figura 23 – Avaliação da rapidez e eficiência nas principais tarefas do sistema (Eficiência).

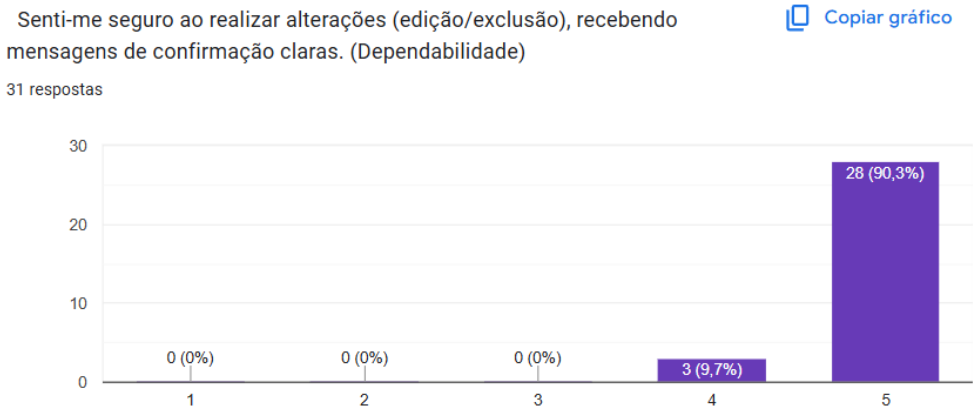


Fonte: Elaborado pelo autor (2025).

5.3.4 Dependabilidade

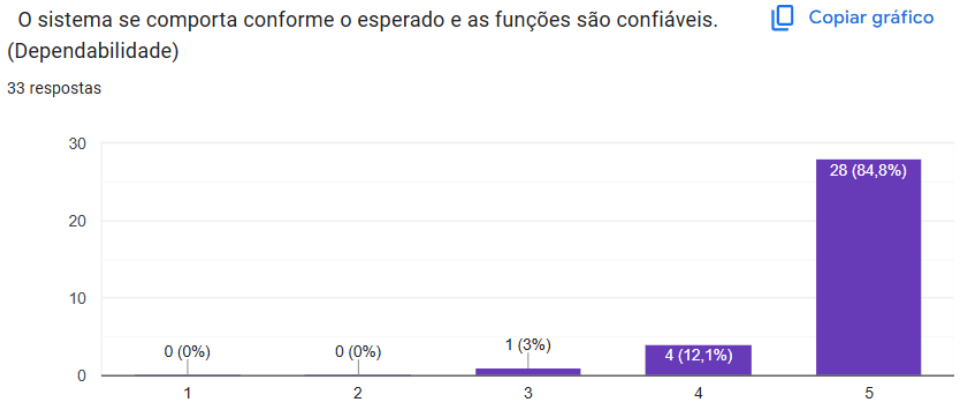
No quesito dependabilidade, a sensação de segurança e confiança foi evidente, como podemos visualizar na Figura 24 que 90,3% dos respondentes sentiram-se seguros ao realizar alterações, enquanto 84,8% dispoto na Figura 25, avaliaram como confiável o comportamento do sistema nas funções oferecidas.

Figura 24 – Sensação de segurança ao realizar operações no sistema (Dependabilidade).



Fonte: Elaborado pelo autor (2025).

Figura 25 – Percepção de confiabilidade e previsibilidade das funções (Dependabilidade).

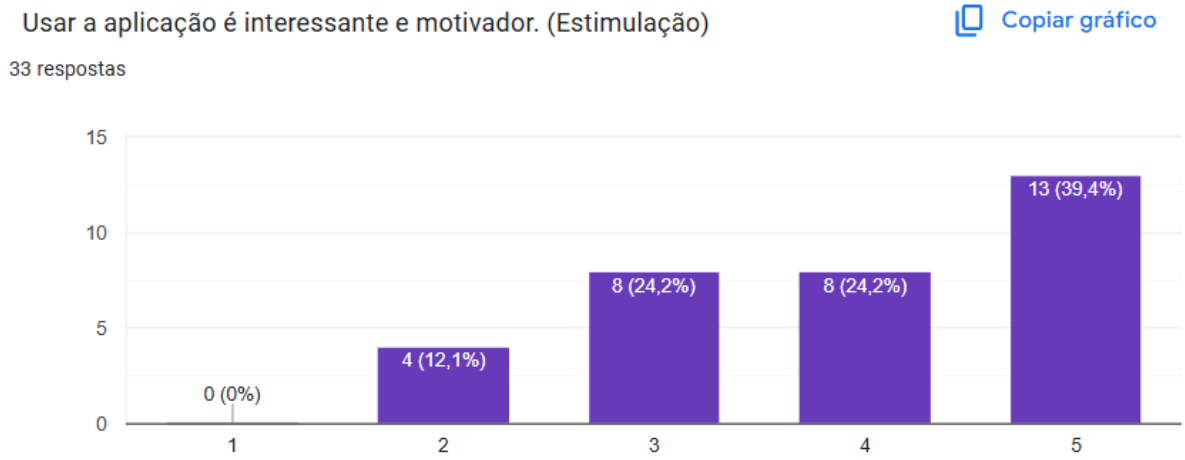


Fonte: Elaborado pelo autor (2025).

5.3.5 Estimulação

A Figura 26 mostra o interesse e motivação gerados pela aplicação, 39,4% dos participantes deram nota máxima, e com boa distribuição nas notas intermediárias.

Figura 26 – Interesse e motivação ao utilizar a aplicação (Estimulação).

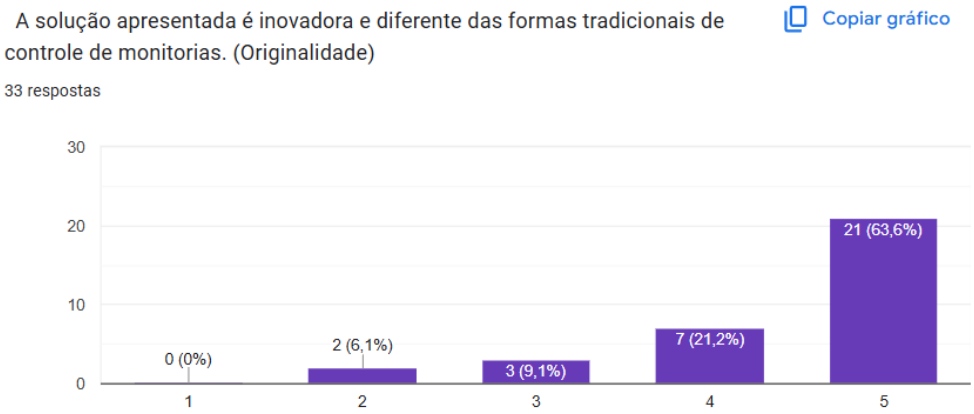


Fonte: Elaborado pelo autor (2025).

5.3.6 Originalidade

Com relação a inovação do sistema, na Figura 27 é mostrado que comparada a métodos tradicionais 63,6% dos usuários atribuiram nota máxima, e apesar de não ser um unânimidade é possivel perceber a originalidade da solução.

Figura 27 – Avaliação da inovação em relação a métodos tradicionais (Originalidade).

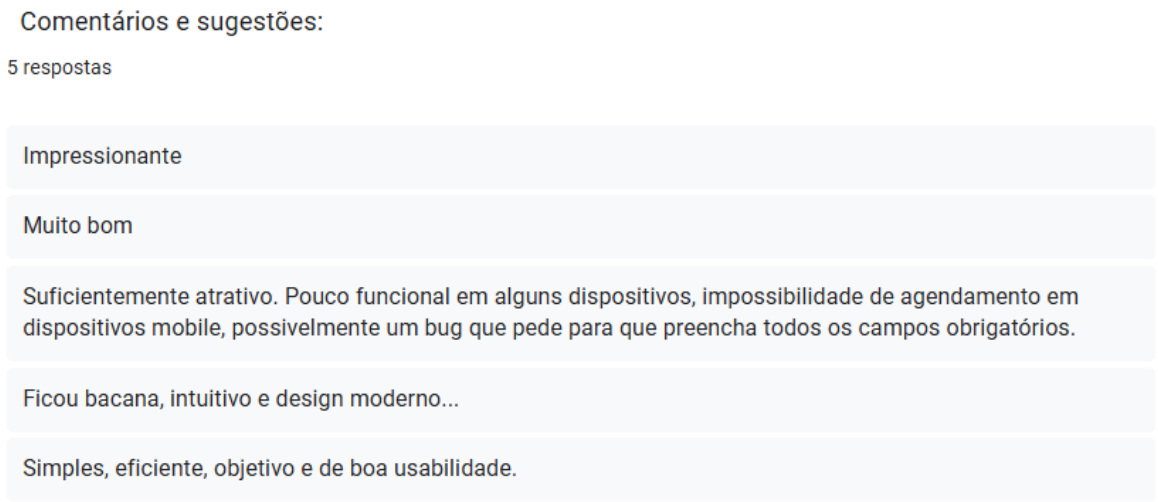


Fonte: Elaborado pelo autor (2025).

5.3.7 Comentários e Sugestões dos Usuários

Os comentários dos participantes destacaram pontos como simplicidade, objetividade, eficiência e design moderno, além de sugerirem melhorias para tornar a experiência em dispositivos móveis ainda mais acessível, tais comentários podem ser visualizados na Figura 28.

Figura 28 – Comentários e sugestões registrados pelos participantes.



Fonte: Elaborado pelo autor (2025).

5.4 Discussão Crítica dos Resultados

A análise dos resultados evidencia que a aplicação atendeu de forma satisfatória aos objetivos propostos, tanto sob o ponto de vista técnico quanto da experiência do usuário. A elevada aceitação e facilidade de uso observadas nas respostas ao questionário de usabilidade reforçam o alinhamento do sistema às necessidades reais do público-alvo.

Entre os pontos fortes, destaca-se o impacto positivo da cultura DevOps na agilidade e confiabilidade do ciclo de entrega de *software*. A automação das etapas de integração, testes e *deploy* não só reduziu falhas, como também acelerou o tempo de resposta a eventuais correções e melhorias — resultado observado tanto nos registros dos *pipelines* quanto na percepção dos usuários finais.

No entanto, o desenvolvimento enfrentou desafios práticos, especialmente em relação à infraestrutura. Algumas plataformas amplamente reconhecidas, como a AWS, mostraram-se restritivas para uso acadêmico devido a limitações do plano gratuito, exigindo adaptações e migração para alternativas mais acessíveis como o Render, que ofereceu maior flexibilidade sem custos iniciais e configuração simplificada. Essas limitações, apesar de terem sido contornadas, ilustram um entrave comum em projetos universitários que buscam aplicar práticas de DevOps utilizando recursos gratuitos.

Na literatura recente, ainda se observa relativa escassez de trabalhos que abordem de modo sistemático a integração entre práticas de DevOps e contextos de ensino superior, principalmente com evidências práticas em projetos reais. Entretanto, esse cenário vem mudando à medida que universidades reconhecem o valor da cultura DevOps e começam a inseri-la em disciplinas e projetos, promovendo iniciativas semelhantes à deste estudo.

Como oportunidades de melhoria, destaca-se a possibilidade de ampliar o conjunto de métricas analisadas nos *pipeline* (por exemplo, tempos médios de *deploy*, cobertura de testes e logs de uso da aplicação), além do aprimoramento contínuo da usabilidade, especialmente em dispositivos móveis, com base nos comentários recebidos dos usuários.

Em síntese, o trabalho contribui para o avanço da cultura DevOps no ambiente acadêmico, servindo como referência prática e incentivando futuras investigações sobre a aplicação desses princípios em projetos universitários.

6 CONCLUSÃO

Este trabalho demonstrou o desenvolvimento de uma aplicação web acadêmica utilizando ferramentas técnicas do ciclo DevOps, destacando práticas de automação, integração contínua e entrega contínua, e a avaliação técnica e A proposta envolveu desde a definição das ferramentas de gestão e automação, passando pela implementação do sistema e da infraestrutura automatizada, até a avaliação de usabilidade junto a usuários reais.

Os resultados obtidos evidenciaram que a integração de ferramentas como*GitHub*, *GitHub Actions*, Vercel e Render proporcionou um fluxo de desenvolvimento ágil, automatizado e alinhado às melhores práticas de entrega contínua e monitoramento. A aplicação atendeu aos principais requisitos técnicos e funcionais definidos, apresentando altos índices de satisfação entre os usuários, especialmente nos quesitos de atratividade, perspicuidade e eficiência, conforme apontado pelo método UEQ.

Apesar do sucesso na implementação e avaliação, algumas limitações foram identificadas ao longo do projeto. O presente trabalho focou predominantemente no desenvolvimento técnico com ferramentas DevOps, não tendo realizado uma avaliação sistemática e objetiva da colaboração, comunicação ou cultura organizacional, devido a limitações temporais e estruturais. Para trabalhos futuros, sugere-se aplicar *frameworks* de maturidade como *CALMS* ou questionários padronizados para medir colaboração, comunicação e feedback contínuo entre equipes. Além disso, ferramentas avançadas como Prometheus e Grafana poderiam aprimorar significativamente o monitoramento contínuo e gerar métricas objetivas, tais como frequência de entrega, tempo médio de recuperação (MTTR) e taxa de falha. Outras Limitações observadas foram as restrições impostas por ferramentas gratuitas de infraestrutura em nuvem — como as limitações do plano gratuito da AWS, que inviabilizaram a manutenção de serviços essenciais como o AWS Lambda, forçando a migração para soluções alternativas como o Render.

Como oportunidades de aprimoramento, recomenda-se principalmente a analise do processo devops com uma equipe onde se possa ser analisado o topíco colaboração, além da expansão da amostra de usuários nos testes de usabilidade, a inclusão de novos módulos e integrações à aplicação, e o aprofundamento da automação dos fluxos de entrega e monitoramento. Futuramente, a replicação deste estudo em outros cursos ou contextos acadêmicos pode contribuir para a consolidação da cultura DevOps no ambiente universitário, servindo de referência para projetos de ensino, pesquisa e extensão.

Em síntese, este trabalho reforça a importância e a aplicabilidade do DevOps como

abordagem metodológica no desenvolvimento de soluções acadêmicas, promovendo não apenas ganhos técnicos e operacionais, mas também o amadurecimento de práticas colaborativas e inovadoras no ensino superior em TI. Espera-se que a experiência aqui documentada sirva de incentivo para novas iniciativas e inspire a adoção de práticas DevOps em outros projetos universitários, bem como inspirar novas iniciativas de pesquisa e inovação em TI.

REFERÊNCIAS

AHMAD, M. O.; MARKKULA, J.; OIVO, M. Kanban in software development: A systematic literature review. In: EUROMICRO ASSOCIATION, 2013, Santander, Espanha. **39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)**. [*S. l.*], 2013. p. 9–16.

ALARCON, R.; VALLE, B.; ROS, J. Método de gestión de requisitos para promover la sostenibilidad en devops: Crets4devops. In: SOCIEDADE BRASILEIRA DE COMPUTAçãO, 27., 2024, Curitiba/PR. **Anais do XXVII Congresso Ibero-Americano em Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2024. p. 340–347. Disponível em: https://sol.sbc.org.br/index.php/cibse/article/view/28459. Acesso em: 5 ago. 2024.

BECK, K.; BEEDLE, M.; BENNEKUM, A. van; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R. C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D. **Manifesto for Agile Software Development**. 2001. Disponível em: https://agilemanifesto.org/. Acesso em: 30 jun. 2025.

BEPPLER, B. A. F. Da teoria à prática: Desafios da implantação da cultura devops na rotina de empresa de desenvolvimento de sistemas. **Cognitionis Scientific Journal**, v. 6, n. 2, p. 644–654, 2023. Disponível em: https://revista.cognitioniss.org/index.php/cogn/article/view/294/246. Acesso em: 30 ago. 2024.

BRANDAO, R. **Métodos ágeis na gestão de projetos**. 2023. Disponível em: https://pt.linkedin.com/pulse/m%C3%A9todos-%C3%A1geis-na-gest%C3%A3o-de-projetos-rafael-brand%C3%A3o. Acesso em: 29 ago. 2024.

CORREA, R. O. **Os desafios na adoção de DevOps em empresa brasileira de tecnologia da informação no setor bancário**. 102 p. Dissertação (Especialização em Gestão de Tecnologia da Informação) – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, São Paulo, SP, Brasil, 2017. Disponível em: https://spo.ifsp.edu.br/images/phocadownload/DOCUMENTOS_MENU_LATERAL_FIXO/POS_GRADUA%C3%A7%C3%83O/ESPECIALIZA%C3%87%C3%83O/Gest%C3%A3o_da_Tecnologia_da_Informa%C3%A7%C3%A3o_____/PRODUCAO/2017/Os_Desafios_na_Ado%C3%A7%C3%A3o_de_DevOps_em_Empresa_Brasileira_de_Tecnologia_da_Informa%C3%A7%C3%A3o_no_Setor_Banc%C3%A1rio.pdf. Acesso em: 30 ago. 2024.

DOERNENBURG, E. The path to devops. **IEEE Software**, PP, p. 1–1, 2018.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) – University of California, Irvine, 2000.

FOWLER, M. **Patterns of Enterprise Application Architecture**. [*S. l.*]: Addison-Wesley Professional, 2002. ISBN 978-0321127426.

GALVO, B. **Visão Geral do Scrum**. 2023. Disponível em: https://brunogalvo-83373.medium.com/vis%C3%A3o-geral-do-scrum-db16182e96d4. Acesso em: 30 ago. 2024.

GIMENEZ, P. J.; SANTOS, G. Diagnóstico de maturidade devops – um estudo de caso em duas organizações públicas. In: SOCIEDADE BRASILEIRA DE COMPUTAçãO, 16., 2020, Nova York, NY, EUA. **Anais do XVI Simpósio Brasileiro de Sistemas de Informação**. Nova York, NY, EUA: SBSI, 2020. p. 1–8.

GUERRERO, J.; DíAZ, S. C.; ZúñIGA, K.; PARDO, C. Tendencias en devops: un mapeo sistemático de la literatura. **RISTI - Revista Iberica de Sistemas e Tecnologias de Informacao**, v. 1, p. 291–305, ago. 2020.

HASSELBRING, W.; HENNING, S.; LATTE, B.; MOBIUS, A.; RICHTER, T.; SCHALK, S.; WOJCIESZAK, M. Industrial devops. In: IEEE, 2019, Hamburgo, Alemanha. **2019 IEEE International Conference on Software Architecture Companion (ICSA-C)**. [*S. l.*], 2019. p. 123–126.

HAVERBEKE, M. **Eloquent JavaScript, 3rd Edition**: A modern introduction to programming. 3rd. ed. [*S. l.*]: No Starch Press, 2018. ISBN 978-1593279509.

HUMBLE, J.; FARLEY, D. **Continuous Delivery**: Reliable software releases through build, test, and deployment automation. [*S. l.*]: Pearson Education, 2010.

KIM, G.; BEHR, K.; SPAFFORD, G. The phoenix project: A novel about it, devops, and helping your business win. **IT Revolution Press**, 2021.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. **The DevOps Handbook**: How to create world-class agility, reliability, & security in technology organizations. [*S. l.*]: IT Revolution Press, 2016.

LEITE, L.; ROCHA, A.; KON, F.; MILOJICIC DEJAN E MEIRELLES, P. A survey of devops concepts and challenges. **ACM Computing Surveys**, ACM, v. 52, n. 6, p. 1–35, 2020.

MORRIS, K. **Infrastructure as Code**: Dynamic systems for the cloud age. 2. ed. O'Reilly Media, 2020. Disponível em: https://www.oreilly.com/library/view/infrastructure-as-code/9781098114664/. Acesso em: 3 ago. 2025.

PAEZ, N.; FONTELA, C. Software engineering education in the devops era. In: SOCIEDADE BRASILEIRA DE COMPUTAçãO, 26., 2023, Montevideo, Uruguai. **Anais do XXVI Congresso Ibero-Americano em Engenharia de Software**. Porto Alegre, RS, Brasil: SBC, 2023. p. 130–137. Disponível em: https://sol.sbc.org.br/index.php/cibse/article/view/24698. Acesso em: 5 ago. 2024.

PAVAN, D.; SOUZA Érica de; OLIVEIRA, R. Detectando e propondo melhorias em cenário devops de uma empresa de desenvolvimento de software. In: SOCIEDADE BRASILEIRA DE COMPUTAçãO, 2018, Dois Vizinhos/PR. **Anais da Escola Regional de Engenharia de Software (ERES)**. Dois Vizinhos/PR: Sociedade Brasileira de Computação, 2018. p. 129–136.

POTVIN, R.; LEVENBERG, J. Why google stores billions of lines of code in a single repository. **Communications of the ACM**, v. 59, n. 7, p. 78–87, 2016.

PRESSMAN, R. **Engenharia de Software**: Uma abordagem profissional. 7a. ed. São Paulo: McGraw-Hill, 2011.

RANI, V. B.; KARTHIKA, P. B. Infrastructure as code (iac) and its role in achieving devops goals. **International Journal of Science and Research (IJSR)**, v. 12, n. 1, p. 332–337, 2024. Disponível em: https://www.ijsr.net/archive/v12i1/SR24304170702.pdf. Acesso em: 7 jul. 2025.

SCHREPP, M.; THOMASCHEWSKI, A. H. e J. Measuring user experience with the user experience questionnaire (ueq). **International Journal of Interactive Multimedia and Artificial Intelligence**, v. 4, n. 4, p. 39–45, 2017. Disponível em: https://www.ijimai.org/journal/sites/default/files/files/2017/04/ijimai_4_4_6_pdf_23432.pdf. Acesso em: 13 jul. 2025.

SCHWABER, K.; BEEDLE, M. **Agile Software Development with Scrum**. 1st. ed. USA: Prentice Hall PTR, 2001. ISBN 0130676349.

SHAHIN, M.; BABAR, M. A.; ZHU, L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. **IEEE Access**, PP, p. 2, 2017.

SILVA, T. M. G. N. da. **Monitoring and Control of Agricultural Environments Using Internet of Things**. 2023. Disponível em: https://www.repository.utl.pt/bitstream/10400.5/30003/1/DM-TMGNS-2023.pdf. Acesso em: 30 ago. 2024.

SOMMERVILLE, I. **Software Engineering**. 7th. ed. [*S. l.*]: Addison-Wesley, 2004.

SPINELLIS, D. Infrastructure as code. **IEEE Software**, v. 35, n. 5, p. 70–76, 2018.

SUTHERLAND, J.; SCHWABER, K. **Scrum**: The art of doing twice the work in half the time. New York: Crown Business, 2020.

APÊNDICE A – QUESTIONÁRIO DE AVALIAÇÃO DE USABILIDADE (UEQ)

A.1 Instrumento aplicado aos participantes

O questionário de usabilidade utilizado neste trabalho foi adaptado a partir do *User Experience Questionnaire* (UEQ), instrumento desenvolvido por Schrepp e Thomaschewski (2017), e teve como objetivo avaliar a experiência dos usuários em relação à aplicação desenvolvida para o gerenciamento de monitorias.

As respostas foram registradas em escala Likert de 1 (discordo totalmente) a 5 (concordo totalmente).

Instruções: Leia atentamente cada afirmação abaixo e marque o grau de concordância de acordo com sua experiência utilizando o sistema.

1. A aplicação é agradável e amigável. *(Atratividade)*
2. A interface é fácil de entender e aprender. *(Perspicuidade)*
3. As tarefas de cadastro, edição e remoção de monitorias são rápidas e eficientes. *(Eficiência)*
4. O sistema se comporta conforme o esperado e as funções são confiáveis. *(Dependabilidade)*
5. Usar a aplicação é interessante e motivador. *(Estimulação)*
6. A solução apresentada é inovadora e diferente das formas tradicionais de controle de monitorias. *(Originalidade)*
7. O sistema funcionou bem em diferentes dispositivos (computador, celular). *(Eficiência/-Dependabilidade)*
8. Senti-me seguro ao realizar alterações (edição/exclusão), recebendo mensagens de confirmação claras. *(Dependabilidade)*
9. De forma geral, fiquei satisfeito com a experiência de uso. *(Atratividade)*
10. Comentários e sugestões: *(Espaço livre para resposta)*

Escala utilizada:

- 1 — Discordo totalmente
- 2 — Discordo parcialmente
- 3 — Não concordo nem discordo
- 4 — Concordo parcialmente
- 5 — Concordo totalmente