UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Vinícius Bohrer dos Santos

IMPLEMENTAÇÃO DE UM PIPELINE DE INTEGRAÇÃO CONTÍNUA PARA O FRONT-END DE UMA APLICAÇÃO WEB DE CÓDIGO ABERTO

Santa Maria, RS
2023

Vinícius Bohrer dos Santos

IMPLEMENTAÇÃO DE UM PIPELINE DE INTEGRAÇÃO CONTÍNUA PARA O FRONT-END DE UMA APLICAÇÃO WEB DE CÓDIGO ABERTO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**. Defesa realizada por videoconferência.

Orientador: Prof. João Vicente Ferreira Lima

Santa Maria, RS
2023

Vinícius Bohrer dos Santos

IMPLEMENTAÇÃO DE UM PIPELINE DE INTEGRAÇÃO CONTÍNUA PARA O FRONT-END DE UMA APLICAÇÃO WEB DE CÓDIGO ABERTO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**.

Aprovado em 3 de fevereiro de 2023:

João Vicente Ferreira Lima, Dr. (UFSM)
(Presidente/Orientador)

Andrea Schwertner Charão, Dra. (UFSM)

João Carlos Damasceno Lima, Dr. (UFSM)

Santa Maria, RS
2023

A tecnologia é uma espada de dois gumes. Não temos escolha sobre se usá-la, mas sim sobre como usá-la.

(Steven Jobs)

RESUMO

IMPLEMENTAÇÃO DE UM PIPELINE DE INTEGRAÇÃO CONTÍNUA PARA O FRONT-END DE UMA APLICAÇÃO WEB DE CÓDIGO ABERTO

AUTOR: Vinícius Bohrer dos Santos
Orientador: João Vicente Ferreira Lima

Este trabalho apresenta a aplicação da prática de Integração Contínua na plataforma de ações solidárias “Faz um Bem!”. A Integração Contínua é uma técnica onde os desenvolvedores enviam frequentemente as alterações no código em que estão trabalhando para um repositório central, sem a necessidade de esperar o seu trabalho estar completo, entretanto sempre mantendo o sistema em um estado funcional. O objetivo principal da Integração Contínua é encontrar e investigar bugs mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de software.
A metodologia utilizada permitiu obter feedback sobre a qualidade do código e do funcionamento da aplicação de forma rápida e eficiente, facilitando a manutenção e a evolução do sistema.

Palavras-chave: pipeline. testes. automação. qualidade. integração contínua.

ABSTRACT

IMPLEMENTATION OF A CONTINUOUS INTEGRATION PIPELINE FOR THE FRONT-END OF AN OPEN-SOURCE WEB APPLICATION.

AUTHOR: Vinícius Bohrer dos Santos
ADVISOR: João Vicente Ferreira Lima

This work presents the application of the practice of Continuous Integration in the “Faz um Bem!” solidarity actions platform. Continuous Integration is a technique where developers frequently send changes in the code they are working on to a central repository, without the need to wait for their work to be complete, although always keeping the system in a functional state. The main objective of Continuous Integration is to find and investigate bugs more quickly, improve the software quality and reduce the time it takes to validate and release new software updates.
The methodology used allowed for feedback to be obtained on the quality of the code and the functioning of the application quickly and efficiently, facilitating maintenance and evolution of the system.

Keywords: pipeline. tests. automation. quality. continuous integration.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CD	Implantação Contínua
CDE	Entrega Contínua
CEP	Código de Endereçamento Postal
CI	Integração Contínua
$E2E$	End-to-end
ID	Identity
$JSON$	JavaScript Object Notation
JWT	JSON Web Token
NPM	Node Package Manager
$UFSM$	Universidade Federal de Santa Maria
UML	Unified Modeling Language
URL	Uniform Resource Locator
$YARN$	Yet Another Resource Negotiator

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos o modelo ágil de desenvolvimento de software tem crescido consideravelmente, pois existe uma demanda por uma solução que seja rápida na entrega de novas funcionalidades para o cliente e flexível para contínuas mudanças de requisitos. Entretanto, para provedores de serviços de TI, que lutam para manter a competitividade no mercado, essa demanda nunca acaba.

Como resultado surgiu uma prática chamada Integração Contínua, um método onde os desenvolvedores enviam frequentemente as alterações no código em que estão trabalhando para um repositório central, sem a necessidade de esperar o seu trabalho estar completo, entretanto, sempre mantendo o sistema em um estado funcional.

Assim que o código chega ao repositório central, o servidor de Integração Contínua entra em ação e roda uma série de processos automatizados, também conhecidos como pipeline. Os principais objetivos da Integração Contínua são encontrar e investigar bugs mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de software.

A Integração Contínua é uma prática muito utilizada já que realiza verificações e validações no código e rapidamente envia um feedback aos interessados caso algum dos processos retorne um resultado não esperado, aumentando a qualidade do software e a produtividade dos desenvolvedores. Com isso em mente, a proposta desse trabalho é desenvolver um Pipeline de Integração Contínua para o frontend da plataforma de ações solidárias “Faz um Bem!”.

A plataforma de ações solidárias “Faz um Bem!” é um projeto desenvolvido colaborativamente por alunos da Universidade Federal de Santa Maria para ser uma plataforma virtual interativa que centralize necessidades de instituições e organizações sociais da região. Assim, “Faz um Bem!”, propõe-se como uma plataforma virtual interativa, incentivadora e mediadora entre doadores e receptores de recursos.

E no desenvolvimento do pipeline será utilizado Jenkins, um servidor de integração contínua de código aberto para criar e automatizar o pipeline que vai compilar a aplicação e executar os testes, e Cypress, um framework de criação e execução de testes que será utilizada nos testes de aceitação da plataforma “Faz um Bem!”.

O restante deste trabalho está organizado em capítulos: no capítulo 2 é feito o embasamento teórico da proposta. No capítulo 3 são apresentadas as ferramentas que serão utilizadas, a criação do pipeline e dos testes, a execução e os resultados obtidos. Por fim, no capítulo 4 serão realizadas algumas considerações finais.

2 REFERENCIAL TEÓRICO

Este capítulo é dividido em cinco seções que abordam o tema de testes de software e pipelines de entrega contínua. Na primeira seção, são apresentadas as diferentes categorias de testes de software, incluindo testes manuais, automatizados, unitários, integrados, de aceitação, de regresso, de desempenho, de segurança e de sistema. Na segunda seção, é discutido o processo de integração, testes, deploy em ambiente de testes e deploy em ambiente de produção, assim como as práticas de integração contínua, entrega contínua e implantação contínua. Na terceira seção, são apresentadas as ferramentas utilizadas para integração contínua, como Jenkins, GitLab CI, Travis CI e CircleCI e as ferramentas para testes automatizados, como Cypress, Robot Framework e Selenium, incluindo o Selenium WebDriver, Selenium Grid e Selenium IDE. Na quarta seção, são apresentados três trabalhos relacionados. Finalmente, a quinta seção conclui o capítulo com uma discussão geral sobre o assunto.

2.1 TESTES DE SOFTWARE

Os testes de software são uma etapa crucial no processo de desenvolvimento de software, pois ajudam a garantir a qualidade e a confiabilidade do produto final. De acordo com (PRESSMAN, 1998) e (PRESSMAN, 2013), os testes de software podem ser divididos em vários tipos, incluindo:

- Testes manuais: São realizados por um humano, que verifica se o software está funcionando conforme os requisitos especificados. Esses testes são úteis para detectar problemas com o software, mas são limitados pelo fato de serem realizados por um humano.
- Testes automatizados: São realizados por um software especializado, que verifica se o software está respeitando os requisitos especificados. Esses testes são mais rápidos e precisos do que os testes manuais, pois são executados de forma automática e não dependem da intervenção humana.
- Testes unitários: São realizados em uma unidade de código específica, e ajudam a garantir que cada unidade de código esteja funcionando de forma correta. Esses testes são realizados principalmente no início do processo de desenvolvimento de software, para garantir que o código esteja correto antes de ser integrado ao sistema.
- Testes integrados: São realizados para verificar como os componentes individuais se integram entre si e como o sistema funciona como um todo. Eles podem incluir

testes de integração entre módulos, testes de integração com sistemas externos e testes de integração em diferentes ambientes.

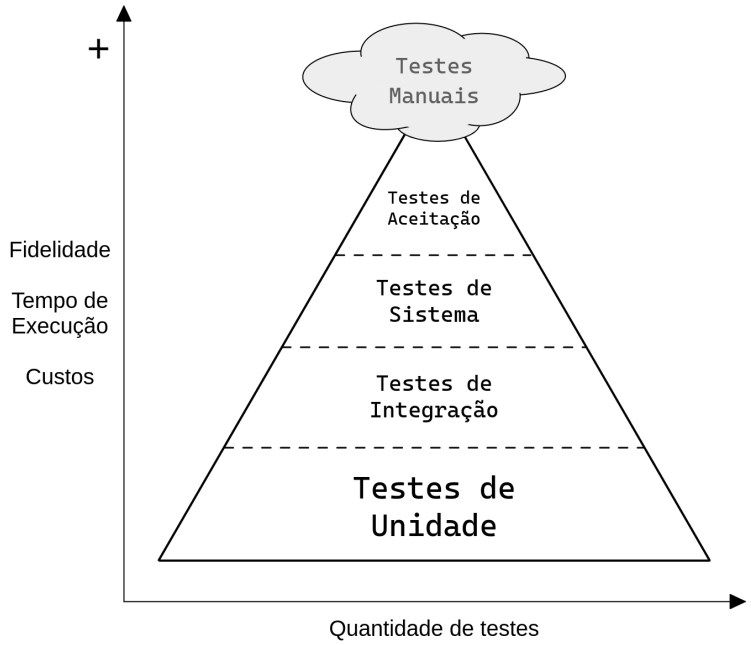
- Testes de aceitação: São realizados pelos usuários finais ou pelos representantes do negócio para garantir que o sistema atenda às suas necessidades e expectativas. Eles podem incluir testes manuais ou automatizados e podem ser baseados em cenários de uso ou requisitos do negócio.
- Testes de regresso: São testes realizados para verificar se uma alteração ou adição recente ao código-fonte não afetou o funcionamento correto do sistema. Eles são uma parte importante do processo de desenvolvimento de software e devem ser realizados sempre que uma alteração é feita no código-fonte.
- Testes de desempenho: São realizados para verificar se o software tem um desempenho adequado sob diferentes cargas e condições de uso.
- Testes de segurança: São realizados para verificar se o software é seguro contra ataques externos e vulnerabilidades.
- Testes de sistema: São realizados para garantir que o sistema funcione corretamente em sua configuração final, incluindo todas as dependências e integrações externas. Eles podem incluir testes de desempenho, testes de segurança e testes de aceitação.

Em sua obra, (HUMBLE; FARLEY; KIM, 2010) descrevem a Pirâmide de Testes como uma forma de garantir que os testes sejam realizados de forma eficiente e eficaz durante o processo de entrega de software. Eles sugerem que os testes devem ser organizados em quatro camadas, da base para o topo da pirâmide: testes de unidade, testes de integração, testes de sistema e testes de aceitação, como pode ser visto na figura 1.

Quando aplicada ao pipeline de entrega de software, a Pirâmide de Testes pode ser usada como uma forma de orientar a ordem em que os diferentes tipos de testes são realizados. Por exemplo, os testes de unidade podem ser realizados no início do pipeline, seguidos por testes de integração e testes de sistema. Os testes de aceitação podem ser realizados no final do pipeline, antes da implantação em ambiente de produção. Dessa forma, é possível garantir que os testes mais rápidos e específicos sejam realizados primeiro, o que pode ajudar a identificar problemas mais cedo no processo e tornar o feedback do processo de integração mais rápido e eficiente.

Ao seguir a Pirâmide de Testes, é possível garantir que os testes sejam realizados de forma eficiente e eficaz e que os problemas sejam identificados e corrigidos mais cedo no processo, o que pode ajudar a aumentar a qualidade e a confiabilidade do software entregue.

Figura 1 – Pirâmide de Testes.



Fonte: Autor. Adaptado de (COHN, 2009).

2.2 PIPELINES

Os pipelines são um conjunto de etapas automatizadas que são executadas para entregar software. Segundo (HUMBLE; FARLEY; KIM, 2010), os pipelines incluem todas as etapas do processo de entrega de software, desde a integração do código até o deploy em produção. Os pipelines são uma prática importante para garantir a qualidade do software e aumentar a eficiência da equipe de desenvolvimento.

Os pipelines podem ter diferentes estágios, dependendo do processo de entrega de software utilizado. Alguns exemplos de estágios comuns incluem:

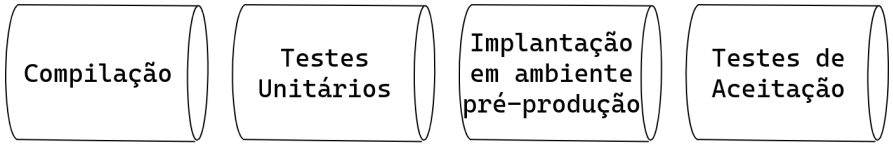
- Integração: Neste estágio, o código é integrado ao repositório principal e testado para garantir que ele esteja funcionando corretamente.
- Testes: Neste estágio, são realizados testes unitários e integrados para garantir que o código esteja funcionando corretamente.
- Deploy em ambiente de testes: Neste estágio, o código é implantado em um ambiente de testes para que os testes de aceitação sejam realizados.
- Deploy em ambiente de produção: Neste estágio, o código é implantado em produção e fica disponível para os usuários finais.

Alguns dos conceitos relacionados ao pipeline incluem integração contínua, entrega contínua e implantação contínua.

2.2.1 Integração Contínua

A Integração Contínua (CI) é um processo no qual o código é integrado frequentemente ao repositório principal de código. Segundo (PRESSMAN, 2013), a CI é um processo automatizado que verifica se o código pode ser integrado sem problemas. Esse processo inclui a execução de determinados testes, como, por exemplo, testes de unidade e testes de integração, para garantir que o código esteja funcionando corretamente antes de ser integrado ao sistema. A CI é uma prática importante para garantir a qualidade do software e para evitar problemas de integração no futuro.

Figura 2 – Exemplo de estágios que compõem um Pipeline de Integração Contínua.

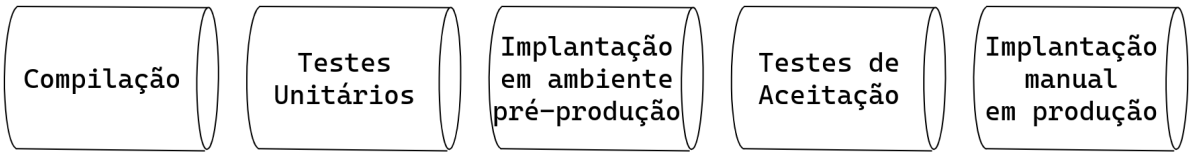


Fonte: Autor.

2.2.2 Entrega Contínua

A Entrega Contínua (CDE) é um processo no qual o código está sempre pronto para ser implantado em produção. Entretanto, como pode ser visto no último estágio do exemplo da figura 3, a implantação precisa ser autorizada. De acordo com (HUMBLE; FARLEY; KIM, 2010), a CDE é uma abordagem que permite que as equipes de desenvolvimento entreguem novas funcionalidades rapidamente e com confiança: elas são testadas automaticamente a cada alteração no código. A CDE envolve a automação de todo o processo de entrega, desde a integração do código até o deploy em produção. Esse processo ajuda a garantir a qualidade do software e aumenta a eficiência da equipe de desenvolvimento.

Figura 3 – Exemplo de estágios que compõem um Pipeline de Entrega Contínua.

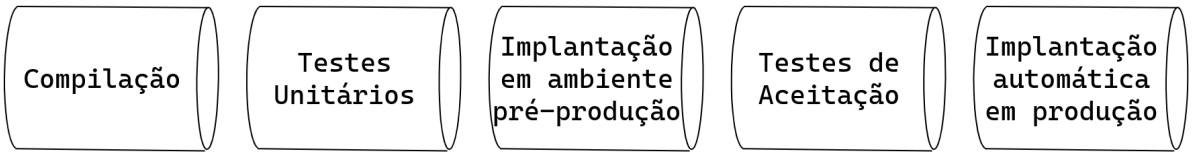


Fonte: Autor.

2.2.3 Implantação Contínua

A Implantação Contínua (CD) é um processo no qual o software é implantado automaticamente em produção sem intervenção humana, como pode ser visto no último estágio do exemplo da figura 4. Segundo (HUMBLE; FARLEY; KIM, 2010), a CD é uma evolução da entrega contínua, pois inclui a automação do processo de implantação em produção. A CD é uma prática importante para garantir que o software seja entregue rapidamente e com confiança, pois elimina a necessidade de intervenção humana no processo de implantação.

Figura 4 – Exemplo de estágios que compõem um Pipeline de Implantação Contínua.



Fonte: Autor.

2.3 FERRAMENTAS UTILIZADAS

2.3.1 Ferramentas de Integração Contínua

Algumas das principais ferramentas de integração contínua incluem Jenkins, GitLab CI, Travis CI e CircleCI.

- Jenkins: é uma das ferramentas de integração contínua mais populares e amplamente utilizadas. É de código aberto, possui uma grande comunidade ativa e oferece

uma ampla variedade de plugins para integração com outras ferramentas e serviços. Além disso, é altamente configurável e permite a criação de pipelines de integração contínua de forma fácil e intuitiva.

- GitLab CI: é uma ferramenta de integração contínua integrada ao GitLab, o qual é uma plataforma de gerenciamento de código-fonte. Ele permite a criação de pipelines de integração contínua com base em configurações em um arquivo .gitlab-ci.yml, que é adicionado ao repositório do projeto.

- Travis CI: é uma ferramenta de integração contínua baseada na nuvem que é especialmente popular entre projetos de código aberto no GitHub. Ele permite a criação de pipelines de integração contínua com base em configurações em um arquivo .travis.yml, que é adicionado ao repositório do projeto.

- CircleCI: é uma ferramenta de integração contínua baseada na nuvem que oferece suporte a várias linguagens de programação e integrações com várias plataformas, incluindo GitHub, GitLab e Bitbucket. Ele permite a criação de pipelines de integração contínua com base em configurações em um arquivo .circleci/config.yml, que é adicionado ao repositório do projeto.

O Jenkins é uma opção popular para implementar um pipeline de integração contínua devido à sua ampla variedade de recursos, sua comunidade ativa e sua facilidade de uso. Ele é de código aberto e possui uma grande variedade de plugins disponíveis, o que o torna altamente configurável e capaz de se integrar com uma ampla variedade de outras ferramentas e serviços. Além disso, é uma ferramenta amplamente utilizada e possui uma grande base de usuários ativos, o que garante suporte e atualizações constantes.

2.3.2 Ferramentas para testes automatizados

Existem várias ferramentas disponíveis para testes de aceitação (E2E) automatizados. Algumas opções populares incluem o Cypress, Robot Framework e Selenium.

- Cypress: é uma ferramenta de teste de aceitação de código aberto que utiliza JavaScript, mesma linguagem de programação utilizada no frontend, que é o foco dos testes de aceitação. Ele é executado diretamente no navegador e fornece uma interface de usuário para escrever e executar testes. Além disso, o Cypress oferece uma série de recursos, como gravação de tela, suporte a vários navegadores e uma série de plugins disponíveis que permitem estender as funcionalidades do cypress para diferentes áreas, tornando o processo de teste mais eficiente e automatizado. Alguns exemplos de plugins são:

  - cypress-image-snapshot: que permite capturar imagens da tela durante os testes e compará-las com imagens anteriormente capturadas.
  - cypress-browser-logs: para coletar logs do browser durante a execução dos testes.
  - cypress-axe: para testar acessibilidade na aplicação.
  - cypress-faker: permite que você crie dados falsos personalizados para atender às suas necessidades de teste.
  - cypress-random-user-agent: permite que você teste a sua aplicação com diferentes tipos de navegadores, dispositivos e sistemas operacionais simulando a visita de usuários com diferentes configurações.

- Robot Framework: é uma ferramenta de teste de aceitação de código aberto que é baseada em Python e suporta várias linguagens de programação. Ele possui uma ampla gama de recursos, incluindo uma extensa biblioteca de palavras-chave e integração com várias ferramentas de teste, como o Selenium. Além disso, ele possui plugins que permitem a extensão de suas funcionalidades. Ele também tem uma comunidade ativa e documentação extensa, o que ajuda os usuários a aprender e usar a ferramenta de maneira eficiente.

- Selenium: é um projeto de abrangência que inclui uma série de ferramentas e bibliotecas que permitem e dão suporte à automação de navegadores web. Ele é compatível com várias linguagens de programação e é uma opção amplamente utilizada em testes automatizados de aceitação devido à sua flexibilidade e ampla gama de recursos. É composto por vários componentes, incluindo o Selenium WebDriver, o Selenium Grid e o Selenium IDE. Além disso, assim como as outras ferramentas citadas, existem vários plugins disponíveis para o Selenium que fornecem recursos adicionais para a automação de testes.

  - Selenium WebDriver: é a parte principal do Selenium que permite a automação de testes em diferentes navegadores. Ele é uma interface de programação de aplicativos (API) que fornece uma forma de acessar os recursos do navegador de forma automatizada e é compatível com várias linguagens de programação.
  - Selenium Grid: é uma ferramenta que permite a execução de testes em diferentes sistemas operacionais e navegadores em paralelo, otimizando o tempo de teste. Ele é útil quando é necessário testar a compatibilidade do aplicativo em diferentes ambientes.
  - Selenium IDE: é uma extensão do navegador que permite a gravação e reprodução de testes no navegador. Ele é útil para a criação rápida de testes de aceitação simples e para a depuração de testes mais complexos.

Observe que o Selenium suporta várias linguagens de programação por padrão, enquanto o Cypress é limitado ao JavaScript e o Robot Framework é limitado ao Python. No entanto, cada uma dessas ferramentas pode ser estendida com plugins para suportar outras linguagens. Outro ponto que vale ressaltar é que o Selenium é composto por vários componentes que são necessários para utilizar a ferramenta. Já o Cypress e o Robot Framework só precisam ser instalados como um único pacote.

Neste projeto, o Cypress foi escolhido para a implementação dos testes de aceitação devido à sua integração direta com o navegador, o que permite o acesso direto aos elementos da página. Além disso, o Cypress é um pacote único que inclui tudo o que é necessário para o desenvolvimento de testes de aceitação, permitindo a utilização do framework por pessoas sem conhecimento específico em automatização de testes. A compatibilidade com JavaScript, que é a linguagem de programação utilizada no frontend, também é uma vantagem, pois permite que os desenvolvedores que trabalham na interface escrevam os testes de aceitação.

2.4 TRABALHOS RELACIONADOS

Os trabalhos relacionados apresentam diferentes soluções para automatizar o processo de testes em pipelines de entrega contínua. O trabalho (WOLF; YOON, 2016) utiliza a ferramenta Jenkins e o Protractor para realizar testes de aceitação em aplicações AngularJS e Angular. O objetivo do trabalho é discutir a importância da automação de testes para a implementação de um processo de entrega contínua, concluindo que a automação de testes é fundamental para a entrega contínua de atualizações de qualidade para os clientes.

Já o trabalho (NGUYEN et al., 2020) utiliza o GitLab CI para implementar um teste de ponta a ponta (E2E) automatizado no processo atual de um sistema de gerenciamento de experiência da web (WEMP). Os estágios incluem compilação, deploy em ambiente de pré-produção e, na branch principal, deploy para produção e testes de aceitação. A ferramenta de testes de aceitação é o Robot Framework e foram selecionados 6 casos de teste baseados em conjuntos de User Stories (Épicos). O objetivo do trabalho é implementar um processo de teste E2E automatizado com integração contínua (CI), entrega contínua (CD) e configuração de monitoramento. A conclusão é que o resultado do projeto é o processo de teste E2E automatizado mencionado, além de destacar que, apesar de ser a melhor opção para testes E2E, o Robot Framework possui algumas limitações.

E por fim, o trabalho (SINGH, 2022) utiliza a ferramenta GitLab CI para criar uma pipeline de CI/CD com os estágios de compilação, testes e implantação. A ferramenta de testes de aceitação é o Cypress e apenas um exemplo com 3 ações de teste foi selecionado. O objetivo do trabalho é melhorar o processo de desenvolvimento e entrega de apli-

cativos automatizando processos de desenvolvimento genéricos e demorados, integrando o uso de métodos de CI/CD ao processo de desenvolvimento. A conclusão é que foi criada uma pipeline CI/CD simples e detalhada que constrói, testa e implanta automaticamente a aplicação de demonstração “Our Travel Gallery”.

2.5 CONCLUSÃO

Os testes de software são fundamentais para garantir a qualidade e confiabilidade de um software. A utilização de pipelines de integração contínua com testes de aceitação automatizados pode trazer vantagens como aumento da agilidade e eficiência do processo de desenvolvimento, diminuição do tempo de entrega e redução de erros no software final. Portanto, é importante considerar a utilização de pipelines de integração contínua com testes de aceitação automatizados em projetos de software em que agilidade e qualidade são importantes.

3 METODOLOGIA

Neste capítulo é apresentado o processo de desenvolvimento do projeto de pipeline de integração contínua com testes de aceitação para a plataforma “Faz um Bem!”. A primeira seção descreve a arquitetura da aplicação, a qual é uma arquitetura de três camadas composta de um banco de dados, um backend e um frontend. A segunda seção apresenta a arquitetura do pipeline de integração contínua. Na terceira seção é detalhado o processo de desenvolvimento e configuração do pipeline e dos testes de aceitação. Finalmente, na quarta seção são discutidos os resultados do projeto.

3.1 PLATAFORMA FAZ UM BEM!

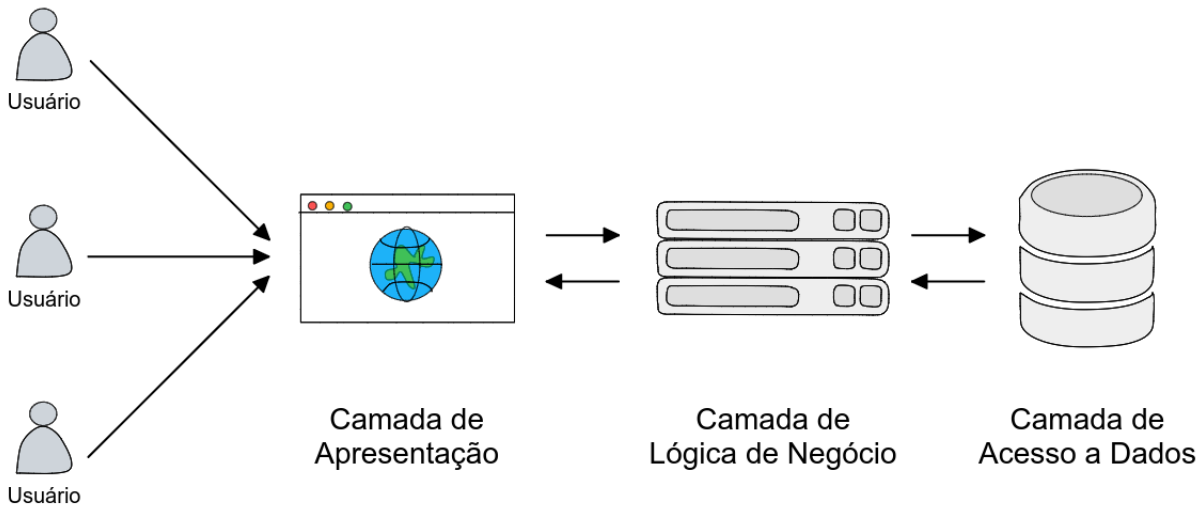
O projeto “Faz um Bem!” é uma iniciativa interdisciplinar da Universidade Federal de Santa Maria (UFSM) que tem como objetivo desenvolver uma plataforma virtual para conectar doadores e instituições sociais de Santa Maria. A plataforma visa incentivar e mediar a doação de recursos para entidades que atendem pessoas em situação de vulnerabilidade social.

A plataforma foi feita utilizando a arquitetura de três camadas, como foi exemplificado na figura 5, é uma estrutura de software que separa o aplicativo em três camadas distintas: camada de apresentação, camada de negócios e camada de dados. A camada de apresentação é responsável por exibir as informações para o usuário e capturar a sua entrada, a camada de negócios lida com as regras de negócios e lógica de aplicação, e a camada de dados é responsável por gerenciar o banco de dados. No projeto, a camada de apresentação foi desenvolvida usando o TypeScript e o React, a camada de negócios foi desenvolvida usando Java e o Spring Boot, e a camada de dados foi desenvolvida usando PostgreSQL como banco de dados.

3.1.1 Camada de Apresentação

No desenvolvimento do frontend foi utilizada a linguagem de programação TypeScript, que é baseada em JavaScript e possui recursos de tipagem estática. Isso permite a definição explícita dos tipos de dados de variáveis, funções e outros elementos, ajudando a detectar erros de digitação e lógica durante o desenvolvimento, em vez de detectá-los em tempo de execução. Além disso, o TypeScript oferece suporte a classes, interfaces, tipos genéricos e outras características avançadas, tornando o código mais organizado.

Figura 5 – Diagrama Arquitetural da Aplicação.



Fonte: Autor.

Para a organização e estruturação do código, foi utilizada a arquitetura de componentização, que se baseia na criação de componentes pequenos, reutilizáveis e independentes, que podem ser combinados para criar interfaces mais complexas. Cada componente é responsável por uma parte específica da interface, como um botão, um formulário ou uma tela inteira. Dessa forma, é possível manter o código organizado e facilmente escalável.

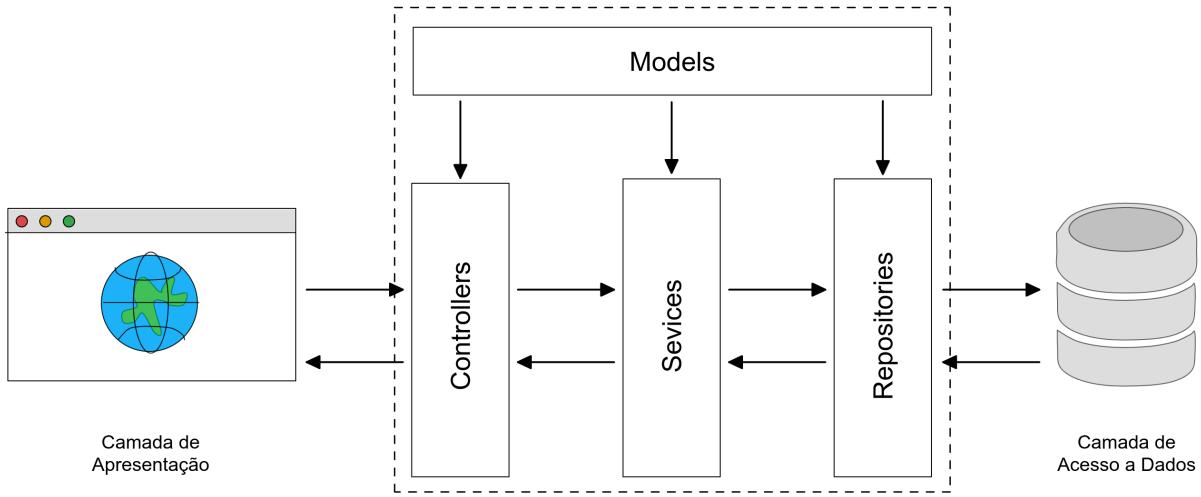
O projeto também utiliza o framework React e possui rotas públicas e privadas. As rotas públicas incluem tela de login, cadastro, campanhas, instituições, sobre e contato. Já as rotas privadas incluem campanhas, nova campanha, instituições, nova instituição, entre outras, que só estão disponíveis para usuários autenticados. Algumas das rotas privadas também possuem restrições de acesso para usuários com papel de administrador. Todo esse gerenciamento é feito com a biblioteca React Router.

Para a estilização e responsividade das páginas, foi utilizada a biblioteca Chakra UI, e para gerenciamento de estados da aplicação e da autenticação de usuário, foi utilizada a funcionalidade Context presente no Framework React.

3.1.2 Camada de Lógica de Negócios

A arquitetura do Backend segue o padrão de projetos de software conhecido como arquitetura em camadas como pode ser visto no diagrama na figura 6, o que significa que os componentes da aplicação são separados em camadas distintas, no caso dessa aplicação ela são: Controller, Services, Repositories e Models.

Figura 6 – Diagrama Arquitetural da Camada de Lógica de Negócios.



Fonte: Autor.

- A camada de Controller é responsável por lidar com as regras de negócios e a interação entre as demais camadas. Ela é responsável por receber as requisições HTTP e fornecer respostas para as mesmas.
- A camada de Services é responsável por manipular os dados de acordo com as regras de negócios da aplicação. Ela é responsável por realizar operações como validação, cálculos e lógica de negócios.
- A camada de Repositories é responsável por gerenciar o acesso aos dados da aplicação. Ela é responsável por realizar operações como busca, inserção, atualização e deleção de dados.
- A camada de Models representa as entidades da aplicação, como usuários, campanhas, endereços e tokens, e é responsável por lidar com os dados de negócio.

Também é utilizado o padrão DTO (Data Transfer Object) que é responsável por formatar os dados para serem retornados ao cliente. Essa camada é utilizada para garantir que os dados sejam retornados de maneira segura e de acordo com as necessidades do cliente.

A aplicação também conta com um sistema de segurança baseado em JWT (JSON Web Token), com classes como `JwtRequestFilter`, `JwtUserDetailsService`, `JwtTokenUtil` e `JwtAuthenticationEntryPoint`, que trabalham juntas para autenticar e autorizar as requisições de acordo com as regras de segurança da aplicação.

A utilização dessas camadas separa a lógica de acesso a dados da lógica de negócios, permitindo que essas camadas sejam facilmente testadas e reutilizadas em outras

partes da aplicação, além de manter o código limpo e organizado. Isso permite que a API seja escalável e flexível, possibilitando novas funcionalidades e melhorias ao longo do tempo.

A API disponibilizada pela aplicação possui rotas para lidar com autenticação, usuários, instituições e campanhas. Entre elas:

- `/user`, com método `PUT` para atualizar informações de usuário ou responsável pela instituição.
- `/campaign`, com métodos `PUT` e `POST` para atualizar e criar informações de campanha, respectivamente.
- `/user/resetpassword/{username}`, com método `POST` para redefinir a senha de um usuário específico.
- `/user/register`, com método `POST` para registrar um novo usuário.
- `/user/authenticate`, com método `POST` para autenticar um usuário existente.

Essas rotas permitem que os usuários possam criar, atualizar e autenticar suas contas, além de criar e atualizar campanhas. A rota de redefinição de senha é uma funcionalidade importante para garantir a segurança dos usuários, permitindo que eles possam redefinir suas senhas em caso de esquecimento.

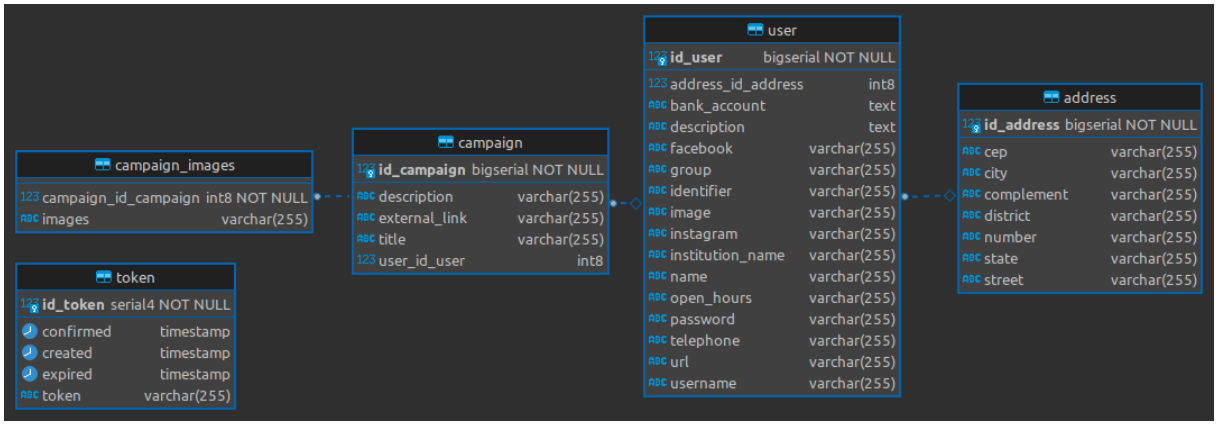
3.1.3 Camada de Acesso a Dados

O banco de dados PostgreSQL é um sistema de gerenciamento de banco de dados relacional de código aberto. Ele é utilizado para armazenar, recuperar e gerenciar grandes quantidades de dados.

Existem cinco tabelas no banco de dados, como pode ser visto no diagrama UML na figura 7, cada uma tem suas próprias colunas e chaves primárias e estrangeiras:

- `user` armazena informações sobre usuários, como conta bancária, descrição, Facebook, grupo, identificador, imagem, Instagram, nome da instituição, nome, horário de funcionamento, senha, telefone, URL, nome de usuário e o ID do endereço. A chave primária é `id_user` e há uma chave estrangeira para a tabela `address`.
- `address` armazena informações de endereço, como CEP, cidade, complemento, bairro, número, estado e rua. A chave primária é `id_address`.
- `token` armazena informações sobre tokens, como data de confirmação, criação, expiração e o próprio token JWT, utilizado para autenticação do usuário. A chave primária é `id_token`.

Figura 7 – Diagrama UML do Banco de Dados.



Fonte: DBeaver (DBeaver Community, [2023]).

- `campaign` armazena informações sobre campanhas, como descrição, link externo, título e o ID do usuário. A chave primária é `id_campaign` e há uma chave estrangeira para a tabela `user`.
- `campaign_images` armazena informações sobre imagens de campanhas e há uma chave estrangeira para a tabela `campaign`

3.2 ARQUITETURA PROPOSTA

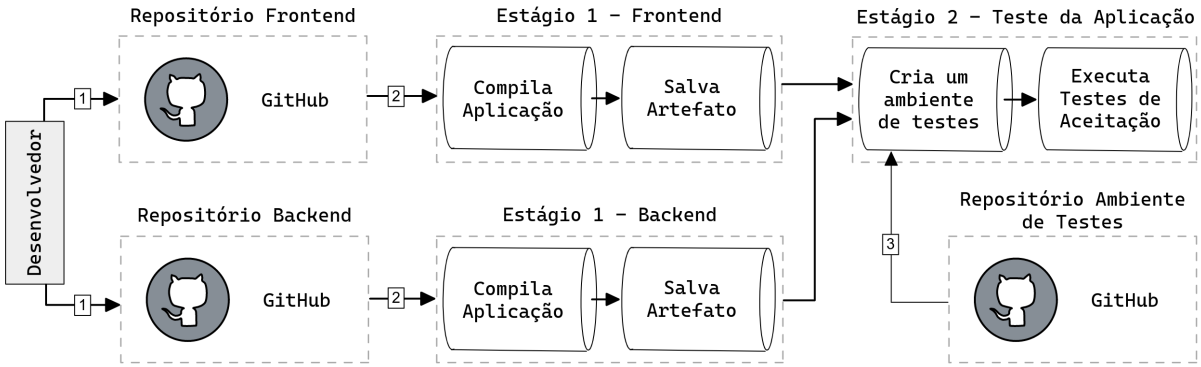
A arquitetura proposta para o projeto, descrita na figura 8, é composta por um pipeline de integração contínua implementado em Jenkins e testes de aceitação desenvolvidos com o Cypress. Nesta figura, pode-se observar a existência de conexões entre os blocos, bem como a presença de diferentes tipos de conexões:

- A conexão identificada com “1”: é feita quando o desenvolvedor envia o código para o repositório central (GitHub) através do comando `git push`.
- A conexão identificada com “2”: é feita quando o servidor de integração contínua recebe um webhook do GitHub, indicando que houve uma atualização no código-fonte da aplicação.
- A conexão identificada com “3”: é feita pelo servidor de integração contínua para buscar a configuração do ambiente, que também está armazenado em um repositório do GitHub, através do comando `git clone`.

- Todas as outras conexões não marcadas ocorrem dentro do servidor de integração contínua sempre que a etapa anterior executa sem erros.

O webhook é uma funcionalidade que permite que o GitHub envie uma notificação para um servidor externo sempre que houver uma atualização no repositório. Isso permite que outras aplicações possam se comunicar com o GitHub e serem notificadas sempre que houver mudanças no código-fonte. No contexto do pipeline de integração contínua, o webhook do GitHub é utilizado para notificar o Jenkins que houve uma atualização no código-fonte e assim iniciar o processo de integração e testes automatizados.

Figura 8 – Diagrama da Arquitetura Proposta. Dois pipelines de primeiro estágio, com validações específicas da camada, se conectam com um pipeline de segundo estágio, com os testes de aceitação da aplicação.



Fonte: Autor.

3.3 DESENVOLVIMENTO

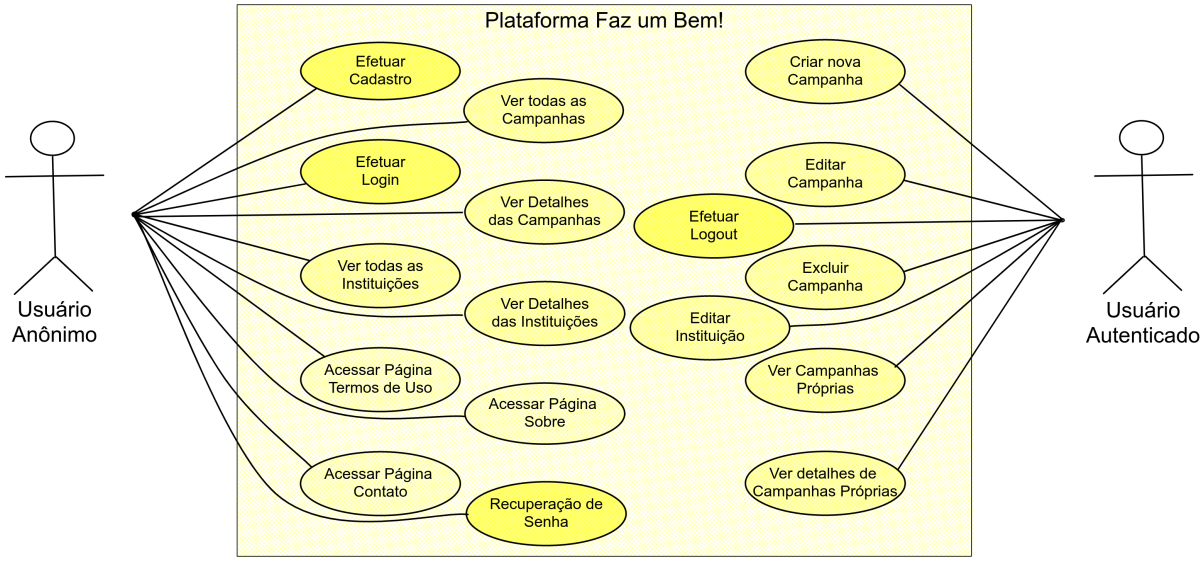
Antes de iniciar a implementação de testes de aceitação, foi necessário realizar uma análise dos casos de uso da aplicação. Com base nos casos de uso, os testes de aceitação foram desenvolvidos e então adicionados no repositório do ambiente de testes com outros arquivos de configuração para executar no segundo estágio do pipeline.

3.3.1 Casos de Uso

Para determinar os casos de uso, foi necessário analisar o código-fonte do frontend, que incluía as telas da aplicação, e do backend, que incluía as rotas da aplicação. Além disso, foi feita uma exploração manual da aplicação, para ter uma compreensão mais completa de suas funcionalidades.

Uma vez que os casos de uso foram identificados, eles foram organizados em um diagrama de caso de uso UML, como pode ser visto na figura 9. Esse diagrama permitiu uma visão geral dos casos de uso e forneceu uma base para a elaboração de testes de aceitação.

Figura 9 – Diagrama de casos de uso UML da plataforma, ele serve como uma visão geral dos recursos e funcionalidades do sistema e como eles são acessíveis para diferentes tipos de usuários.



Fonte: Autor.

3.3.2 Testes de Aceitação

Para iniciar o desenvolvimento dos testes de aceitação foi efetuada a instalação do Cypress utilizando um gerenciador de pacotes JavaScript, como npm ou yarn. Em seguida, foi necessário configurá-lo, utilizando o arquivo cypress.config.js para configurar variáveis de ambiente e o arquivo cypress/support/e2e.js para definir configurações como endereço da aplicação que passará pelos testes e plugins.

A instalação dos plugins também pode ser feita utilizando algum gerenciador de pacotes JavaScript. Os plugins que foram utilizados neste projeto foram:

- cypress-image-upload: fornece uma maneira fácil de lidar com o upload de arquivos de imagem durante os testes, permitindo que você configure o caminho para o arquivo de imagem e verifique se ele foi corretamente carregado. Ele também oferece recursos adicionais como redimensionamento e comparação de imagens.

- cypress-maildev: permite simular o envio de e-mails durante os testes, possibilitando verificar se os e-mails foram enviados corretamente e se possuem os conteúdos esperados.

Junto ao plugin cypress-maildev foi adicionado um servidor SMTP no ambiente de testes, com o objetivo de capturar os e-mails enviados pela aplicação durante a execução dos testes.

Os testes são criados dentro da pasta cypress/e2e, geralmente com a extensão .cy.js. O Cypress possui uma série de comandos que permitem interagir com a página, como `visit()` para acessar uma URL, `get()` para selecionar elementos HTML e `should()` para verificar se determinado comportamento está ocorrendo corretamente. Um exemplo de teste no Cypress é verificar se o título da página é “Cypress Test” quando a página é carregada:

```
describe("Formulário de Login", () => {
    beforeEach(() => {
        cy.visit("/login");
    });
    it("deve receber erro ao informar dados incorretos", () => {
        cy.getLoginData().then((login) => {
            cy.get('input[name="email"]').type(login.email);
        });
        cy.get('input[name="password"]').type("senha-errada");
        cy.get("button[type='submit']").click();
        cy.get('[role="status"]').should(
            "contain",
            "Verifique seus dados e tente novamente"
        );
    });
});
```

Assim foram desenvolvidos os testes de aceitação para todos os casos de uso da aplicação citados anteriormente. A execução dos testes é feita através do comando `cypress open`, que permite visualizar os testes sendo executados em tempo real, ou o comando que permite executar os testes sem abrir o navegador: `cypress run --headless`. O último método será o utilizado, pois é essencial para rodar os testes em um servidor de integração contínua.

3.3.3 Pipeline de Integração Contínua

O Jenkins é uma ferramenta “self-hosted”, o que significa que é necessário baixar e instalar em um servidor próprio. Uma vez instalado, é possível configurar as ferramentas necessárias, como plugins e pipelines.

Para instalar novos plugins no Jenkins, é necessário acessar o menu “Gerenciar Jenkins” e selecionar a opção “Gerenciar plugins”. É possível buscar e instalar os plugins desejados. Neste projeto foram utilizados os seguintes plugins:

- Docker Pipeline: que permite ao Jenkins utilizar contêineres para a execução de tarefas e automatizar a construção e implantação de aplicativos em contêineres. Com esse plugin, é possível configurar pipelines de automação de construção e teste.
- Build Pipeline: que permite a criação de pipelines de construção, permitindo visualizar o estado e o progresso das etapas do pipeline.
- GitHub Commit Status: permite alterar o status do commit no Github, permitindo que o Jenkins atualize o status dos commits com o resultado dos testes e o estado do pipeline.

Ao configurar os pipelines deve ser informado se vai ser descrito pela interface do Jenkins ou utilizando um Jenkinsfile. Um Jenkinsfile é um arquivo de script que descreve etapas e configurações para o pipeline. Ele é geralmente armazenado no repositório de código da aplicação e é usado pelo Jenkins para executar as tarefas especificadas. O Jenkinsfile utilizado no backend pode ser visto abaixo:

```
1  pipeline {
2      agent any
3      stages {
4          stage('Build') {
5              steps {
6                  sh 'docker build --rm -t vbsantos-tcc/backend:latest .'
7              }
8          }
9      }
10 }
```

No projeto, a configuração dos pipelines foi realizada utilizando o Jenkinsfile, um arquivo de script que descreve as etapas e configurações do pipeline. Ele é armazenado no repositório de código da aplicação e é usado pelo Jenkins para executar as tarefas especificadas. A primeira etapa dos pipelines do frontend e do backend foi descrita através desse arquivo, enquanto a segunda etapa foi configurada diretamente na interface do Jenkins.

Assim, a cada alteração no código-fonte da aplicação, o Jenkins é notificado e inicia a execução dos passos descritos no Jenkinsfile, o principal passo é criação de uma imagem docker utilizando a descrição que está no arquivo Dockerfile, também armazenado dentro do repositório do projeto. O Dockerfile utilizado no backend pode ser visto abaixo:

```
FROM maven:3.6.0-jdk-11-slim AS build
COPY src /home/app/src
COPY pom.xml /home/app
RUN mvn -f /home/app/pom.xml clean package
FROM openjdk:11-jre-slim
EXPOSE 8080
COPY --from=build /home/app/target/app-0.0.1-SNAPSHOT.jar /app.jar
CMD [ "java", "-jar", "/app.jar" ]
```

Caso não ocorram erros durante a execução da primeira etapa, o pipeline avança para a segunda etapa, onde são executados os testes de aceitação. A segunda etapa do pipeline foi desenvolvida usando a mesma linguagem utilizada na etapa anterior, porém diretamente na interface do Jenkins. Os estágios mais importantes podem ser vistos abaixo:

```
stages {
  // ...
  stage('Setup Test Environment Configuration') {
    steps {
      sh '''
        git clone "https://github.com/vbsantos/fazumbem-test-env-tcc.git"
        mv fazumbem-test-env-tcc/* .
        mv fazumbem-test-env-tcc/.[!.]* .
        chmod +x ./*.sh
      '''
    }
  }
  stage('Build Acceptance Test Environment') {
    steps { sh './build-test-env.sh' }
  }
  stage('Run Acceptance Tests') {
    steps { sh './run-automated-acceptance-tests.sh' }
  }
  // ...
}
```

E, por fim, a integração com o Github permite que o Jenkins altere o status dos commits, informando sobre o resultado dos testes e o estado do pipeline. Dessa forma, é possível acompanhar o progresso do desenvolvimento e identificar problemas rapidamente.

3.4 RESULTADOS

O projeto proposto contemplou todos os casos de uso da aplicação. O primeiro estágio dos pipelines do frontend e do backend compilam a aplicação e constroem a imagem Docker rapidamente, como pode ser visto no quadro 1. Assim como o segundo estágio do pipeline, que consiste na implantação do ambiente de testes e execução dos testes de aceitação.

Quadro 1 – Tempos de execução dos estágios do pipeline.

Estágio	**Execução 1**	**Execução 2**	**Execução 3**	**Média**
Primeiro estágio - Frontend	0:01:15	0:01:15	0:01:12	0:01:14
Primeiro estágio - Backend	0:05:17	0:04:31	0:04:34	0:04:47
Segundo estágio	0:04:38	0:04:36	0:04:19	0:04:31

Fonte: Autor.

É importante mencionar que os tempos de execução dos pipelines podem variar dependendo da velocidade da internet, das configurações do servidor e das configurações dos pipelines. Até o momento, nenhuma medida foi tomada para otimizar esses tempos. No entanto, com a metodologia adotada, foi possível obter um feedback sobre a qualidade do código e do funcionamento da aplicação em menos de 10 minutos.

Com relação aos testes de aceitação, embora tenham sido realizados com sucesso, foram encontrados erros em três dos 17 casos de uso testados, como pode ser visto na figura 10. Esses erros foram:

- Caso de uso “Acessar página termos de uso” com usuário não-autenticado: a página não existe.
- Caos de uso “Ver detalhes de instituições” com usuário não-autenticado: a página só pode ser acessada por meio de um botão dentro da aplicação, mas não de pela URL.
- Caos de uso “Ver detalhes de campanhas” com usuário não-autenticado: a página só pode ser acessada por meio de um botão dentro da aplicação, mas não de pela URL.

Figura 10 – Trecho final do relatório gerado pelo Cypress com o resultado e o tempo de cada teste.

	Spec		Tests	Passing	Failing	Pending	Skipped
✔	T01-register.cy.js	00:22	3	3	-	-	-
✔	T02-login.cy.js	00:09	2	2	-	-	-
✔	T03-editar-instituição.cy.js	00:07	1	1	-	-	-
✔	T04-nova-campanha.cy.js	00:14	2	2	-	-	-
✔	T05-campanhas.cy.js	00:05	1	1	-	-	-
✔	T06-ver-campanha.cy.js	00:05	1	1	-	-	-
✔	T07-editar-campanha.cy.js	00:05	1	1	-	-	-
✔	T08-excluir-campanha.cy.js	00:05	1	1	-	-	-
✔	T09-logout.cy.js	00:03	1	1	-	-	-
✔	T10-recuperacao-de-senha.cy.js	00:05	1	1	-	-	-
✔	T11-contact.cy.js	00:07	2	2	-	-	-
✔	T12-about.cy.js	00:03	1	1	-	-	-
✔	T13-campaigns.cy.js	00:06	2	2	-	-	-
✖	T14-campaign-id.cy.js	00:07	2	1	1	-	-
✔	T15-institutes.cy.js	00:05	2	2	-	-	-
✖	T16-institute-id.cy.js	00:07	2	1	1	-	-
✖	T17-termo.cy.js	00:08	1	-	1	-	-
✖	3 of 17 failed (18%)	02:11	26	23	3	-	-

Fonte: Cypress (Cypress.io, [2023]).

Os repositórios do projeto, incluindo o backend[1] e o frontend[2], podem ser encontrados no Github. Nestes repositórios, é possível encontrar pequenas alterações do projeto original[3], como a adição do Dockerfile, do Jenkinsfile e a maioria das configurações que se tornaram variáveis de ambiente para serem determinadas no momento do deploy. Além disso, há também um repositório com as configurações do ambiente de testes[4] onde é possível encontrar os arquivos do Cypress com vídeos de todos os testes que foram exe-

[1] <https://github.com/vbsantos/fazumbem-backend-tcc>
[2] <https://github.com/vbsantos/fazumbem-frontend-tcc>
[3] <https://github.com/AndreaInfUFSM/fazumbem>
[4] <https://github.com/vbsantos/fazumbem-test-env-tcc>

cutados com sucesso e screenshots com os erros dos testes que falharam. Isso permite que qualquer pessoa interessada possa ter acesso aos resultados dos testes e verificar como o projeto foi executado. Além disso, esses repositórios também são úteis para futuras manutenções e evoluções do projeto.

Conclui-se que a metodologia utilizada foi eficiente, permitindo obter feedback sobre a qualidade do código e do funcionamento da aplicação de forma rápida. Isso facilita a manutenção e a evolução do sistema, garantindo que ele esteja funcionando corretamente e possa ser melhorado continuamente.

4 CONCLUSÃO

Este trabalho apresentou a implementação de Integração Contínua com testes de aceitação na plataforma de ações solidárias “Faz um Bem!”. Utilizando Jenkins como servidor de Integração Contínua e Cypress como framework de testes, foi desenvolvido um pipeline de Integração Contínua que compila a aplicação e executa testes de aceitação headless. Como resultado, foi possível obter feedback sobre a qualidade do código e do funcionamento da aplicação de forma rápida e eficiente, facilitando a manutenção e a evolução do sistema. A Integração Contínua é uma prática fundamental para provedores de serviços de TI que buscam manter a competitividade no mercado, pois ajuda a encontrar e investigar erros mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de software.

Para futuros trabalhos na área de testes, sugere-se adicionar testes de unidade e testes de mutação para garantir a qualidade do código. Além disso, seria interessante utilizar ferramentas para análise estática de código para detectar bugs. Outra sugestão é implementar testes de integração para garantir a correta interação entre diferentes camadas do sistema. Em relação ao pipeline, além de adicionar mais estágios com novos testes, a sugestão seria evoluir para um pipeline de implantação contínua.

REFERÊNCIAS

COHN, M. **Succeeding with Agile: Software Development Using Scrum**. Nova York, Estados Unidos: Addison-Wesley Professional, 2009. ISBN 9786612430565.

HUMBLE, J.; FARLEY, D.; KIM, G. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**. Nova York, Estados Unidos: Addison-Wesley Professional, 2010. ISBN 9781282852860.

NGUYEN, H. et al. **End-to-end testing on Web Experience Management Platform**. 2020, Espoo, Finlândia, 2020.

PRESSMAN, R. S. **Testing Computer Software**. Nova York, Estados Unidos: John Wiley Sons, 1998. ISBN 9780470311998.

____. **Software Engineering: A Practitioner's Approach**. Nova York, Estados Unidos: McGraw-Hill Education, 2013. ISBN 9780078022128.

SINGH, V. **Developing a CI/CD pipeline with GitLab**. 2022, Turku, Finlândia, 2022.

WOLF, J.; YOON, S. Automated testing for continuous delivery pipelines. In: **industrial talk**. Óregon, Estados Unidos: Pacific NW Software Quality Conference, 2016.

NUP: 23081.018030/2023-10 **Prioridade:** Normal

Homologação de ata de defesa de TCC e estágio de graduação

125.322 - Bancas examinadoras de TCC: indicação e atuação

COMPONENTE

Ordem	Descrição	Nome do arquivo
4	Trabalho de conclusão de curso (TCC) (125.32)	tcc_vinicius_bohrer_final_final.pdf

Assinaturas

24/02/2023 09:19:42

VINICIUS BOHRER DOS SANTOS (Aluno de Graduação)
07.09.09.01.0.0 - Curso de Engenharia de Computação - 121624