UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS SOBRAL

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

IZAIAS MACHADO PESSOA NETO

FERRAMENTAS PARA APOIAR O DESENVOLVIMENTO WEB: UM ESTUDO EXPLORATÓRIO

SOBRAL

2024

IZAIAS MACHADO PESSOA NETO

FERRAMENTAS PARA APOIAR O DESENVOLVIMENTO WEB: UM ESTUDO EXPLORATÓRIO

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

Orientador: Prof. Dr. Fischer Jônatas Ferreira

Coorientador: Prof. Dr. Evilásio Costa Júnior

SOBRAL

2024

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)

P567f Pessoa Neto, Izaias.
Ferramentas para apoiar o desenvolvimento Web : Um estudo exploratório / Izaias Pessoa Neto. – 2024.
98 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2024.
Orientação: Prof. Dr. Fischer Jônatas Ferraira.
Coorientação: Prof. Dr. Evilásio Costa Júnior.

1. Ferramentas Web. 2. Desenvolvimento Web. 3. Entrevistas com profissionais. I. Título.
CDD 621.39

IZAIAS MACHADO PESSOA NETO

FERRAMENTAS PARA APOIAR O DESENVOLVIMENTO WEB: UM ESTUDO EXPLORATÓRIO

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

Aprovada em: 24 de Setembro de 2024

BANCA EXAMINADORA

Prof. Dr. Fischer Jônatas Ferreira (Orientador)
Universidade Federal de Itajubá (UNIFEI)

Prof. Dr. Evilásio Costa Júnior (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Iális Cavalcante de Paula Júnior
Universidade Federal do Ceará (UFC)

Prof. Dr. Lucas Antônio de Oliveira
Universidade Federal de Itajubá (UNIFEI)

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela força, sabedoria e oportunidades que me permitiram chegar até aqui.

Em seguida, agradeço aos meus pais, Jean Marcos Fernandes Leite e Fátima Regina Machado Carneiro Leite, por todos os sacrifícios e por sempre terem proporcionado o melhor para que eu pudesse alcançar meus objetivos. A força e o apoio de vocês foram essenciais para que eu chegasse até aqui.

Aos meus irmãos, Iago Machado Carneiro Leite e Jean Marcos Fernandes Leite Filho, pela inspiração constante, pelas palavras de incentivo e pelo apoio diário, meu sincero obrigado.

Minha gratidão se estende ao Programa de Educação Tutorial (PET) do curso de Engenharia de Computação da UFC – Campus Sobral, bem como ao Programa de Capacidades Analíticas da UFMG, pelo auxílio financeiro e aprendizados adquiridos durante minha formação.

Um agradecimento especial ao meu orientador, o professor Dr. Fischer Jônatas Ferreira, pelo incentivo contínuo e pelas oportunidades que me proporcionou durante minha jornada acadêmica.

Ao meu coorientador, professor Dr. Evilásio Costa Júnior, sou grato pelo suporte, valiosas contribuições e o olhar atento no desenvolvimento deste trabalho.

Sou também grato aos membros da banca, Dr. Iális Cavalcante de Paula Júnior e Dr. Lucas Antônio de Oliveira, pela disponibilidade, colaborações e por aceitarem participar deste momento importante da minha trajetória acadêmica.

Agradeço também aos professores do curso de Engenharia de Computação da UFC – Campus Sobral, pelo conhecimento compartilhado e por todo o suporte ao longo de minha graduação.

Aos meus amigos de graduação, William Bruno Sales de Paula Lima e Marcos Vinícius Andrade de Sousa, agradeço as contribuições prestadas, amizade e o incentivo ao longo dessa jornada.

A todos que, de alguma forma, colaboraram para o desenvolvimento deste trabalho, deixo aqui os meus sinceros agradecimentos.

RESUMO

CONTEXTO: Com o surgimento contínuo de novas opções de tecnologias, o desenvolvimento de aplicações Web modernas se torna cada vez mais complexo. No entanto, para auxiliar nesse processo, existem ferramentas disponíveis que oferecem suporte aos desenvolvedores. Essas ferramentas têm como objetivo acelerar o desenvolvimento, melhorar a qualidade do código e automatizar processos. Além disso, algumas dessas ferramentas são aplicáveis a projetos Web independente do que está sendo desenvolvido, pois atendem à necessidades comuns encontradas na construção dessas aplicações. MOTIVAÇÃO: Contudo, até o momento, não foram identificados estudos recentes na literatura que se dediquem a caracterizar e oferecer orientações para a seleção de ferramentas que apoiam o desenvolvimento de aplicações Web. OBJETIVO: Nesse sentido, este estudo busca descrever as principais características dessas ferramentas. Isto inclui fornecer informações detalhadas sobre como cada ferramenta pode ser utilizada no processo de desenvolvimento Web. METODOLOGIA: A fim de alcançar o objetivo proposto, o estudo utilizará entrevistas com profissionais experientes no mercado, em diferentes níveis profissionais. Essas entrevistas serão usados para identificar e destacar características importantes na escolha das ferramentas que apoiam o desenvolvimento Web. As percepções coletadas por meio das entrevistas possibilitarão a extração de lições relevantes sobre algumas das principais ferramentas que auxiliam no desenvolvimento Web. BENEFICIADOS: Os resultados deste estudo serão úteis tanto para estudantes quanto para profissionais iniciantes, permitindo que conheçam as ferramentas e necessidades comuns em projetos Web. Desenvolvedores experientes também poderão se beneficiar das lições aprendidas, possibilitando uma tomada de decisão mais assertiva na escolha de ferramentas que apoiam o desenvolvimento Web. Em suma, o estudo fornecerá um conhecimento relevante para melhorar as habilidades e a tomada de decisão dos desenvolvedores.

Palavras-chave: ferramentas; desenvolvimento; web; entrevistas; comunicação; controle de versão; integração contínua; entrega contínua; software

ABSTRACT

CONTEXT: With the continuous launch of new options for Web development, the process of developing modern applications is becoming increasingly complex. However, there are tools available to assist developers in speeding up code delivery, deployment, and improving code quality. Furthermore, some of these tools can be applied to Web applications regardless of the specific use case, as they address common needs encountered in Web development. MOTIVATION: Nonetheless, recent literature lacks studies that characterize and offer guidance for the selection of tools that support the development of Web applications. OBJECTIVE: In this study, we aim to describe the main characteristics of these tools. This includes providing detailed information about each tool and explaining how it can be used in the Web development process. METHODOLOGY: To achieve the proposed objective, the study will do interviews with experienced professionals, at different professional levels. These interviews will be used to identify and highlight important features for choosing these tools. The insights collected through the interviews will help to highlight relevant information about each of choosen tools. BENEFITS: The findings of this study will provide valuable insights for students and professionals, including those seeking or already working in entry-level positions. It will enable them to familiarize themselves with the tools and common requirements in Web projects. Even experienced developers can benefit from this study, as it offers relevant knowledge to enhance their decision-making skills when selecting new tools for project development.

Keywords: tools; development; web; interviews; communication; version control; continuous integration; continuous delivery; software

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O avanço da internet, impulsionado pela popularização de dispositivos móveis e a melhoria da infraestrutura de rede, tem provocado uma mudança significativa no comportamento das pessoas. Essa transformação impactou diretamente a forma como elas consomem informações, se comunicam e interagem (BAHRINI; QAFFAS, 2019). Com isso, surgiu uma demanda por aplicações cada vez mais avançadas e que fossem capazes de atender às necessidades das pessoas em diferentes dispositivos (LAZAR, 2001).

Para enfrentar esses desafios, surgem novas tecnologias e ferramentas específicas para lidar com essas demandas emergentes (TKACHEV, 2023). Por conseguinte, passa a ser cada vez mais necessário utilizar soluções que permitam aos desenvolvedores aproveitarem o conhecimento acumulado e evitar o retrabalho ao construir aplicações. Nesse contexto, padrões, bibliotecas e *frameworks* se tornam aliados indispensáveis no desenvolvimento de aplicações Web (VALENTE, 2022).

No âmbito do desenvolvimento Web, existem ferramentas que oferecem suporte aos profissionais durante o processo de construção e implantação de aplicações. Essas ferramentas são voltadas para os desenvolvedores e proporcionam recursos e funcionalidades que visam facilitar o trabalho e aumentar a eficiência no desenvolvimento Web. Elas ajudam os profissionais a lidar com a complexidade desse tipo de projeto e a atender às expectativas dos usuários (RAJENDRAN; VEILUMUTHU, 2010; FRATERNALI, 1999).

Diante desse cenário, essas ferramentas de desenvolvimento Web são aplicáveis a diversos tipos de projetos, independentemente do domínio do que está sendo desenvolvido. Isso ocorre porque elas atendem a demandas comuns encontradas na construção de aplicações Web. Ao utilizar essas ferramentas, os desenvolvedores podem agilizar e aprimorar o processo de criação dessas aplicações. É importante ressaltar que essas ferramentas de apoio ao desenvolvimento Web não devem ser confundidas com *frameworks* ou bibliotecas, pois possuem um escopo mais amplo, abrangendo áreas como versionamento de código, comunicação, colaboração, hospedagem de código e implantação de aplicações (ROSSON *et al.*, 2004).

Apesar da importância das ferramentas que apoiam o desenvolvimento Web, até o momento não existem estudos recentes na literatura que descrevam e ofereçam orientações para escolher as ferramentas adequadas para um projeto. Embora haja uma ampla variedade de opções disponíveis, a falta de diretrizes claras representa um desafio para os profissionais e estudantes que buscam escolher as ferramentas mais adequadas para suas necessidades no

desenvolvimento de aplicações Web.

Dado a importância das ferramentas que apoiam o desenvolvimento Web e também a lacuna existente na literatura para auxiliar na escolha dessas tecnologias, este estudo tem como principal objetivo descrever as principais características dessas ferramentas. Além disso, busca-se fornecer informações detalhadas sobre como que cada ferramenta pode ser utilizada no processo de desenvolvimento Web. Isto inclui oferecer orientações que auxiliem desenvolvedores na escolha dessas ferramentas.

A fim de alcançar o objetivo proposto, este estudo utilizará uma abordagem metodológica que envolve a realização de entrevistas com profissionais experientes (CONSTANTINO *et al.*, 2023; CONSTANTINO *et al.*, 2020; TZAFILKOU *et al.*, 2020). Essa forma de coleta de dados será utilizada para identificar percepções importantes na escolha das ferramentas que apoiam o desenvolvimento Web. As percepções coletadas a partir dos resultados serão organizadas como de lições aprendidas.

Com base nessa análise, serão documentadas informações relevantes para a escolha das ferramentas que apoiam o desenvolvimento Web. Isso inclui características, vantagens e desvantagens específicas das ferramenta, bem como diretrizes que podem orientar na sua seleção. Ao final desse processo, é esperado um conjunto de informações que podem auxiliar os desenvolvedores na escolha apropriada dessas tecnologias.

Os principais beneficiados com os resultados desse estudo são os profissionais e estudantes envolvidos no desenvolvimento Web. Com as informações documentadas ao fim desse estudo, os profissionais e estudantes serão capazes de tomar decisões informadas ao selecionar as ferramentas adequadas para seu projeto. Por trazer características de cada ferramenta, o estudo também traz consigo as necessidades que são comuns em projetos Web. Isto é especialmente relevante para estudantes e profissionais que estão ingressando no mercado de trabalho.

O Capítulo 2 aborda a fundamentação teórica, que consiste em uma introdução sobre as ferramentas que apoiam o desenvolvimento Web, apresentando exemplos de algumas dessas ferramentas. No Capítulo 3, é descrita a metodologia utilizada para a realização das entrevistas, incluindo o planejamento e a execução do estudo. O Capítulo 4 discute os resultados das entrevistas, analisando as ferramentas citadas e sua relevância no contexto do desenvolvimento Web. No Capítulo 5, são apresentados os trabalhos relacionados. Por fim, no Capítulo 6, são apresentadas as conclusões deste estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção tem como objetivo apresentar as ferramentas Web e explorar suas funcionalidades. Inicialmente é definido o escopo das ferramentas que apoiam o desenvolvimento Web (Seção 2.1). Com isso, nas seções seguintes é apresentado um contexto das principais ferramentas e também um exemplo prático. As outras seções que compõem o capítulo discutem: controle de versão com Git (Seção 2.2), compartilhamento e gerenciamento de código com GitHub (Seção 2.3), virtualização de aplicações com Docker (Seção 2.4) e, por fim, gerenciamento de identidade e controle de acesso com WSO2 Identity Server (Seção 2.5).

2.1 Ferramentas que apoiam o desenvolvimento Web

Uma característica importante da Web é o desacoplamento entre *front-end* e *back-end*. Ferramentas Web são utilizadas para construir tanto aplicações *front-ends*, quanto *back-ends*. O *front-end* contempla os sistemas de *software* que executam no navegador do cliente. Já as tecnologias que são executadas no lado do servidor da aplicação são chamadas de *back-end*. O *front-end* faz requisições por meio de chamadas Hypertext Transfer Protocol (HTTP) e como resposta o servidor (*back-end*) retorna o resultado de um processamento (MOZILLA, 2021).

Essa separação entre *front-end* e *back-end* permite que cada camada seja desenvolvida de forma independente, o que facilita a colaboração entre equipes especializadas em cada área. Esse desacoplamento traz vantagens como flexibilidade na escolha das tecnologias mais adequadas para cada camada, permitindo o uso das melhores soluções disponíveis.

Por existirem necessidades em comum na construção de quaisquer aplicações Web, existem ferramentas que podem ser empregadas independentemente do domínio da aplicação desenvolvida. Algumas dessas ferramentas podem oferecer funcionalidades essenciais, como controle de versão, compartilhamento de código, documentação, portabilidade da aplicação e gerenciamento de identidade. Essas são algumas funcionalidades que são necessárias no contexto do desenvolvimento Web. Ao utilizar essas ferramentas, os desenvolvedores podem aprimorar o processo de desenvolvimento, colaboração e implantação de aplicações Web, independentemente do tipo de projeto em que estejam trabalhando.

Para apoiar a comunicação e a colaboração eficazes entre as equipes, existem ferramentas, como o Git e o GitHub. O Git é um sistema de controle de versão distribuído que permite o rastreamento das alterações no código-fonte ao longo do tempo. Ele também permite que

os desenvolvedores trabalhem em paralelo na construção de novas funcionalidades e correções de *bugs* e, posteriormente, integrem suas alterações ao código principal. Já o GitHub, por sua vez, é uma plataforma de hospedagem de código que utiliza o Git como base. Ele oferece recursos adicionais, como o gerenciamento de *bugs*, solicitações de novas funcionalidades e também facilita a revisão e discussão do código entre os membros da equipe. Isso permite que os desenvolvedores colaborem em projetos, revisem alterações propostas e resolvam problemas de forma centralizada.

Além de controle de versão e comunicação, o desenvolvimento e implantação de aplicações Web podem se beneficiar do uso do Docker. Essa ferramenta permite a criação de ambientes consistentes e portáteis, chamados *containers*, encapsulando as dependências e configurações necessárias para a execução de uma aplicação. Com o Docker, é possível garantir que a aplicação funcione de maneira consistente em diferentes máquinas e sistemas operacionais, facilitando a colaboração entre equipes e simplificando o processo de implantação do código (DOCKER, 2020).

Por fim, temos ferramentas que auxiliam no controle de acesso, o WSO2 Identity Server é uma ferramenta que fornece recursos de segurança, como autenticação e autorização, além de oferecer suporte a protocolos de login único. Com essa ferramenta, é possível garantir a proteção das aplicações e o controle de acesso adequado aos recursos. É importante ressaltar que o uso de uma ferramenta centralizada de login permite que um mesmo login seja usado em todas as aplicações de uma organização.

2.2 Controle de versão com Git

Os Sistemas de Controle de Versão (SCV) têm a finalidade de gerenciar as alterações realizadas ao longo das várias versões de um projeto, permitindo, caso necessário, reverter mudanças feitas em um ou mais arquivos (CHACON; STRAUB, 2014). Essa capacidade de controle é obtida por meio de um banco de dados de versão, que registra todas as modificações realizadas nos arquivos ao longo do tempo. Para o Git, após cada *commit* é criada uma nova versão, que tem um identificador único. Nesse sentido, é possível visualizar as versões por meio de um histórico, o que permite rastrear as modificações realizadas. Além disso, o Git facilita a colaboração entre os membros da equipe, permitindo que cada pessoa trabalhe em sua própria versão do projeto e, posteriormente, as versões desenvolvidas são combinadas. Essa abordagem torna mais fácil rastrear quem fez cada alteração e também facilita que várias pessoas trabalhem

modificando um mesmo arquivo.

Duas entre as diferentes abordagens de versionamento de código incluem o versionamento local e o versionamento centralizado. O versionamento local é um processo de controle de versão realizado diretamente na máquina do desenvolvedor. Nessa abordagem, as modificações são feitas no diretório de trabalho, que é a pasta do projeto local, sendo rastreadas por um banco de dados de versão local. Isso permite que o desenvolvedor gerencie e acompanhe as alterações feitas no código-fonte sem depender do acesso a um servidor remoto. Dessa forma, o versionamento local oferece mais autonomia e flexibilidade ao desenvolvedor durante o processo de controle de versão. Já no versionamento centralizado, é utilizado um servidor remoto para compartilhar as versões do projeto entre os desenvolvedores. Entretanto, o ponto negativo desses sistemas é que por serem centralizados, falhas no servidor podem acarretar na ociosidade do time de desenvolvimento, ou até a perda do histórico de mudanças.

Por outro lado, o Git se destaca, pois é um sistema de controle de versão distribuído, que consiste em uma abordagem flexível que combina as vantagens do versionamento local e do centralizado. No Git, cada desenvolvedor possui uma cópia completa do repositório em sua máquina. Isso permite que trabalhem de forma independente, cada um com um histórico local de versões. O Git também permite a sincronização e compartilhamento entre versões de diferentes colaboradores por meio de um repositório remoto, onde as modificações de diferentes desenvolvedores são mescladas. Como o repositório local é independente do repositório remoto, caso o servidor de Git fique indisponível os desenvolvedores conseguem continuar trabalhando.

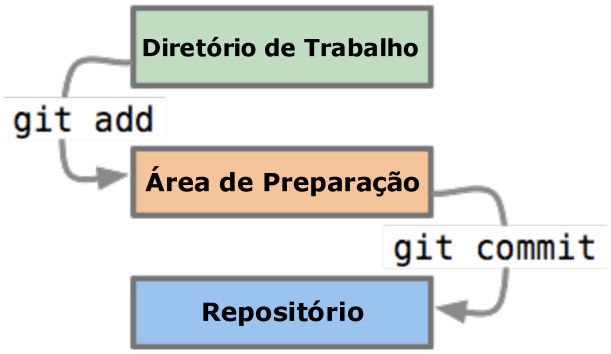
No contexto do Git, plataformas como GitHub[1], GitLab[2] e BitBucket[3] desempenham um papel fundamental no desenvolvimento colaborativo. Além de fornecerem servidores para hospedar repositórios Git, essas plataformas oferecem recursos adicionais que facilitam o trabalho em equipe. Elas fornecem recursos avançados para revisão de código, discussão de problemas, gerenciamento de tarefas, planejamento de projetos e documentação. Além disso, essas plataformas permitem que os repositórios de código aberto sejam disponibilizados para uso e colaboração.

Em um repositório Git, as modificações são realizadas de forma preliminar no diretório de trabalho. Para cada modificação que se deseja criar uma nova versão, é necessário primeiro adicionar os arquivos modificados a uma área de preparação (*staging*). Com isso, é

[1] https://github.com
[2] https://about.gitlab.com
[3] https://bitbucket.org

possível realizar um *commit*, que na prática, faz com que os arquivos da área de preparação façam parte de uma nova versão do repositório. Caso haja alguma modificação no diretório de trabalho que não foi adicionada a área de preparação, essa modificação não é propagada para a nova versão. A Figura 1 mostra que as modificações feitas no diretório de trabalho são enviadas para a área de preparação por meio do comando *git add*. Por conseguinte, as modificações da área de preparação são salvas em uma nova versão do repositório por meio do comando *git commit*.

Figura 1 – Propagação das modificações



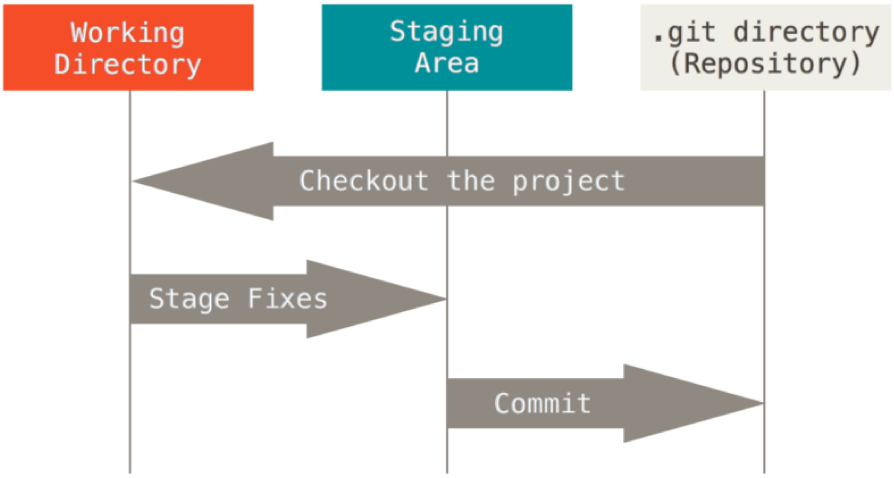
Fonte: Adaptado de Chacon (2009)

O Git é um SCV em que o banco de dados de versão é armazenado em uma pasta especial na raiz do projeto, chamada `.git`. Essa pasta contém todas as informações necessárias para rastrear as versões, como por exemplo, o histórico de alterações do projeto e informações sobre o autor de cada modificação. Dessa forma, mesmo em casos de perda dos arquivos do projeto, desde que a pasta `.git` seja preservada, é possível recuperar versões anteriores dos arquivos.

Nesse contexto, a Figura 2 apresenta as três principais seções de um projeto Git: o diretório de trabalho chamado na imagem de *working directory*, área de preparação chamada de *staging area* e a pasta `.git` que é o repositório que contém os arquivos de versionamento. A imagem contém setas que ilustram as ações realizadas para alterar a área do projeto em que os arquivos se encontram. Partindo do diretório de trabalho, as modificações são movidas para a área de preparação e, em seguida, é feito o *commit*, que registra as modificações em preparação como parte de uma nova versão. Além disso, a Figura 2 também ilustra que é possível trazer uma versão do repositório para o diretório de trabalho. Isso pode ser feito porque cada *commit* possui um identificador único, o que permite escolher uma versão anterior do projeto e trazer para o diretório de trabalho.

Foram abordados os estados de um projeto Git na perspectiva de versões, mas o

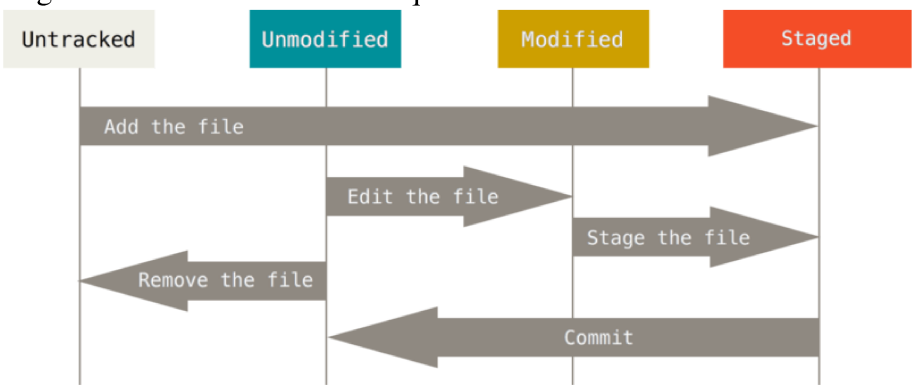
Figura 2 – Seções de um projeto Git



Fonte: Chacon e Straub (2014)

ciclo de vida dos arquivos também é um conceito básico desse sistema de versionamento. Nesse sentido, a Figura 3 ilustra que ao criar um novo arquivo no diretório de trabalho ele tem seu *status* como *untracked*, ou seja, não rastreado, porque até o momento o Git não está configurado para rastrear modificações nesse arquivo. Após enviar o arquivo para área de preparação, esse arquivo será marcado como *staged* e depois do *commit* é marcado como não modificado, ou seja, *unmodified*. Isto acontece porque o conteúdo do arquivo é o mesmo da versão mais recente, porque a modificação foi enviada para o repositório, portanto não há diferença entre o conteúdo do arquivo e da versão mais atual. Com isso, ao modificar o conteúdo do arquivo ele passa a ter estado *modified* e ao realizar o mesmo procedimento de *commit* feito para o arquivo criado, volta novamente a ter *status unmodified*.

Figura 3 – Ciclo de vida dos arquivos



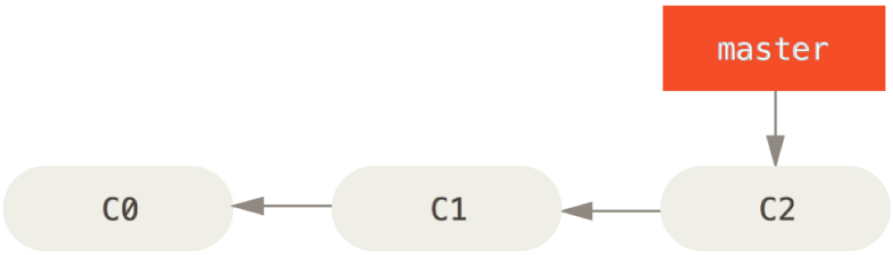
Fonte: Chacon e Straub (2014)

O *HEAD* é uma referência importante no controle de versão do Git. Ele aponta para um *commit* específico, marcando a posição atual do histórico de versões. Desse modo, as

versões dos arquivos presentes no *commit* que essa referência aponta serão as mesmas dos que estão no diretório de trabalho. Essa referência permite comparar as alterações em cada arquivo e determinar se um arquivo está sendo rastreado ou não, bem como se possui modificações em relação à versão atual no histórico de versões. O comando `git checkout` é utilizado para alterar a referência do *HEAD*, possibilitando a transição entre diferentes versões de um repositório.

Ao criar um projeto Git, automaticamente é criado uma *branch* com o nome "master" em que os *commits* são realizados. Cada *commit* aponta para um *commit* anterior, conforme ilustrado na Figura 4. Nesse sentido, o histórico de *commits* da imagem é formado por C0, C1 e C2. Já uma *branch*, consiste em uma referência que aponta para um *commit* específico, assim como o *HEAD*. No caso da imagem, a *branch* "master" é uma referência que aponta para C2.

Figura 4 – Ilustração de um histórico de *commits* linear

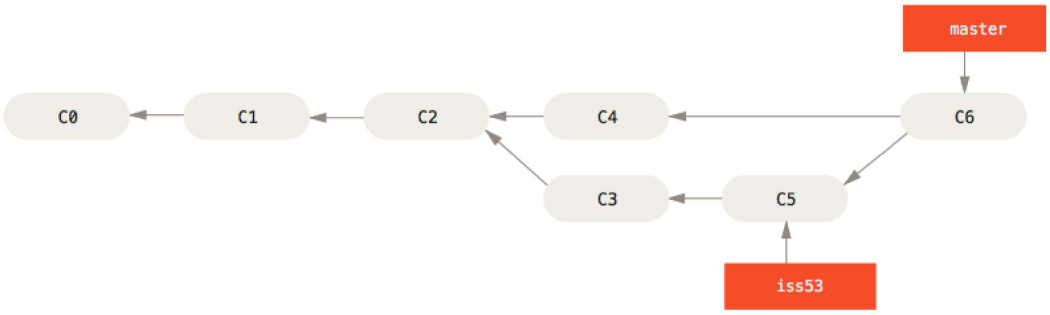


Fonte: Chacon e Straub (2014)

Quando dois ou mais *commits* têm um mesmo *commit* como ponto de partida, isso é conhecido como *branching*, o que resulta na criação de um novo ramo no histórico de versões do projeto. Essa situação cria diferentes linhas de desenvolvimento de um mesmo projeto. A abordagem facilita o desenvolvimento de recursos independentes e permite que várias alterações sejam realizadas simultaneamente sem interferir no código principal do projeto. Cada ramificação representa uma sequência única de *commits* que registram as modificações específicas feitas nessa linha de desenvolvimento.

No contexto da Figura 5, a partir de C2 é criada uma *branch iss53*, na qual são realizados os *commits* C3 e C5, de forma paralela, é feito o *commit* C4 na "master". O *commit* C6 é denominado *merge commit*, em que as modificações realizadas em *iss53* são mescladas com o conteúdo da "master". Ocasionalmente, o processo de *merge* não é realizado de forma automática devido a conflitos com relação ao conteúdo dos arquivos. Isso ocorre quando são feitas alterações nas mesmas partes de um arquivo em cada uma das *branches*. Neste caso, o *merge* é interrompido para que possam ser resolvidos os conflitos de forma manual, por meio da modificação do conteúdo dos arquivos.

Figura 5 – Ilustração de um histórico de *commits* com *branches* e *merges*



Fonte: Chacon e Straub (2014)

2.2.1 Comandos básicos

Os comandos apresentados Figura 6, são `git init`, `ls` com o parâmetro `-a` e `git status`. O comando `git init` é usado para inicializar um novo repositório Git em uma máquina. Já o `ls` tem o intuito de mostrar quais arquivos e pastas pertencem à pasta atual. Os arquivos e pastas que tem um ponto no início do nome, como a pasta `.git`, não são mostrados por padrão pelo comando `ls`, por isso é necessário utilizar o parâmetro adicional `-a`. Por fim, o comando `git status` mostra qual o estado dos arquivos e pastas em relação ao repositório, conforme explicado na discussão do ciclo de vida dos arquivos.

Nesse sentido, na Figura 6, após a inicialização do repositório, uma mensagem informando a criação do repositório é exibida. Ao listar os arquivos e pastas, observa-se a presença da pasta `.git` e do arquivo "index.html". Ao verificar o estado dos arquivos do repositório, é apresentado que o arquivo "index.html" está no estado *untracked*, ou seja, não está sendo rastreado pelo Git.

Figura 6 – Inicialização do projeto Git

```
→  git-example git init
Initialized empty Git repository in /Users/izaias/dev/tcc/git-example/.git/
→  git-example git:(main) ✗ ls -a
.           ..          .git        index.html
→  git-example git:(main) ✗ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)
```

Fonte: Próprio autor

Para que as modificações em um arquivo passem a ser rastreadas pelo Git é necessário que ele faça parte de uma versão do repositório. O comando `git add`, usado na Figura 7, é utilizado para adicionar arquivos e pastas na área de preparação, dado um caminho. Após adicionar o arquivo na área de preparação é apresentado pelo comando `git status` que o estado de "index.html" é como um novo arquivo. Com isso, é necessário realizar um *commit* para que as modificações presentes na área de preparação passem a fazer parte de uma nova versão do repositório de código.

Na Figura 7, após realizar o *commit* do conteúdo da área de preparação, com o comando `git commit`, e utilizar novamente o comando `git status`, não há mais modificações para serem incluídas em um novo *commit*. Essa informação é indicada pela mensagem do Git após a execução do último comando. Isso ocorre porque o arquivo "index.html" agora está sendo rastreado pelo Git e seu conteúdo é idêntico ao conteúdo presente no repositório. Portanto, o arquivo está em sincronia com a última versão do repositório e não há mais alterações a serem registradas, conforme esperado.

O comando utilizado para realizar o *commit* na Figura 7 possui um parâmetro `-m`, referente a uma mensagem utilizada para descrever as alterações realizadas. Caso a mensagem seja confusa ou não identifique o propósito do *commit*, isso pode atrapalhar a equipe de desenvolvimento, especialmente se for necessário encontrar um *commit* após algum tempo (GITKRAKEN, 2023). Com relação a quantidade de alterações por *commit*, é necessário que cada modificação possa ser descrita claramente por meio do parâmetro de mensagem, que tem tamanho máximo de 72 caracteres. Por conseguinte, é recomendado fazer mais de um *commit* se as modificações realizadas não possam ser descritas dado a restrição de tamanho da mensagem.

Figura 7 – *Commit* do arquivo *index.html*

```
→  git-example git:(main) ✗ git add index.html
→  git-example git:(main) ✗ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html

→  git-example git:(main) ✗ git commit -m "adicionar arquivo html"
[main (root-commit) b6b45b9] adicionar arquivo html
 1 file changed, 10 insertions(+)
 create mode 100644 index.html
→  git-example git:(main) git status
On branch main
nothing to commit, working_tree clean
```

Fonte: Próprio autor

O comando apresentado `git log` é utilizado para visualizar o histórico de *commits* de um repositório. A Figura 8 apresenta o resultado do uso desse comando, exibindo informações relevantes sobre cada *commit*. Para cada *commit*, é listado um identificador único, autor do *commit*, data em que foi realizado, descrição da modificação e também a indicação de qual *commit* é a referência *HEAD*. Essas informações são essenciais para acompanhar o progresso no desenvolvimento, entender as alterações realizadas ao longo do tempo e facilitar a colaboração entre os membros da equipe.

Figura 8 – Histórico de modificações do repositório

```
commit b6b45b91b5fcc96bde9eccc15879bd152f58d878 (HEAD -> main)
Author: Izaias Machado <izaiasmachado.dev@gmail.com>
Date:   Mon Jun 5 19:08:53 2023 -0300

    adicionar arquivo html
(END)
```

Fonte: Próprio autor

Quando um arquivo existente no repositório é modificado no diretório de trabalho, o Git é capaz de identificar e registrar as alterações específicas em relação à versão anterior. Isso significa que o Git pode rastrear as modificações feitas no conteúdo dos arquivos ao longo do tempo. Por outro lado, novas pastas e arquivos são elementos que não existiam nas versões anteriores do repositório. Ao serem adicionados pela primeira vez ao diretório de trabalho, esses elementos não têm histórico de versões anteriores, por isso, o Git os trata como novos arquivos a serem rastreados (*untracked*). Dessa forma, o Git diferencia entre modificações de arquivos existentes e arquivos não rastreados, permitindo um controle mais preciso das alterações realizadas no projeto.

Nesse contexto, para demonstrar a capacidade de rastrear modificações em arquivos e também rastrear a criação de novos arquivos e pastas, foram feitas alterações no diretório de trabalho do exemplo anterior. Inicialmente, o arquivo “index.html”, que já foi incluído em um *commit*, teve seu conteúdo modificado. Além disso, foi criada uma pasta chamada “assets”, que contém dois arquivos: “script.js” e “style.css”, com o intuito de destacar que o Git é também capaz de identificar novas pastas. Com essas modificações realizadas no diretório de trabalho, na Figura 9 é verificado o *status*. O resultado mostra que o arquivo “index.html” tem modificações de conteúdo em relação à sua última versão. Além disso, é mostrado que passa a existir uma pasta não rastreada com nome “assets”. Com o comando `tree` (que não é um comando do Git), é mostrado a estrutura de arquivos do diretório e que a pasta não rastreada tem os arquivos “script.js” e “style.css”.

Figura 9 – Restauração do diretório de trabalho - Parte 01

```
→  git-example git:(main) ✗ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        assets/

no changes added to commit (use "git add" and/or "git commit -a")
→  git-example git:(main) ✗ tree
.
├── assets
│   ├── script.js
│   └── style.css
└── index.html

1 directory, 3 files
```

Fonte: Próprio autor

O comando `git restore` é utilizado para restaurar o conteúdo de um ou mais arquivos para a versão do *commit* referenciado pelo *HEAD*. Na Figura 10, esse comando é utilizado passando o caminho dos arquivos a serem restaurados. Adicionalmente, poderia ser passado o caminho para uma pasta ou arquivo específico, de modo a restaurar somente o conteúdo do caminho. Após restaurar os arquivos da pasta e verificar o *status*, é constatado que o arquivo "index.html" teve seu conteúdo restaurado, pois deixa de ser *modified*. Porém, a pasta "assets" continua no diretório de trabalho.

Considerando esse contexto, é necessário utilizar o comando `git clean`, para que os arquivos e pastas não rastreados pelo Git sejam removidos. O uso desse comando é exemplificado na Figura 10. Com isso, após a execução da diretiva `clean` é verificado o *status* novamente e o resultado é que não há nenhum *commit* que possa ser realizado, conforme apresentado pela Figura 10. Isso acontece porque a pasta "assets" foi removida, junto dos arquivos "script.js" e "style.css".

Figura 10 – Restauração do diretório de trabalho - Parte 02

```
→  git-example git:(main) ✗ git restore .
→  git-example git:(main) ✗ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        assets/

nothing added to commit but untracked files present (use "git add" to track)
→  git-example git:(main) ✗ git clean -f -d
Removing assets/
→  git-example git:(main) git status
On branch main
nothing to commit, working_tree clean
```

Fonte: Próprio autor

2.2.2 Comandos para utilização de branches

Para listar todas as *branches* no repositório local pode ser utilizado o comando `git branch`. O uso desse comando é exemplificado na Figura 11, o resultado lista todas as *branches*, mas também a *branch* atual é marcada em verde e com um asterisco a esquerda de seu nome. Ainda na Figura 11, a fim de criar uma nova *branch* é usado o comando `git checkout` com o parâmetro `-b`, seguido de "traducao-portugues", que é o nome da ramificação. O comando `git branch` é utilizado novamente para verificar que a ramificação de tradução para português foi de fato criada. Além disso, a ramificação "traducao-portugues" passa a ser a *branch* atual.

Figura 11 – Criação e listagem de *branches*

```
[→  git-example git:(main) git branch
* main
[→  git-example git:(main) git checkout -b traducao-portugues
Switched to a new branch 'traducao-portugues'
[→  git-example git:(traducao-portugues) git branch
  main
* traducao-portugues
```

Fonte: Próprio autor

Nesse contexto, na Figura 12, são feitas modificações no arquivo "index.html" que são adicionadas a área de preparação e salvas por meio de um *commit*. O comando `git checkout` é utilizado sem o parâmetro adicional para que a *branch* atual passe a ser a "main". Mudar a *branch* principal consiste em alterar o *commit* referenciado pelo *HEAD*. Nesse sentido, na *branch* "main" também é feito um *commit* que modifica o arquivo "index.html". Por fim, é realizado um *merge commit* utilizando o comando `git merge` seguido do nome da *branch* que deseja fazer o *commit*.

O comando `git log` pode ser utilizado para visualizar o histórico de modificações, como exemplificado previamente. De forma adicional, com os parâmetros mostrados na Listagem 2.2.1 é possível visualizar o histórico de *commits* mostrando somente as informações relevantes sobre as alterações realizadas e em que *branches* elas foram feitas. O resultado produzido pode ser visto na Figura 13, em que é ilustrado que os *commits* para adicionar o corpo do texto e traduzir para português foram feitos de forma paralela e mesclados na *branch* "main" por um *merge commit*.

Ao finalizar as modificações de uma *branch*, a mesma pode ser deletada por meio do comando mostrado na Listagem 2.2.2. O comando `git branch` é o mesmo utilizado para listar as ramificações no repositório local, seguido de um parâmetro `-D` que é acompanhado do

Figura 12 – *Commits* e *merge* de *branches*

```
➜  git-example git:(traducao-portugues) git add index.html
➜  git-example git:(traducao-portugues) ✗ git commit -m "traduzir para portugues
"
[traducao-portugues f112357] traduzir para portugues
 1 file changed, 2 insertions(+), 2 deletions(-)
➜  git-example git:(traducao-portugues) git checkout main
Switched to branch 'main'
➜  git-example git:(main) git add index.html
➜  git-example git:(main) ✗ git commit -m "adiciona corpo do texto"
[main 65b3376] adiciona corpo do texto
 1 file changed, 4 insertions(+)
➜  git-example git:(main) git merge traducao-portugues
Auto-merging index.html
Merge made by the 'ort' strategy.
 index.html | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
```

Fonte: Próprio autor

```
1 git log --oneline --graph
```

Listagem 2.2.1 – Mostra histórico *commits* em linha única e *branches* em modo gráfico

Figura 13 – Histórico de *commits* em linha única e *branches* em modo gráfico

```
*   eec899e (HEAD -> main) Merge branch 'traducao-portugues'
|\
| * f112357 (traducao-portugues) traduzir para portugues
* | 65b3376 adiciona corpo do texto
|/
* ca36b7d adiciona titulo
* b6b45b9 adicionar arquivo html
(END)
```

Fonte: Próprio autor

nome da *branch* que se deseja remover. Assim como a listagem de *branches*, esse comando serve somente para remover ramificações do repositório local.

```
1 git branch -D traducao-portugues
```

Listagem 2.2.2 – Comando para deletar a *branch* de tradução para português

Antes dos comandos mostrados na Figura 14, foi feita uma *branch* de nome “muda-titulo” que possui um *commit* que muda o título do arquivo “index.html”. Em paralelo, na ramificação principal também foi modificado o título da página. Com isso, ao tentar realizar o *merge*, é reportado um conflito, assim como apresentado pela imagem. Ainda na Figura 14, é mostrado o conteúdo mesclado do arquivo “index.html”, presente nas *branches* “main” e “muda-titulo”, em que são adicionados delimitadores que demarcam os conflitos e a *branch* de origem de cada conteúdo.

Figura 14 – Erro de *merge* e conteúdo conflituoso dos arquivos

```
➜  git-example git:(main) git merge muda-titulo
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
➜  git-example git:(main) ✗ cat index.html
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Documento</title>
  </head>
  <body>
<<<<<<< HEAD
    <h1>Porquê Git? (main)</h1>
=======
    <h1>Título Novo Aqui (branch muda-titulo)</h1>
>>>>>>> muda-titulo
    <p>
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Quisquam
      voluptatibus, quibusdam, quia, quos voluptatem voluptatum quod
    </p>
  </body>
</html>
```

Fonte: Próprio autor

Por fim, para resolver os conflitos devem ser apagados os delimitadores e o conteúdo dos arquivos deve ser mesclado de forma manual pelo desenvolvedor. O conteúdo do arquivo resultante após a resolução é apresentado pela Figura 15. Além disso, é necessário adicionar as alterações na área de *staging* e realizar o *commit* das modificações, a fim de que o *merge* seja finalizado. Caso o desenvolvedor precise cancelar a resolução dos conflitos é possível utilizar o comando da Listagem 2.2.3.

Figura 15 – Resolução do conflito

```
➜  git-example git:(main) ✗ cat index.html
<!DOCTYPE html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Documento</title>
  </head>
  <body>
    <h1>Título Novo Aqui (main)</h1>
    <p>
      Lorem ipsum dolor sit amet consectetur adipisicing elit. Quisquam
      voluptatibus, quibusdam, quia, quos voluptatem voluptatum quod
    </p>
  </body>
</html>
➜  git-example git:(main) ✗ git add index.html
➜  git-example git:(main) ✗ git commit -m "fix conflicts"
[main 19e84ab] fix conflicts
```

Fonte: Próprio autor

```
1 git merge --abort
```

Listagem 2.2.3 – Abortar *merge commit* conflituoso

2.2.3 Uso de um servidor remoto de Git

Para importar um repositório já existente para servidores remotos de Git deve ser criado o repositório no site e copiado a URL do repositório, com isso o *link* encontrado é substituído no comando da primeira linha da Listagem 2.2.4. É possível verificar se a ação foi bem sucedida por meio do comando da segunda linha do mesmo Listing. Caso o repositório tenha sido criado diretamente no site que fornece um servidor remoto de Git, então é possível clonar o repositório remoto para a máquina local utilizando o comando `git clone`, exemplificado na Listagem 2.2.5.

```
1 git remote add origin <url-repositório>
2 git remote -v
```

Listagem 2.2.4 – Adicionar e verificar link do repositório remoto

```
1 git clone <url-repositório>
```

Listagem 2.2.5 – Clone de um repositório remoto

Nesse contexto, é possível utilizar o comando da primeira linha da Listagem 2.2.6 a fim de enviar os *commits* feitos no repositório local para uma *branch* correspondente no repositório remoto. Entretanto, caso tenha sido criada uma *branch* no repositório local, sem existir uma correspondente no repositório remoto, essa ação resulta em um erro. Para lidar com essa situação, é necessário utilizar o comando da segunda linha da Listagem 2.2.6 para criar a nova *branch* no repositório remoto. Deve-se ressaltar que as modificações feitas em *branches* remotas não são sincronizadas automaticamente com o repositório local. Portanto, é importante entender as formas de sincronização descritas a seguir.

```
1 git push
2 git push --set-upstream origin <nome-da-branch>
```

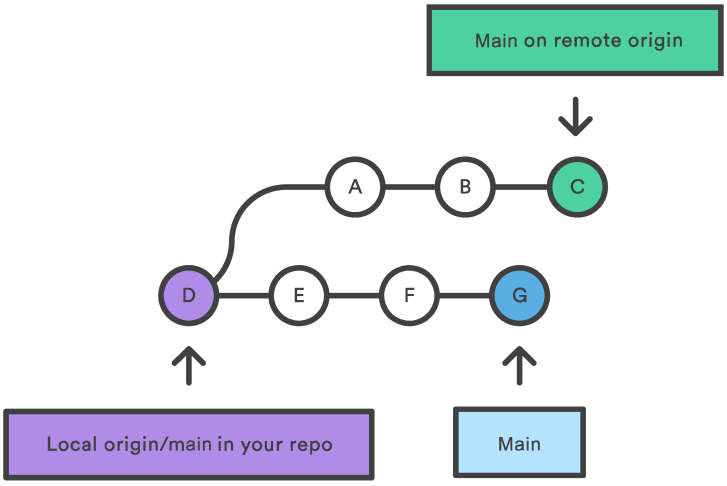
Listagem 2.2.6 – Enviando alterações para o repositório remoto

A Figura 16 ilustra uma ramificação em azul que corresponde a versão local da *branch* “main”, já em verde é mostrado a versão do repositório remoto da *branch* principal e em roxo está destacado o último *commit* em comum entre as ramificações. Nesse sentido, ao executar o comando `git fetch`, destacado na primeira linha da Listagem 2.2.7, são baixadas as modificações feitas no repositório remoto, mas isso não modifica a *branch* ou o *HEAD* do repositório local. Por conseguinte, para que as duas ramificações sejam mescladas é necessário realizar um *merge commit* com o comando `git merge` entre as *branches*, conforme explicado na seção anterior.

```
1 git fetch
2 git pull
```

Listagem 2.2.7 – Baixando alterações do repositório remoto

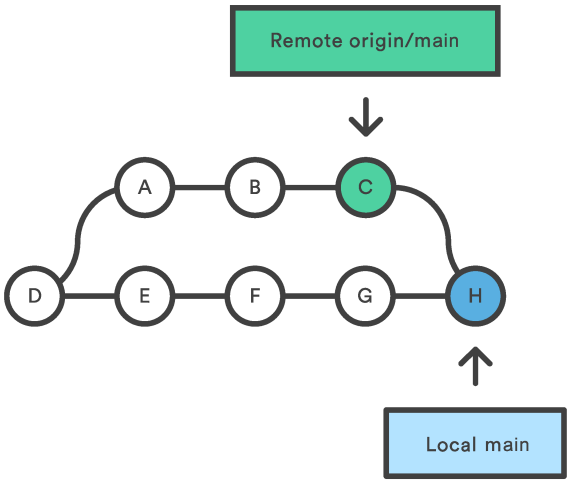
Figura 16 – Resultado após realizar um *fetch*



Fonte: Atlassian (2023)

Na Figura 17 é ilustrado o resultado do comando `git pull`, mostrado na segunda linha da Listagem 2.2.7, que consiste em um `git fetch` seguido do `git merge`. Entretanto, é importante ressaltar que o uso desse comando, assim como os outros *merge commits*, está sujeito a conflitos de *merge* que precisam ser resolvidos manualmente.

Figura 17 – Resultado após realizar um *pull* ou *fetch* seguido de *merge*



Fonte: Atlassian (2023)

2.3 Compartilhamento e gerenciamento de código com GitHub

Além de servir como um servidor remoto para hospedar repositórios com o sistema de controle de versão Git, o GitHub é um ambiente que fornece ferramentas que visam facilitar o trabalho em colaboração com outras pessoas (BEER, 2018). Nesse sentido, é fornecido uma interface Web amigável que permite visualizar, discutir e revisar modificações. É importante citar que além de colaboração em projetos pessoais ou de empresas, o GitHub também tem funcionalidades que viabilizam a contribuição para projetos de código aberto.

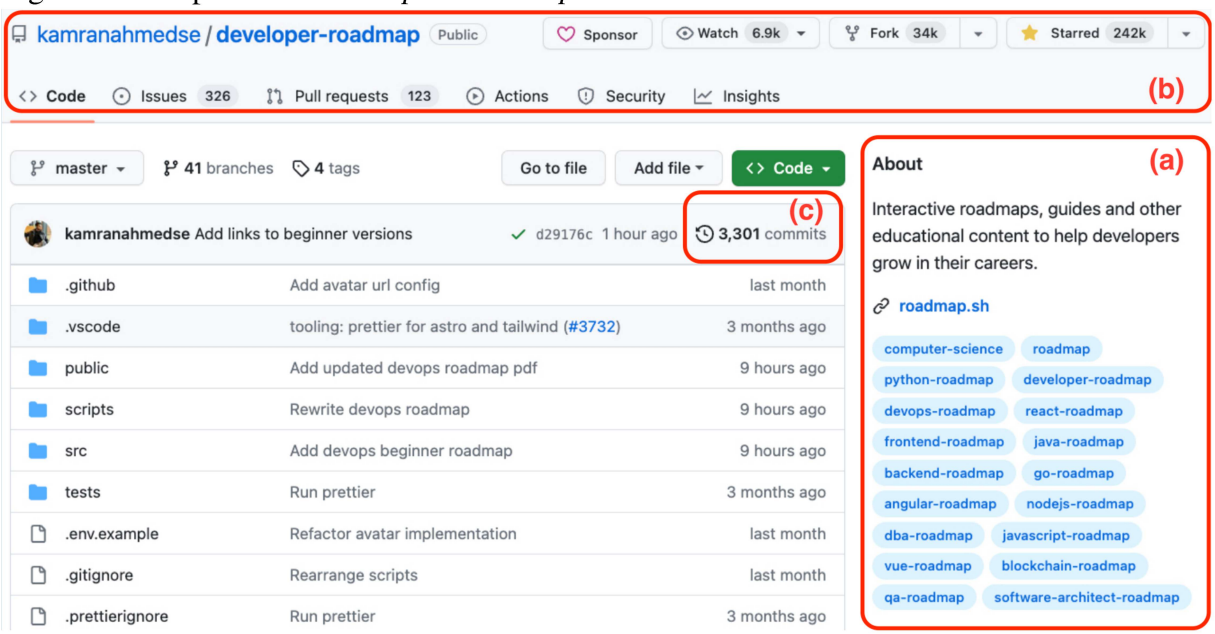
2.3.1 Conceitos principais

A Figura 18 mostra a página que pode ser visualizada ao acessar o repositório de código *Developer Roadmap*[4]. A imagem apresenta os diversos arquivos e pastas do repositório, mas foram destacadas em vermelho as partes **(a)**, **(b)** e **(c)** para serem discutidas. Em **(a)** é mostrada uma breve descrição, palavras-chave e o link para o *website* do projeto. Já na parte **(b)** da captura de tela, no botão *watch* são mostrados que 6,9 mil pessoas são notificadas das modificações no projeto, 34 mil fizeram o *fork* e 242 mil marcaram com estrela. O significado do *fork* será explicado ainda nesta seção, já a estrela serve para que o repositório fique salvo em uma lista de repositório favoritos. Ainda em **(b)**, existe o nome do repositório, nome autor do repositório e também submenus. Por fim, em **(c)**, é mostrado o número do *commits* feitos

[4] https://github.com/kamranahmedse/developer-roadmap

na *branch* principal do repositório. Além disso, ao clicar na quantidade de *commits*, em **(c)**, o usuário é redirecionado para o histórico de *commits*.

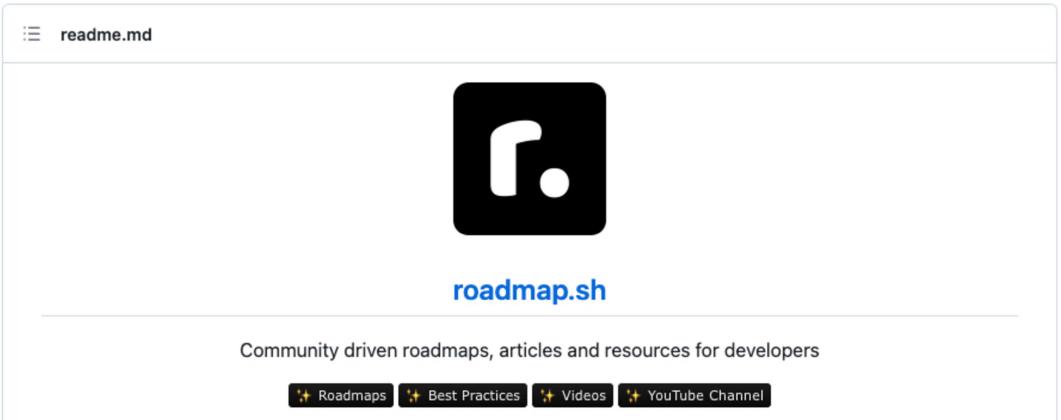
Figura 18 – Repositório *Developer Roadmap*



Fonte: Próprio autor

Ainda na página do repositório, ao rolar a tela, pode ser visualizada a *README* do projeto, apresentada na Figura 19. Esta parte de qualquer repositório consiste em um arquivo mostrado na tela inicial do repositório, que tem nome *README.md*, e é utilizado para fornecer uma introdução e informações importantes sobre o projeto. O GitHub utiliza um padrão chamado Markdown para formatação de texto, inserção de imagens, tabelas, blocos de código e links. Esse tipo de formatação pode ser utilizado em campos de texto dentro da ferramenta, mas também em qualquer arquivo com extensão *md*.

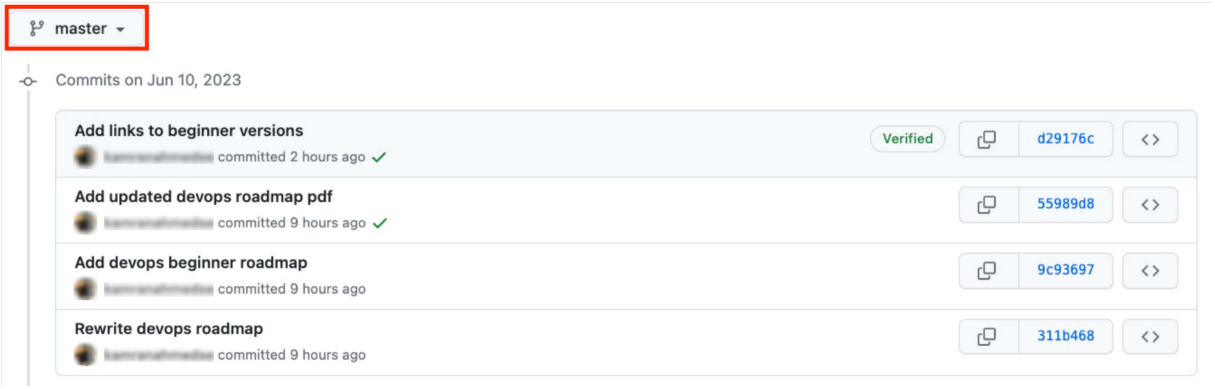
Figura 19 – *README* do repositório *Developer Roadmap*



Fonte: Próprio autor

No repositório local é utilizado o comando `git log` para mostrar os *commits*, já no GitHub pode ser visualizada a lista de modificações no histórico de *commits*, apresentado na Figura 20. Cada modificação é seguida de sua descrição, nome do colaborador, há quanto tempo foi feita a modificação e também um identificador único do *commit*. Note que assim como no Git, o histórico de modificações varia de acordo com a *branch*. Devido a isso, existe um menu que permite alternar entre as ramificações do projeto, esse menu está destacado na captura de tela da Figura 20.

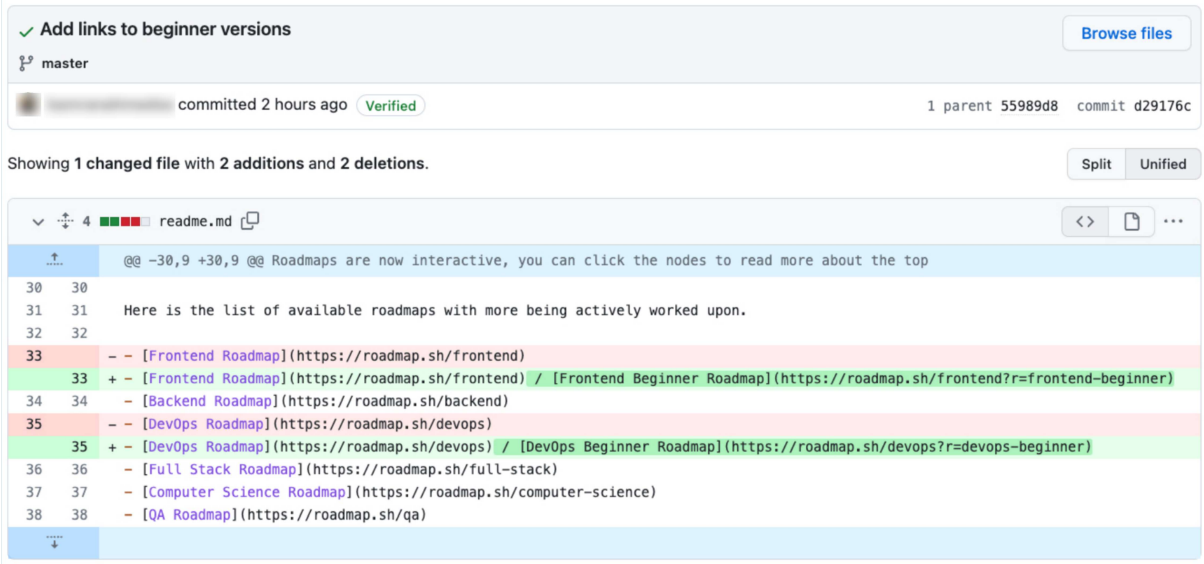
Figura 20 – Histórico de *commits* do repositório *Developer Roadmap*



Fonte: Próprio autor

Outra funcionalidade importante é que ao clicar no identificador único de um *commit* o usuário é redirecionado para a tela da Figura 21, que mostra a modificação que foi feita. Em verde e ao lado de um sinal de mais são apresentadas as linhas que foram adicionadas e em vermelho, ao lado de um sinal de menos, estão as linhas que foram removidas no *commit*.

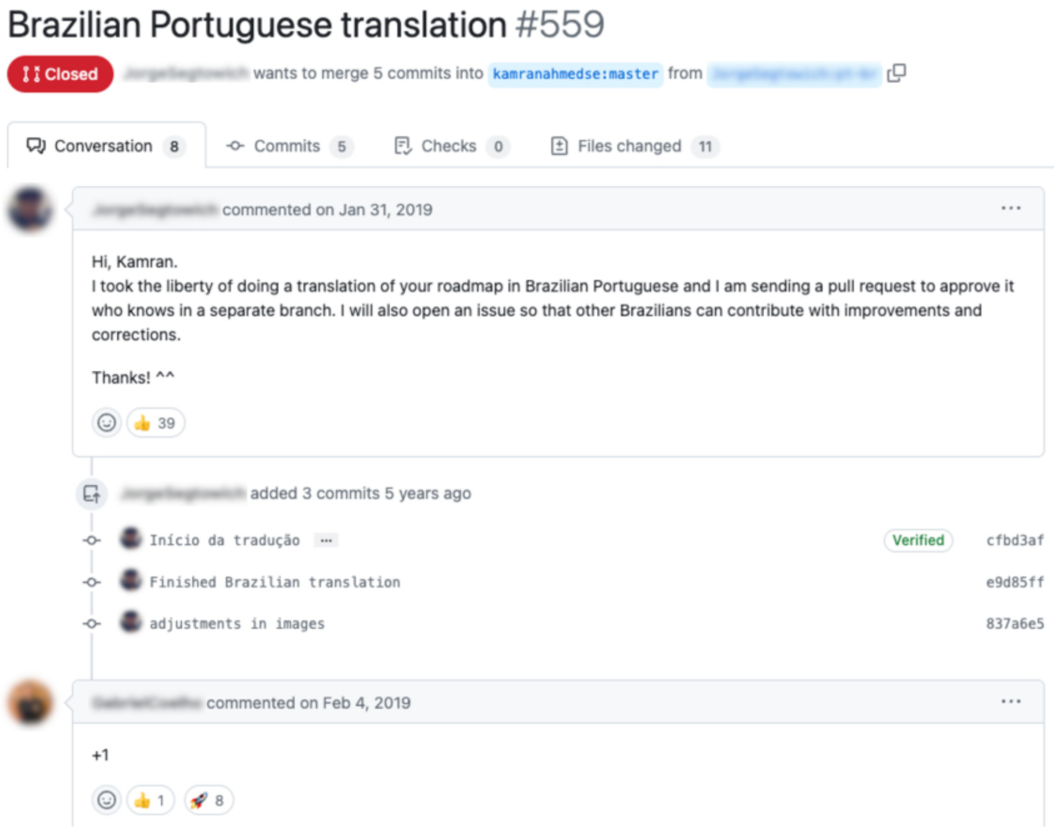
Figura 21 – Resultado após realizar um *pull* ou *fetch* seguido de *merge*



Fonte: Próprio autor

Ao realizar o pedido de *pull request*, um desenvolvedor solicita que alterações feitas em uma *branch* sejam mescladas em uma outra ramificação, por padrão a solicitação é feita para a *branch* principal. Na Figura 22, é apresentado que o autor do *pull request* adiciona um título e também pode adicionar uma descrição. A descrição pode conter explicação sobre os *commits* feitos, capturas de tela e outros detalhes relevantes para a modificação que vai ser introduzida. Além disso, existe também uma seção de comentários em que outros desenvolvedores podem discutir sobre o *pull request*.

Figura 22 – Corpo de um *pull request*



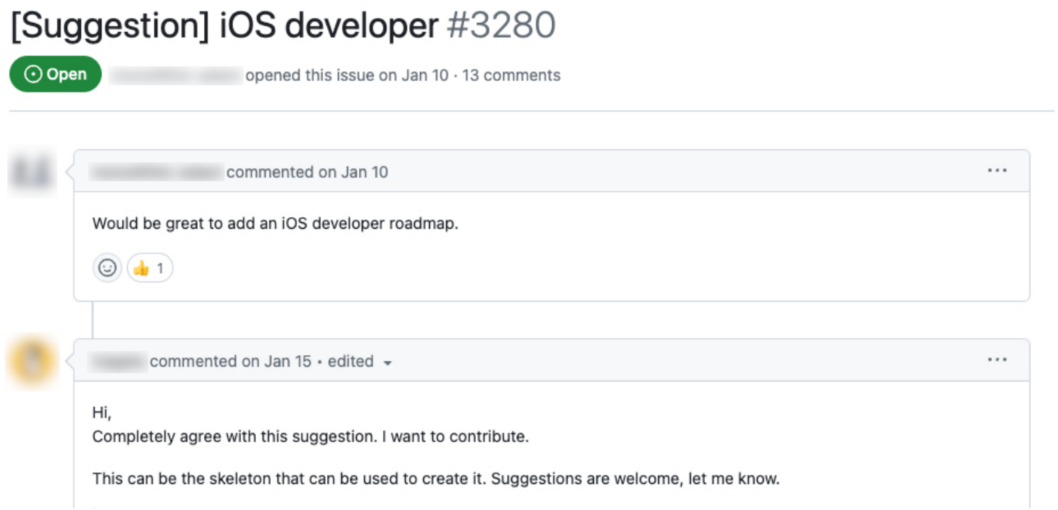
Fonte: Próprio autor

O conteúdo de uma *issue* é mostrado na Figura 23, que segue o mesmo padrão de um *pull request*. Ao invés de solicitar modificações, as *issues* podem ser utilizadas para sugerir novas funcionalidades, reportar *bugs* e realizar discussões em geral. Diferente do *pull request*, a *issue* não é associada com *commits*, mas também tem um tópico com descrição e outros desenvolvedores podem realizar comentários.

Cada *issue* e *pull request* no GitHub é atribuído a um identificador único, que permite referenciar e vincular discussões entre eles. Um exemplo disso é destacado na Figura 23, onde a *issue* possui o identificador `#3280`. Esse identificador pode ser utilizado para fazer referência

a essa discussão em outras *issues*, *pull requests* ou comentários, tanto na descrição quanto nos comentários. Os identificadores únicos citados nas discussões funcionam como links para acessar rapidamente a *issues* ou *pull requests* correspondente.

Figura 23 – Corpo de uma *issue*



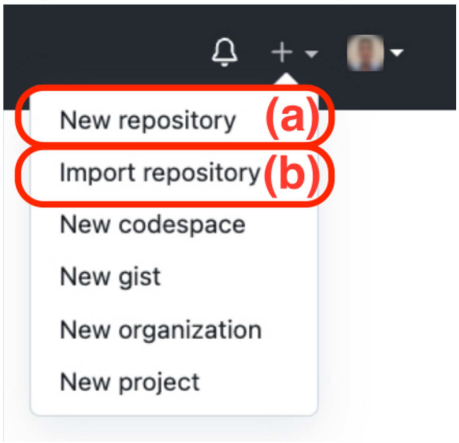
Fonte: Próprio autor

2.3.2 Criação e manutenção de um projeto próprio

O GitHub oferece, no canto superior direito, um menu de criação acessível em qualquer página do *site*. A Figura 24 mostra esse menu, além disso, na imagem foram destacadas as opções **(a)** e **(b)** para discussão. Destacado por **(a)** na imagem, temos a opção que permite criar um novo repositório. Essa é a escolha ideal para iniciar um projeto completamente novo. Ao selecionar essa opção, o usuário será direcionado para a página de criação de um novo repositório, onde poderá inicializar o novo projeto. Por outro lado, o botão destacado por **(b)**, na Figura 24, oferece a possibilidade de importar um repositório existente de outra plataforma de hospedagem de repositório Git, como GitLab ou Bitbucket. Essa opção é útil para migrar um projeto para o GitHub, aproveitando todos os recursos e funcionalidades oferecidos pela plataforma. Ao utilizar a importação do GitHub, é possível transferir projeto, incluindo o histórico de versões (GITHUB, 2023).

A Figura 25 e a Figura 26 ilustram a página de criação de um novo repositório no GitHub. É importante notar que a Figura 26 é uma continuação da Figura 25 e também que ambas têm áreas destacadas para a discussão. Essas imagens fornecem uma visão geral do processo de criação de um novo repositório e ajudam os usuários a entender as diferentes opções

Figura 24 – Menu de criação

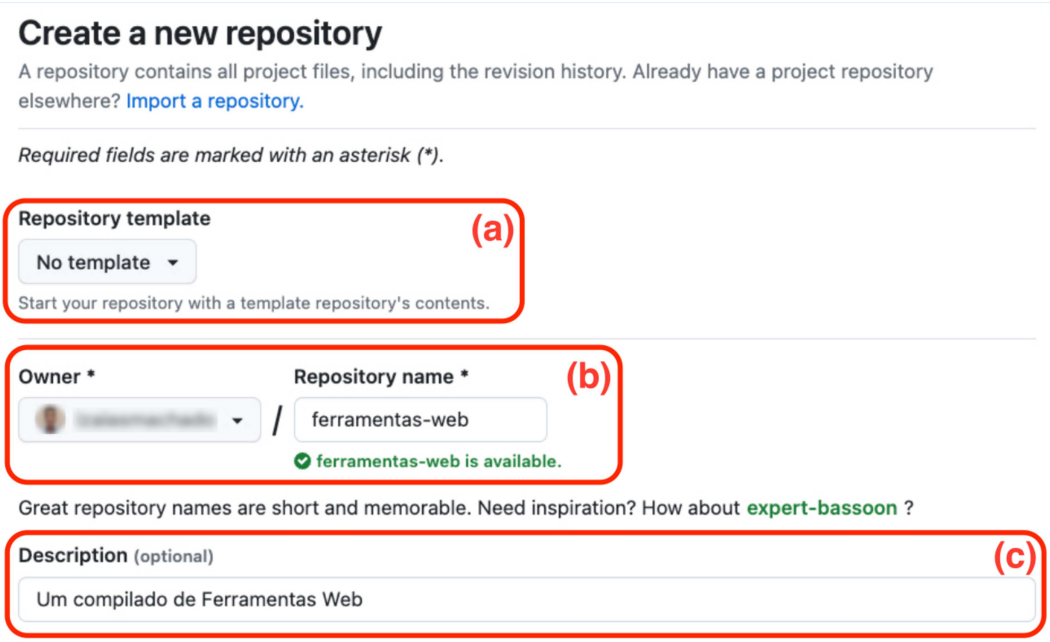


Fonte: Próprio autor

disponíveis ao configurar um projeto no GitHub.

O primeiro fragmento da página para criação de um novo repositório é apresentado na Figura 25. Em **(a)**, há um menu que permite navegar entre os repositórios *templates* salvos pelo desenvolvedor, que podem ser usados como base para a criação do novo repositório. Em **(b)**, há uma caixa de texto na qual é possível nomear o novo repositório. Nessa mesma área, um texto em verde destaca que o nome escolhido está disponível. Isso ocorre porque não é permitido ter dois repositórios com exatamente o mesmo nome em uma mesma conta do GitHub. Já em **(c)**, é apresentada outra caixa de texto que permite adicionar uma descrição, que é mostrada na página do repositório.

Figura 25 – Criação de um repositório - Parte 01



Fonte: Próprio autor

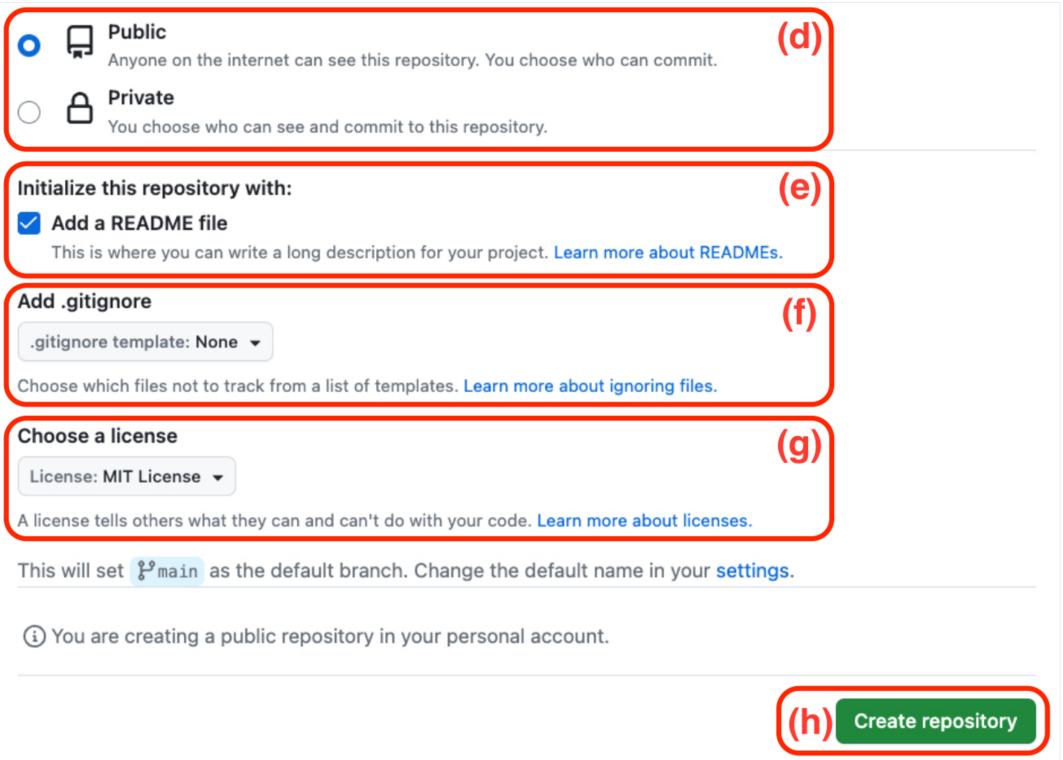
A Figura 26 apresenta o segundo fragmento da página de criação de um novo repositório. Em **(d)**, é possível escolher entre disponibilizar o repositório de forma pública ou privada. Ao torná-lo público, qualquer pessoa da internet poderá visualizar o repositório, mas apenas as pessoas selecionadas poderão contribuir para ele. Já na opção privada, é necessário adicionar pessoas específicas a uma lista para que possam ver o repositório e fazer contribuições. É importante destacar que, após a criação do repositório, é possível alterar sua configuração de público para privado e vice-versa por meio da página de configurações do repositório.

Na área destacada por **(e)**, na captura de tela da Figura 26, pode-se marcar uma opção para que um arquivo *README* seja criado de forma automática. Essa opção permite a criação do arquivo no processo de inicialização do repositório, eliminando a necessidade de criar o arquivo usando um cliente Git. Destacado por **(f)**, existe um menu que permite selecionar um dos *templates* disponíveis para o arquivo `.gitignore`. Esse arquivo é utilizado para especificar quais arquivos ou diretórios devem ser ignorados pelo sistema de controle de versão Git ao realizar um *commit*. Na opção destacada por **(g)**, é possível escolher a licença de uso para o código do repositório. Uma das licenças de código aberto mais comuns é a Licença MIT. Essa licença permite o uso, modificação e distribuição do código, tanto para fins comerciais quanto não comerciais. Caso selecionada essa opção, será adicionado um arquivo de licença ao repositório com termos da licença.

Para finalizar o processo de criação de um novo repositório deve ser clicado no botão destacado por **(h)**, apresentado na Figura 26, para criar o repositório. Neste exemplo, o repositório tem o nome *ferramentas-web*, possui uma descrição associada a ele e foi criado de forma pública, incluindo automaticamente um arquivo *README.md*. Nenhuma opção de arquivo `.gitignore` foi selecionada e a licença MIT foi escolhida como a licença de uso para o código.

A Figura 27 mostra o repositório após a sua criação. Ao criar um repositório no GitHub, é automaticamente feito um *commit* inicial com a descrição “Initial commit”. Nesse *commit* inicial, são adicionados os arquivos da licença escolhida e o arquivo *README*, que contém informações importantes sobre o projeto. No exemplo apresentado na Figura 27, os arquivos *README* e licença foram adicionados, enquanto o arquivo `.gitignore` não foi incluído. É importante ressaltar que caso nenhum arquivo seja criado ou selecionado durante a criação do repositório, não haverá o *commit* inicial e o repositório será criado vazio, sem nenhum *commit* registrado no histórico de versões.

Figura 26 – Criação de um repositório - Parte 02



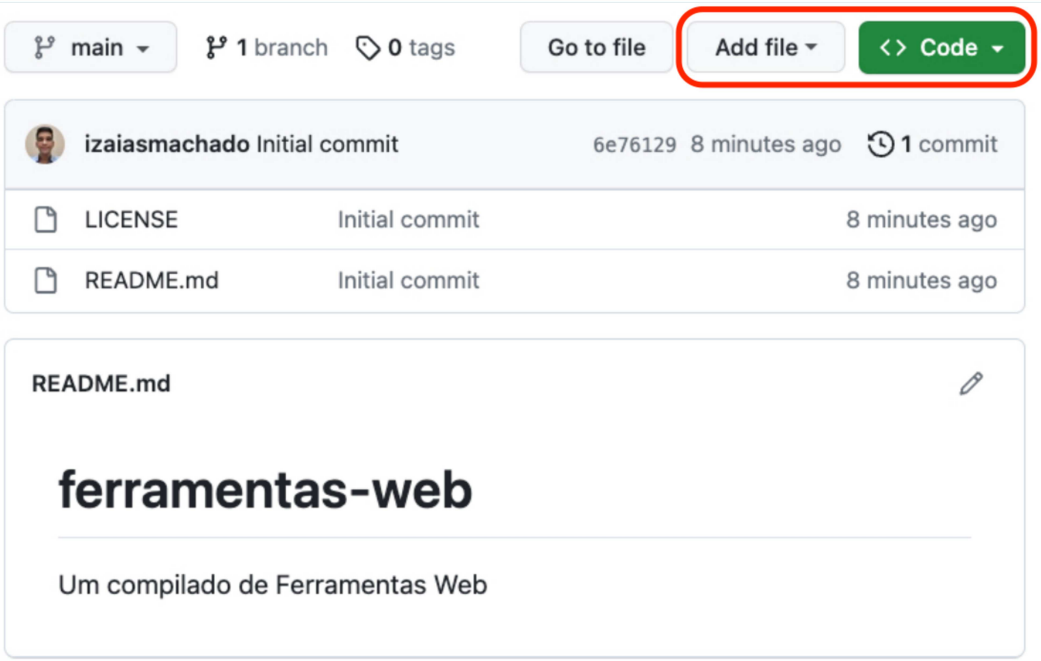
Fonte: Próprio autor

Está destacado em vermelho, na Figura 27, os botões: “Add file” e “Code”. Com isso, o botão “Add file” está presente para adicionar novos arquivos ao repositório por meio da interface do GitHub, sem precisar executar comandos pelo Git. Já o botão verde de nome “Code” é utilizado para abrir o menu de opções relacionadas ao código do repositório, permitindo baixar ou também acessar o link do repositório, para realizar o clone com o Git.

A Figura 28 apresenta o menu exibido ao clicar no botão “Code”. Esse menu fornece algumas informações e também opções para baixar e editar o código do repositório. Dentre essas opções, conforme destacado em vermelho na imagem, é listado o endereço do repositório remoto em HTTPS e SSH. Esse endereço é utilizado para baixar o repositório remoto para a máquina do desenvolvedor usando o comando `git clone` do Git, que é explicado detalhadamente na Subseção 2.2.3.

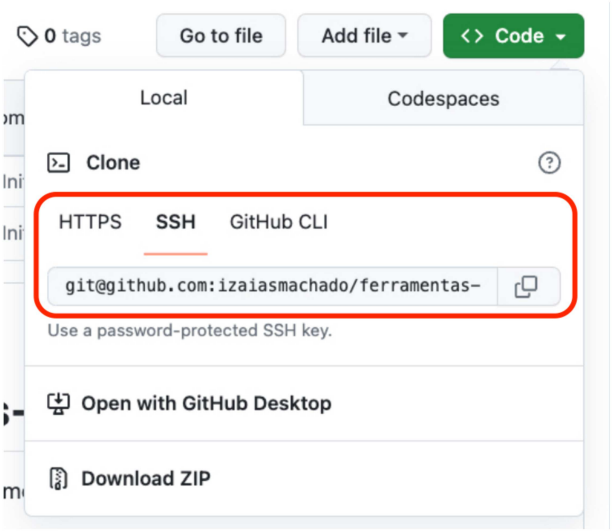
Ao clicar no botão “Add file”, disponível na página do repositório, é mostrado um menu com duas opções para a criação de um novo arquivo. Esse menu é apresentado na Figura 29, em que foram destacadas as seções **(a)** e **(b)** da imagem. É importante ressaltar que, ao criar arquivos por meio da interface do GitHub, o processo de *commit* é realizado internamente pela plataforma, sem a necessidade de usar o Git. Nesse sentido, é apresentada em **(a)** a opção de criar um novo arquivo utilizando a interface Web do GitHub. Já em **(b)**, encontra-se a opção de

Figura 27 – Repositório criado



Fonte: Próprio autor

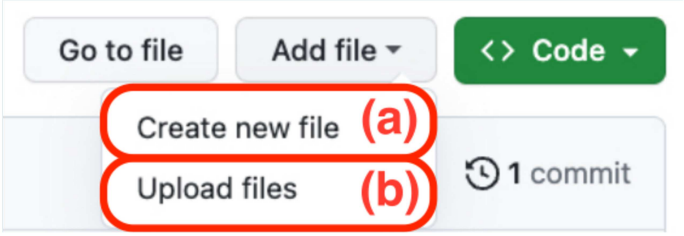
Figura 28 – Fazer o clone do repositório



Fonte: Próprio autor

realizar o *upload* de um arquivo da máquina do desenvolvedor diretamente para o repositório.

Figura 29 – Menu para adicionar arquivos



Fonte: Próprio autor

Ao clicar na opção para criar um novo arquivo pela interface do GitHub, é aberta a página mostrada na Figura 29. Nessa página, além de adicionar o conteúdo do arquivo no editor de texto do site, há uma caixa de texto, destacada em vermelho na imagem, onde é possível adicionar o nome do arquivo. No exemplo dado, o nome do arquivo digitado foi *docs/introducao.md*, indicando que um arquivo chamado *introducao.md* será criado na pasta *docs*. Quando o desenvolvedor terminar de adicionar o conteúdo do arquivo, deve-se clicar no botão verde "Commit changes..." para fazer o *commit* do novo arquivo. Em um caso adverso, se necessário, é possível clicar no botão "Cancel changes" para descartar a criação do novo arquivo.

Figura 30 – Página para criação de novo arquivo

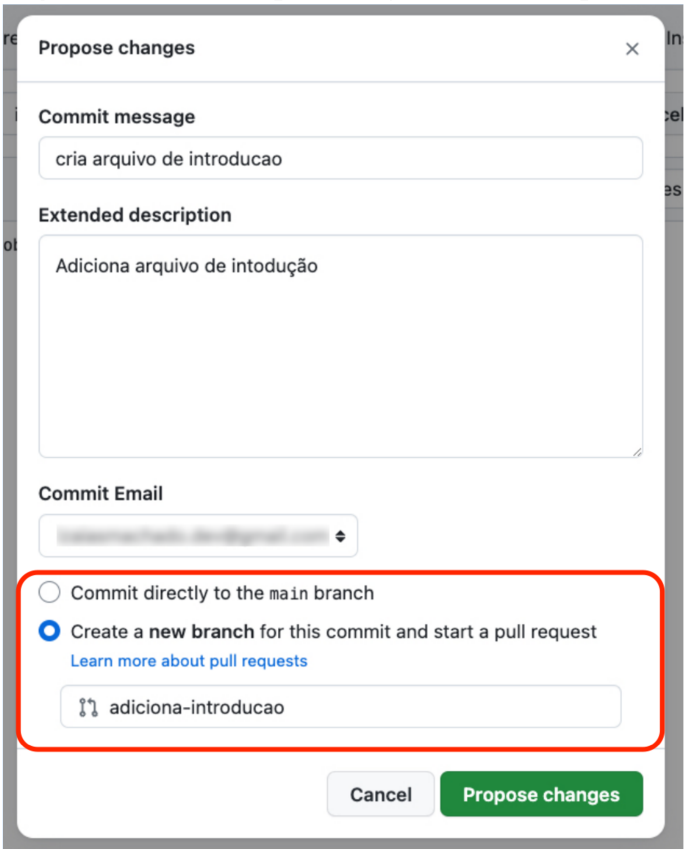
Fonte: Próprio autor

Ao clicar no botão para realizar o *commit* do novo arquivo, uma tela de configurações do *commit* é aberta. Essa tela é apresentada na Figura 31, em que existem os campos de texto para a mensagem de *commit* e descrição estendida. A mensagem de *commit* consiste em uma breve descrição das alterações realizadas. Além disso, na caixa de texto maior é possível inserir uma descrição mais detalhada das modificações feitas. O preenchimento da mensagem de *commit* é obrigatório, enquanto a descrição mais detalhada é opcional. Destacado na imagem, há a opção de escolher entre fazer o *commit* diretamente na *branch* principal "main" ou criar uma nova *branch* para fazer o *commit*. No exemplo apresentado, foi escolhida a criação de uma nova *branch* com o nome *adiciona-introducao*.

Ao realizar o *commit* da nova *branch*, por estar utilizando a interface do GitHub, o desenvolvedor é redirecionado para uma tela de criação do *pull request*. É importante enfatizar que, geralmente, ao criar uma nova *branch* utilizando o Git em uma máquina local, o desenvolvedor não é automaticamente direcionado para a tela de criação do *pull request* no GitHub. Essa transição ocorre apenas quando são feitos *commits* pela interface do GitHub.

O *pull request* abre uma solicitação de *merge* de *branches* por meio da interface do GitHub. Após aberta, a solicitação pode ser revisada e discutida pelos colaboradores do projeto. É importante ressaltar que o *merge* não necessariamente precisa ser realizado na *branch* principal. Dessa forma, o *pull request* permite que as modificações sejam incorporadas ao código de forma

Figura 31 – *Commit* para criação do novo arquivo



Fonte: Próprio autor

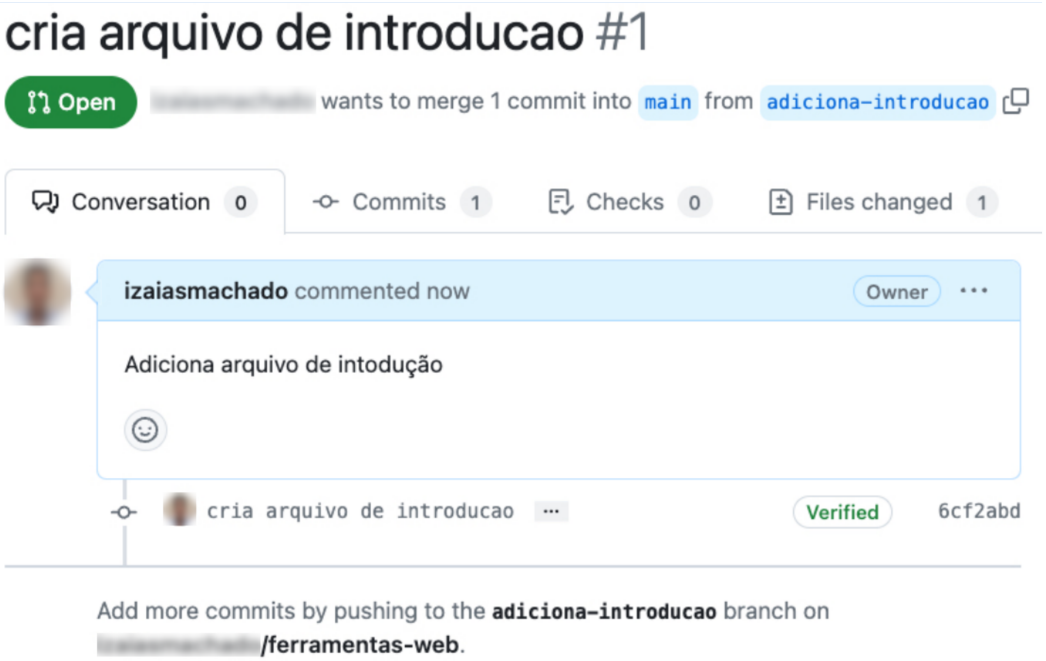
controlada e após a revisão dos colaboradores.

A Figura 32 apresenta um fragmento da tela do *pull request* que cria o arquivo *introducao.md*. Nessa parte da tela do *pull request*, pode-se observar no título, uma descrição do que está sendo proposto e também a mensagem do *commit* que foi feito. É também destacado que esse *pull request* solicita a realização do *merge* da *branch adiciona-introducao* na branch principal "main". Além disso, é mostrado um estado que é *Open*, para indicar o que está disponível para revisão e discussão pelos colaboradores do projeto.

Outra seção da tela do *pull request* apresenta a informação de que não há conflito entre as *branches* envolvidas, como pode ser observado pelo texto em negrito na Figura 33. No caso de haver conflitos no conteúdo dos arquivos entre as *branches* do *pull request*, não será possível realizar o *merge* diretamente. Para resolver esses conflitos, é necessário utilizar um editor de texto no próprio GitHub ou o Git localmente.

O botão em destaque na cor vermelha, na Figura 33, permite realizar o *merge* proposto pelo *pull request*. É importante mencionar que apenas os colaboradores do projeto com permissão para realizar o *merge* têm acesso a esse botão. Após a conclusão do *merge*, o estado do *pull request* é alterado de *Open* para *Merged*, indicando que as modificações foram

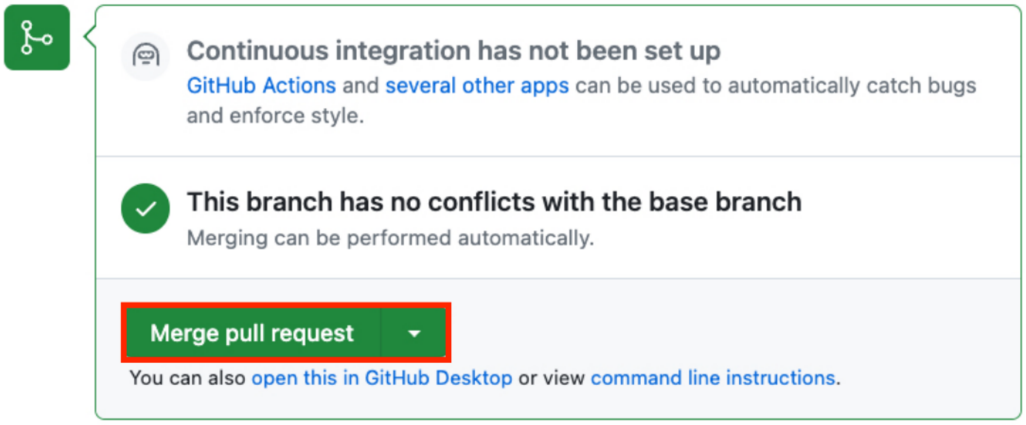
Figura 32 – *Pull request* - Parte 01



Fonte: Próprio autor

incorporadas à *branch* principal.

Figura 33 – *Pull request* - Parte 02



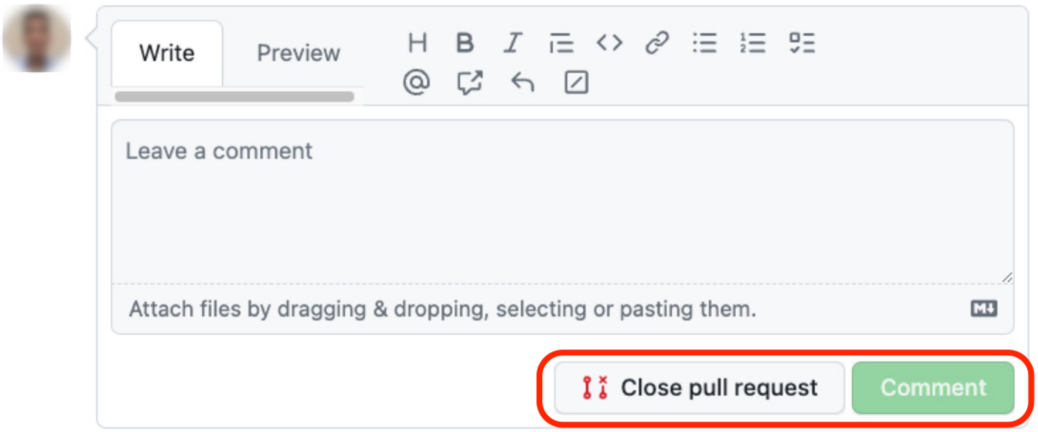
Fonte: Próprio autor

Seguindo a discussão sobre o *pull request*, é mostrado na Figura 34 a parte da página que permite fazer comentários e fechar o *pull request*. Nessa seção, os colaboradores têm a possibilidade de adicionar comentários com sugestões e observações sobre as modificações propostas. Desse modo, é por meio dos comentários que podem ser abordados ajustes de código e possíveis melhorias nas funcionalidades implementadas.

Para finalizar a discussão da Figura 34, destacado junto do botão para deixar um comentário, há também o botão para fechar o *pull request*. Ao clicar nesse botão, o *pull request*

passa do estado aberto *Open* para o estado fechado *Closed*. O fechamento do *pull request* indica que a discussão está concluída, mas as modificações não são mescladas na *branch* de destino. Somente *pull requests* abertos permitem fazer o *merge*. Nesse sentido, caso seja decidido reconsiderar a solicitação, é possível reabrir o *pull request*, para que volte do estado *Closed* para *Open* novamente.

Figura 34 – *Pull request* - Parte 03



Fonte: Próprio autor

2.3.3 Como colaborar com projetos de código aberto?

Repositórios de código aberto são projetos de *software* disponibilizados publicamente para acesso, colaboração e contribuição da comunidade de desenvolvedores. Esses repositórios permitem que o código-fonte seja compartilhado de forma transparente e que pessoas de todo o mundo possam os estudar e contribuir. Muitas bases de código tem encontrado no GitHub uma plataforma ideal para sua hospedagem e colaboração. Podem ser citados alguns projetos de grande importância que estão hospedados no GitHub, como o *kernel* do Linux[5], o *framework* Node.js[6] e a biblioteca do *React*[7]. Essa escolha se deve à ampla adoção do GitHub pela comunidade de desenvolvedores e os recursos que a plataforma oferece.

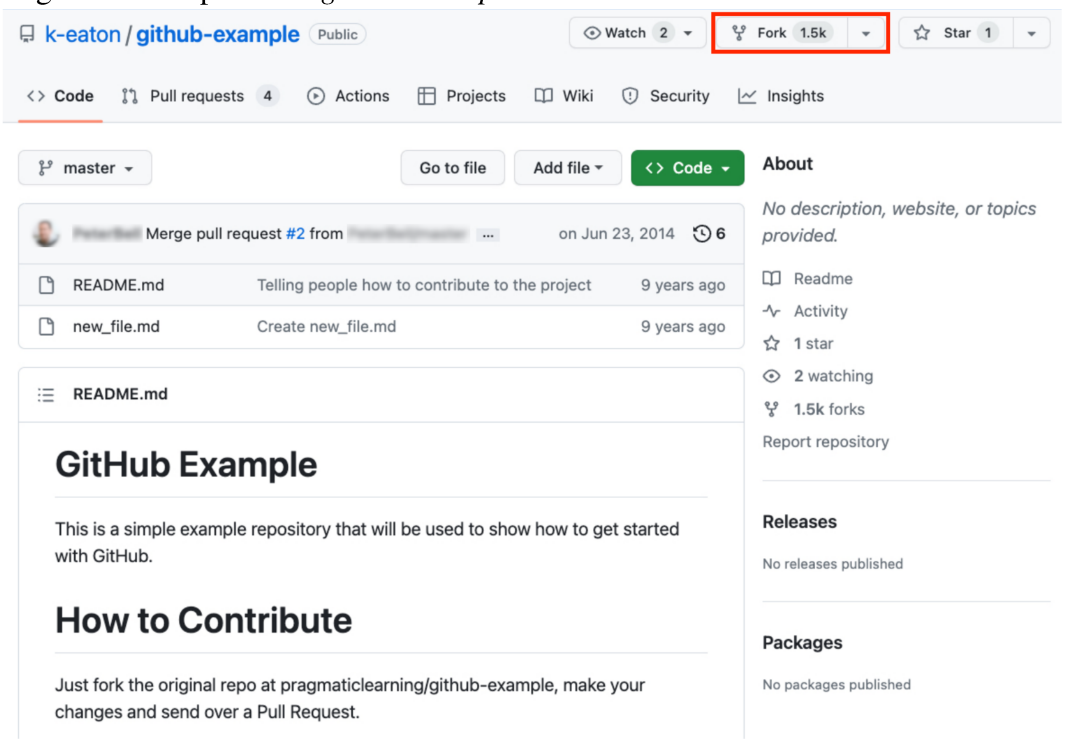
Para contribuir de forma direta, como na Subseção 2.3.2 é necessário ser dono ou possuir permissões de colaborador em um repositório. No entanto, é possível contribuir para quaisquer repositórios fazendo uma cópia desse repositório em sua própria conta do GitHub, o que é conhecido como *fork*. Ao fazer um fork, é possível realizar modificações na cópia

[5] https://github.com/torvalds/linux
[6] https://github.com/nodejs/node
[7] https://github.com/facebook/react

do repositório e, em seguida, solicitar que essas alterações sejam incorporadas ao repositório original por meio de um *pull request*. Dessa forma, o *fork* permite que os desenvolvedores contribuam com projetos em que não possuem acesso direto, possibilitando o envolvimento em projetos de código aberto e a colaboração com a comunidade de desenvolvedores.

O repositório *GitHub Example*[8] foi escolhido para exemplificar uma contribuição em um projeto por meio de um *fork*. Nesse sentido, na Figura 35, é mostrado a página principal do repositório em questão. O botão de *fork* está destacado em vermelho, e ao clicá-lo, inicia-se o processo de criação de uma cópia do repositório na conta do GitHub do desenvolvedor que realiza o *fork*.

Figura 35 – Repositório *github-example*



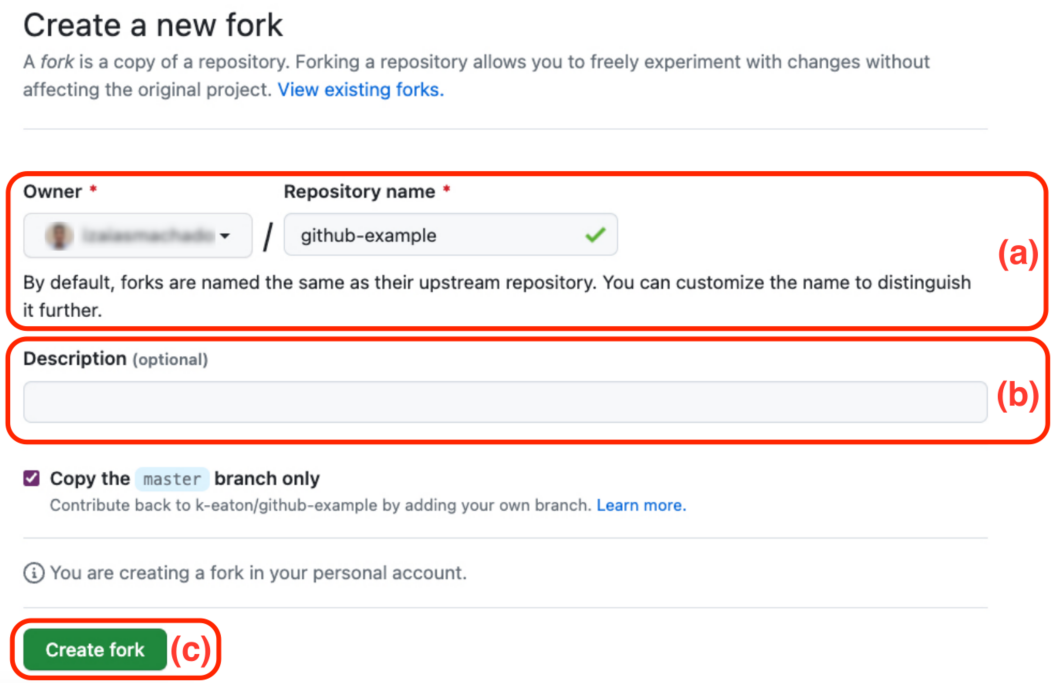
Fonte: Próprio autor

Após clicar no botão de *fork*, o desenvolvedor é redirecionado para a página mostrada na Figura 36. Nessa imagem, existem algumas seções destacadas em vermelho e identificadas pelas letras **(a)**, **(b)** e **(c)**. Na seção **(a)**, há uma caixa de texto na qual é possível dar um nome ao repositório que será criado. Por padrão, os *forks* têm o mesmo nome do repositório original, mas o desenvolvedor pode modificá-lo conforme necessário. Em **(b)**, é possível adicionar uma descrição ao novo repositório. Essa descrição pode ser útil para fornecer informações adicionais sobre o objetivo ou conteúdo do repositório. Posteriormente, após a criação do repositório *fork*,

[8] https://github.com/k-eaton/github-example

também será possível modificar a descrição. Por fim, em **(c)**, encontra-se o botão para efetuar a criação do repositório *fork*. Ao clicar nesse botão, o GitHub irá criar uma cópia do repositório original, associando-o à conta do desenvolvedor que realizou o *fork*.

Figura 36 – Criação de um repositório *fork*



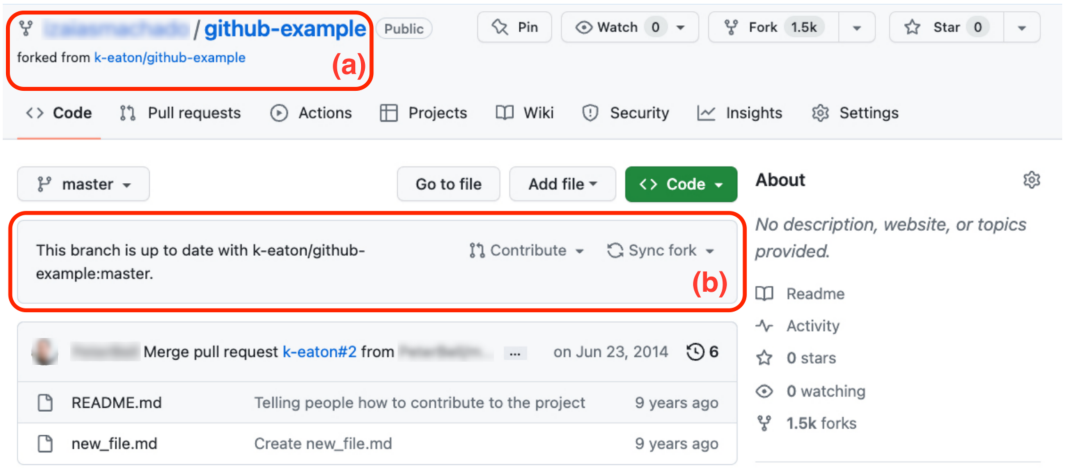
Fonte: Próprio autor

Após criar o repositório *fork*, o resultado é mostrado na Figura 37. Está destacado em **(a)**, o nome do repositório e também que esse é um repositório *fork*. Nesse sentido, na seção **(a)**, o repositório exibido não é mais o repositório original, mas sim o *fork* criado a partir dele. Isso significa que o desenvolvedor fez uma cópia do repositório base em sua própria conta do GitHub. Em **(b)** existe uma barra de *status* que compara os *commits* do repositório *fork* com o repositório base. Nesse caso específico, o *status* indica que o *fork* não possui modificações em relação ao repositório base. Isso significa que o desenvolvedor ainda não fez alterações no *fork*. Na barra de *status*, também existem os botões *contribute* e *sync fork*, que serão discutidos a seguir.

O botão *contribute*, mostrado em **(b)** da Figura 37, é uma opção que permite ao desenvolvedor contribuir para o repositório base por meio do processo de *pull request*. Ao clicar nesse botão, o desenvolvedor é redirecionado para uma página na qual pode comparar as alterações feitas em seu *fork* com o repositório base e solicitar a incorporação dessas alterações ao projeto original. Para contribuir, é necessário fazer *commits* em seu *fork*, registrando as alterações realizadas nos arquivos do repositório.

Para finalizar a discussão da Figura 37, o botão *sync fork* permite que o desenvolvedor atualize o seu *fork* com as alterações mais recentes feitas no repositório base. No entanto, é importante ressaltar que essa funcionalidade só está disponível se houverem modificações no repositório base. Ao clicar no botão *sync fork*, será possível realizar uma sincronização, incorporando os *commits* mais recentes do repositório base ao repositório *fork* do desenvolvedor. Essa funcionalidade é útil quando o desenvolvedor deseja manter seu *fork* atualizado com as modificações feitas no projeto original, permitindo acompanhar o progresso e as melhorias feitas pela comunidade.

Figura 37 – Repositório *fork* criado

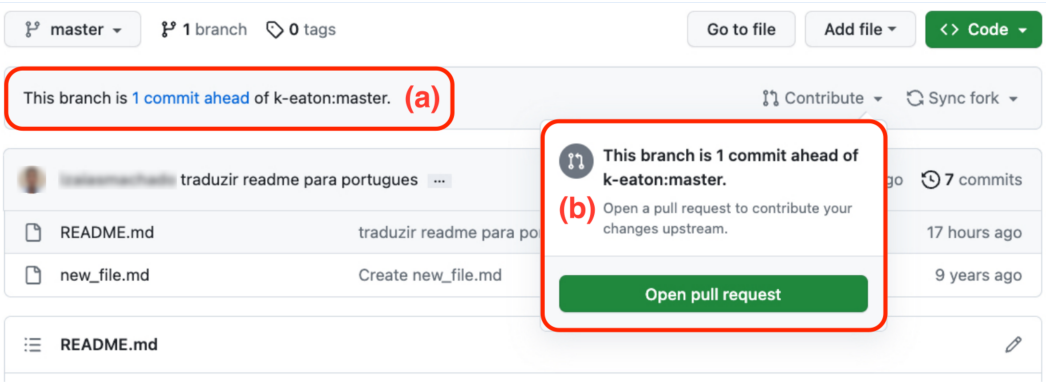


Fonte: Próprio autor

A Figura 38 exibe uma mensagem, destacada por **(a)**, indicando que o repositório *fork* está um *commit* à frente do repositório base. Isso significa que foram feitas alterações no *fork* não incorporadas ao projeto base. No exemplo em questão, foi feito um *commit* que modifica o conteúdo do arquivo *README*, traduzindo-o para português. Anteriormente, foi discutido sobre o botão para contribuir, mas não haviam modificações para a criação de um *pull request*. Nesse caso, pelo fato de possuir um *commit* a frente do repositório base, é possível clicar no botão para contribuir e visualizar a área destacada por **(b)**. Nesta parte destacada da Figura 38, é informado novamente que a *branch* principal do repositório *fork* está um *commit* à frente do repositório base e existe um botão que permite abrir um novo *pull request*.

Já a Figura 39 mostra a tela para criar um novo *pull request* do repositório *fork* para o repositório base. Nela, está destacado em vermelho o repositório de origem e a *branch* de origem, assim como o repositório de destino e a *branch* de destino. Nos campos correspondentes, é exibido o nome do repositório *fork* e sua *branch* principal que é a “master”, seguido pelo

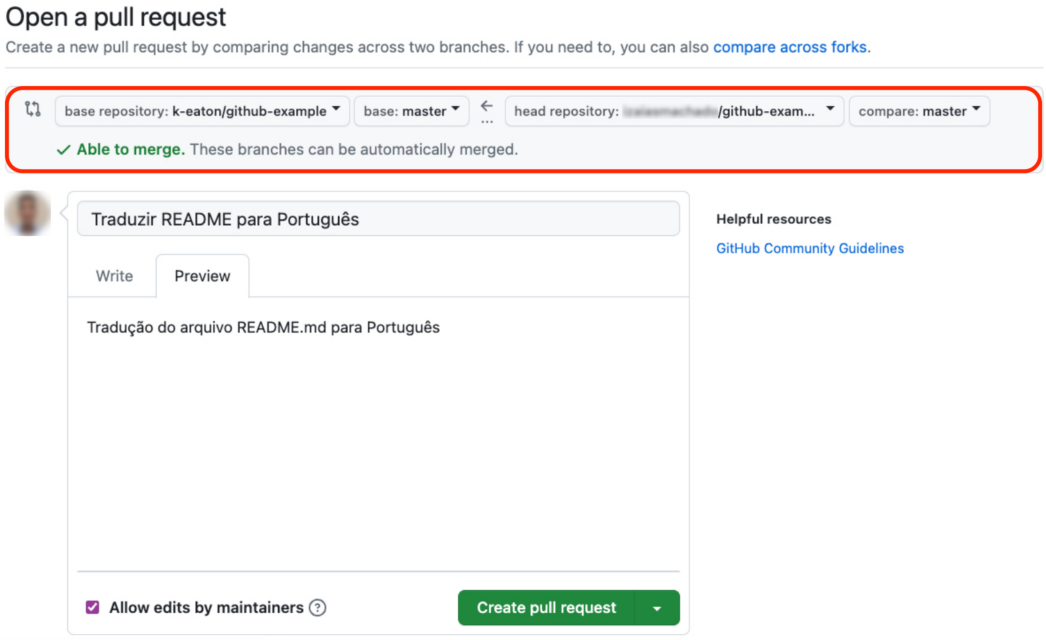
Figura 38 – Repositório *fork* após um *commit*



Fonte: Próprio autor

nome do repositório base e sua *branch* principal “master”. Essas informações são essenciais para definir a origem e o destino do *pull request*, possibilitando que as modificações feitas no repositório *fork* sejam solicitadas e incorporadas ao projeto original.

Figura 39 – Criação do *pull request* do repositório *fork* para o repositório base



Fonte: Próprio autor

É importante ressaltar que, ao criar um *pull request* com destino o repositório base, ele será listado no repositório base, para que os colaboradores desse projeto possam revisar e analisar as modificações propostas. Portanto, é fundamental fornecer um título descritivo para o *pull request*, juntamente com uma descrição detalhada do que foi alterado e por quê. Essa abordagem deve facilitar a identificação e compreensão das modificações pelos desenvolvedores do projeto de código aberto.

2.4 Virtualização de aplicações com Docker

Docker[9] é uma plataforma aberta que permite o desenvolvimento e implantação de aplicações em ambientes de execução chamados de *containers* (DOCKER, 2020). Esses *containers* permitem isolar uma aplicação junto de suas dependências da máquina hospedeira, garantindo que os processos da máquina não tenham acesso aos processos em execução dentro do *container*, e vice-versa. Em síntese, um *container* é uma instância isolada que contém tudo o que é necessário para a execução de um aplicativo, incluindo sistema de arquivos, dependências, configurações e o código da aplicação.

Uma das vantagens dos *containers* é que eles são autocontidos, o que significa que podem ser facilmente executados em diferentes ambientes, sem a necessidade de configurações adicionais. Isso é especialmente útil quando desenvolvedores de uma mesma aplicação utilizam sistemas operacionais diferentes. Nesse sentido, com o uso do Docker é possível ter um ambiente virtual padrão para desenvolvimento e implantação de aplicações. Assim, diferente de ferramentas que rodam nativamente em mais de uma plataforma, a utilização de *containers* oferece uma solução consistente que evita erros decorrentes de configurações inadequadas, devido à execução em um único ambiente virtual padronizado.

2.4.1 Conceitos básicos e arquitetura

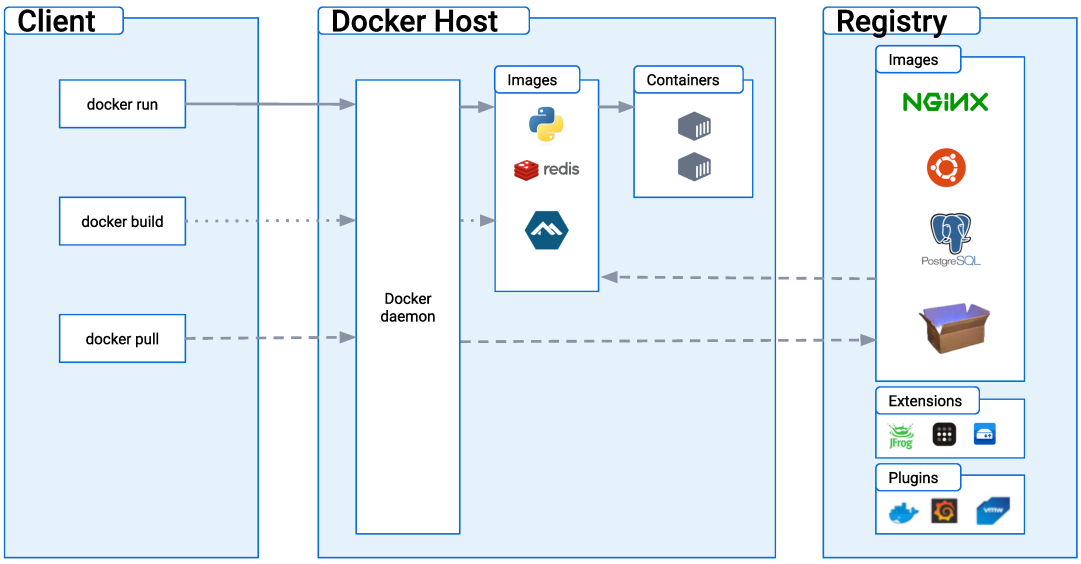
Detalhes essenciais da arquitetura Docker são apresentados na Figura 40. Nessa arquitetura, o cliente, representado como *client*, desempenha um papel fundamental, pois é por meio dele que os usuários interagem com o Docker. Ele estabelece a comunicação com o *daemon*, também conhecido como *dockerd*, que é responsável por receber e processar as requisições, além de gerenciar os objetos Docker, como *containers* e imagens. O *host* mencionado na figura se refere à máquina na qual o *daemon* está em execução. É importante destacar que o cliente e o *daemon* podem ser executados em máquinas diferentes e ainda assim se comunicarem.

Além disso, na arquitetura Docker, existe o conceito de *registry*, que é uma aplicação de código aberto responsável pelo armazenamento e distribuição de imagens Docker. No contexto da Figura 40, um exemplo de *registry* é o Docker Hub. O Docker Hub é uma plataforma que oferece um *registry* público na qual desenvolvedores e organizações podem armazenar suas imagens Docker. Essas imagens podem ser distribuídas de forma pública, permitindo que

[9] https://www.docker.com

outros usuários possam utilizá-las, ou de forma privada, restringindo o acesso apenas a usuários autorizados. O Docker Hub é uma opção popular para compartilhar e descobrir imagens prontas para uso.

Figura 40 – Arquitetura do Docker



Fonte: Docker (2020)

Uma imagem serve como um modelo para criar um *container*, de modo que podem haver vários *containers* da mesma imagem em uma máquina. Desenvolvedores podem optar por baixar imagens públicas ou também criar suas próprias imagens. Por sua vez, um *container* é uma instância em execução de uma imagem. Após criado, a execução pode ser parada, retomada ou também o *container* pode ser deletado. O sistema de arquivos utilizado é próprio e não compartilhado com outras instâncias e é fornecido pela imagem usada como base na criação da imagem.

Conforme mencionado anteriormente, cada *container* requer uma série de configurações, como o sistema operacional e *scripts* necessários para a execução do projeto. Para criar uma imagem, é necessário descrever essas configurações em um arquivo chamado Dockerfile. O Dockerfile permite utilizar outras imagens como base para construir a nova imagem. Por exemplo, é possível usar uma imagem do sistema operacional Alpine Linux como base e adicionar os componentes e configurações específicas necessários para executar a aplicação em desenvolvimento.

2.4.2 *Exemplo prático com Dockerfile*

Para a inicialização do projeto, em um mesmo repositório são criados dois arquivos, um chamado *index.js* e outro *Dockerfile*. O primeiro, armazena o código de uma aplicação escrita em Node.js[10]. Já o outro, armazena as informações da imagem utilizada para criar o *container* da aplicação. O conteúdo do primeiro arquivo é mostrado na Listagem 2.4.1 e consiste em um servidor Web, que escuta requisições do tipo GET e retorna uma mensagem. Nas linhas 4 e 5 são utilizadas variáveis de ambiente para receber, respectivamente, os valores da porta que a aplicação deve rodar e o nome do ambiente de execução. Variáveis de ambiente são parâmetros globais que podem ser acessados por todos os programas que rodam em uma máquina. Caso o valor de *env* seja *development*, na resposta da requisição GET também é enviado o valor da variável que define o ambiente de execução.

```
1  const express = require("express");
2  const app = express();
3
4  const port = process.env.PORT;
5  const env = process.env.NODE_ENV;
6
7  app.get("/", (req, res) => {
8    const response = {
9      message: "Hello World!",
10   };
11
12   if (env === "development") {
13     response.env = env;
14   }
15
16   res.json(response);
17 });
18
19 app.listen(port);
```

Listagem 2.4.1 – Arquivo *index.js* do código do servidor Web

Na Listagem 2.4.2, é mostrado o *Dockerfile*, que é o arquivo de configuração da imagem dessa aplicação. A primeira linha define que a imagem utilizada como base para montar o *container* é do sistema operacional Alpine Linux que já vem com a Versão 16 do Node.js instalada. Nas linhas 3 e 5 é definido o diretório usado para salvar os arquivos no *container* e em

[10] https://nodejs.org

seguida copiado o arquivo da Listagem 2.4.1 para dentro do *container*. A linha 7 define um valor padrão para a variável de ambiente que é usada no código da aplicação para definir a porta. Na linha 9 é inicializado um arquivo de configurações do gerenciador de pacotes NPM[11] e na 10 instalada a biblioteca Express[12] utilizando o gerenciador de pacotes. O Express foi escolhido, pois é um *framework* que permite a criação de servidores Web com poucas linhas de código. Por fim, a linha 12 expõe a porta 3000, isto é, permite que a porta seja acessada pela interface de rede da máquina *host* e a 14 executa código do servidor Web.

```
1  FROM node:16-alpine
2
3  WORKDIR /app
4
5  COPY . index.js
6
7  ENV PORT=3000
8
9  RUN npm init -y
10 RUN npm install express
11
12 EXPOSE 3000
13
14 CMD ["node", "index.js"]
```

Listagem 2.4.2 – Arquivo *Dockerfile*

Através do Terminal ou Prompt de Comando do Windows, é possível acessar o diretório onde os arquivos foram criados e executar o comando de construção da imagem, conforme exemplificado na Listagem 2.4.3. Esse comando utiliza a *flag* `-t` seguida pelo nome *node-api* para nomear a imagem que será criada a partir desse arquivo. Além disso, no final do comando é passado um ponto que indica que o diretório em que o comando foi executado é o mesmo em que se encontra os arquivos *Dockerfile* e da aplicação. Adicionalmente, após criar a imagem pode ser executado o comando da Listagem 2.4.4, que lista todas as imagens disponíveis no *host*. Além do nome das imagens disponíveis, mostrado na Figura 41, também é listado uma *tag* relativa a versão da imagem, um identificador único *image id*, o tempo desde a criação *created* e o espaço ocupado em disco *size*.

Diante da imagem criada, o comando *run* permite instanciar o *container* a partir de

[11] https://www.npmjs.com
[12] https://expressjs.com

```
docker build -t node-api .
```

Listagem 2.4.3 – Criação da imagem por meio do *Dockerfile*

```
docker images
```

Listagem 2.4.4 – Listar imagens disponíveis

Figura 41 – Lista de imagens disponíveis

```
[➜  node-api-example docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED             SIZE
node-api                            latest              033c16c18774        2 days ago          125MB
```

Fonte: Próprio autor

uma imagem em um *host*. O comando da Listagem 2.4.5 é um *run* partindo da imagem *node-api* fornecendo configurações adicionais. A *flag* `--name` é utilizada para nomear o *container*, caso não seja nomeado, o *daemon* gera um nome de forma aleatória. Já a configuração `-p` mapeia a porta 3000 interna do *container* para a porta 3001 do *host*. Como já mencionado, o código do servidor Web utiliza variáveis de ambiente como parâmetros e o valor de *NODE_ENV* não foi definido no arquivo *Dockerfile* da Listagem 2.4.2. Dito isso, o parâmetro `-e` do comando *run* define valores para variáveis de ambiente. Por fim, a *flag* `-d` é utilizada para que o *container* execute em segundo plano.

```
docker run --name node-api-dev -p 3001:3000 -e NODE_ENV=development -d
↪  node-api
```

Listagem 2.4.5 – Instanciando *container* em ambiente simulado de desenvolvimento

O comando da Listagem 2.4.6 é muito similar ao da Listagem 2.4.5 que foi discutido, as únicas diferenças são que a porta mapeada no *host* foi a 3002 ao invés de 3001 e também que a variável de ambiente *NODE_ENV* tem o valor *production*. Desse modo, as duas instâncias explicadas são utilizadas para simular ambientes independentes de desenvolvimento e produção.

```
docker run --name node-api-prod -p 3002:3000 -e NODE_ENV=production -d
↪  node-api
```

Listagem 2.4.6 – Instanciando *container* em ambiente simulado de produção

O comando da Listagem 2.4.7, com saída mostrada na Figura 42, lista todos os *containers* em execução no *host*. Além dos nomes definidos para cada *container*, são listados os

identificadores únicos *container id*, imagem que é utilizada como base para a instância, comando utilizado para a inicialização e há quanto tempo foi criado. Por padrão, o comando lista somente as instâncias com *status up* e ao lado o tempo em que se encontra nesse estado. Ao usar o parâmetro `-a`, todos os *containers* do *host* são exibidos. No caso de mostrar todos os *containers*, alguns *status* que poderiam aparecer são o *exited* quando foi encerrado e o *paused* quando foi pausado pelo *daemon*. Por fim, ainda se referindo ao resultado de execução da Figura 42, é possível visualizar a faixa de portas ocupadas pelo *container* e seu nome.

```
1 docker ps
```

Listagem 2.4.7 – Listando *containers* em execução

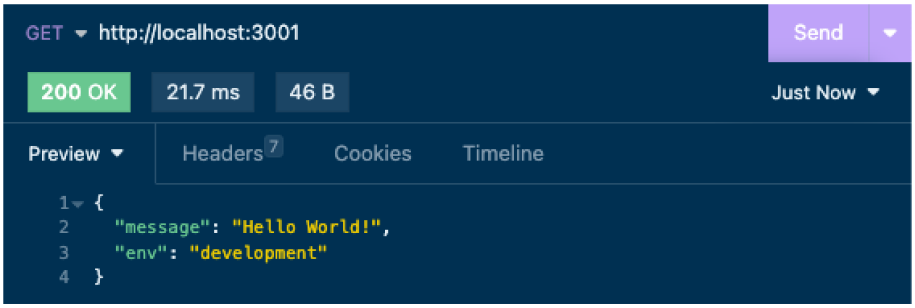
Figura 42 – Lista de *containers* em execução

```
→  node-api-example docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS                    NAMES
190b41692f35   node-api   "docker-entrypoint.s…"   41 minutes ago   Up 41 minutes   0.0.0.0:3001->3000/tcp   node-api-dev
d6d947a07746   node-api   "docker-entrypoint.s…"   41 minutes ago   Up 41 minutes   0.0.0.0:3002->3000/tcp   node-api-prod
```

Fonte: Próprio autor

Com as instâncias Docker *node-api-dev* e *node-api-prod* dos ambientes de desenvolvimento e produção, podem ser feitas requisições para cada servidor Web e verificado se as aplicações se comportam da maneira esperada. A Figura 43 mostra a resposta de uma requisição do tipo GET para a porta 3001 em que o ambiente de desenvolvimento está executando. É retornado o valor de *env* que só é enviado como resposta se o ambiente for de desenvolvimento. Nesse sentido, a variável de ambiente *NODE_ENV* recebeu corretamente o valor *development*. Já na resposta da requisição da Figura 44 é retornado somente a mensagem e não mais o valor da variável ambiente de execução.

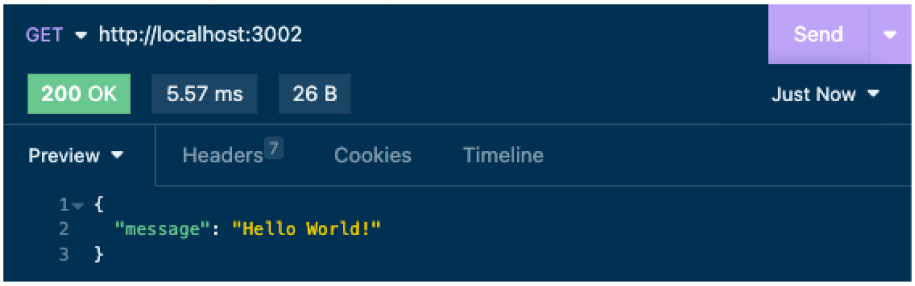
Figura 43 – Requisição GET para *node-api-dev*



Fonte: Próprio autor

O comando *stop* pode ser usado para parar um ou mais *containers* que se encontram em execução, ou seja, *status up*. Na Listagem 2.4.8, o comando é utilizado para parar as duas

Figura 44 – Requisição GET para *node-api-prod*



Fonte: Próprio autor

instâncias da imagem *node-api* que estavam em execução. Após isso, as instâncias passam a ter *status stopped*. Neste caso, é utilizado o nome do *container*, mas também poderiam ser usados os identificadores únicos *container id* que são mostrados na Figura 42.

```
docker stop node-api-dev node-api-prod
```

Listagem 2.4.8 – Parando os dois ambientes

Outrossim, com o intuito de voltar a executar um ou mais *containers* pausados, pode ser utilizado o comando *start* para que os *status* passem de *paused* para *up* novamente. Sabendo que esse comando funciona de forma similar ao *stop*, o comando *start* é exemplificado na Listagem 2.4.9, em que é retomada a execução do *container* de produção.

```
docker start node-api-prod
```

Listagem 2.4.9 – Retomando a execução do ambiente de produção

Já para deletar *containers* e imagens, são utilizados, respectivamente, os comandos *rm* e *rmi*. A Listagem 2.4.10 exemplifica a remoção da instância de desenvolvimento que já estava parada. Para que um *container* seja removido da forma exemplificada, é necessário que se encontre pausado. De mesmo modo, para deletar uma imagem de uma máquina é necessário que todas as instâncias da imagem sejam removidas. Neste sentido, é necessário primeiro pausar e depois remover o *container* de produção para que o comando da Listagem 2.4.11 seja executado com sucesso.

```
docker rm node-api-dev
```

Listagem 2.4.10 – Remoção da instância de desenvolvimento

```
docker rmi node-api-image
```

Listagem 2.4.11 – Remoção da imagem

2.5 WSO2 Identity Server: Gerenciamento de identidade e controle de acesso

O WSO2 Identity Server é uma ferramenta que facilita o acesso seguro a aplicativos utilizando um login único (WSO2, 2023e; KARUNARATHNA; KARUNARATNE, 2017). Por meio dessa solução, organizações conseguem gerenciar de forma eficiente identidades de seus usuários. Uma das vantagens do uso deste servidor de identidade é que, por ser de código aberto, oferece flexibilidade e personalização para atender as necessidades de cada empresa. Dois protocolos suportados para autenticação são o OpenID Connect e o OAuth 2.0. Além disso, o próprio WSO2 Identity Server também fornece conectores já prontos para ligar o WSO2 a bancos de dados com credenciais de usuários. Isso simplifica o processo de integração com sistemas já existentes.

Para servidores de identidade, recurso é a combinação de dados que podem ser acessados ou ações que podem ser realizadas em uma aplicação (AUTH0, 2023). Adicionalmente, autenticação é definida como o processo de verificação de identidade do usuário. Quanto ao termo autorização, é utilizado no contexto de verificar se o usuário pode ter acesso a um recurso protegido dentro de uma aplicação. Além disso, a autorização pode também ser empregada no contexto de delegar quais recursos do usuário uma aplicação tem acesso.

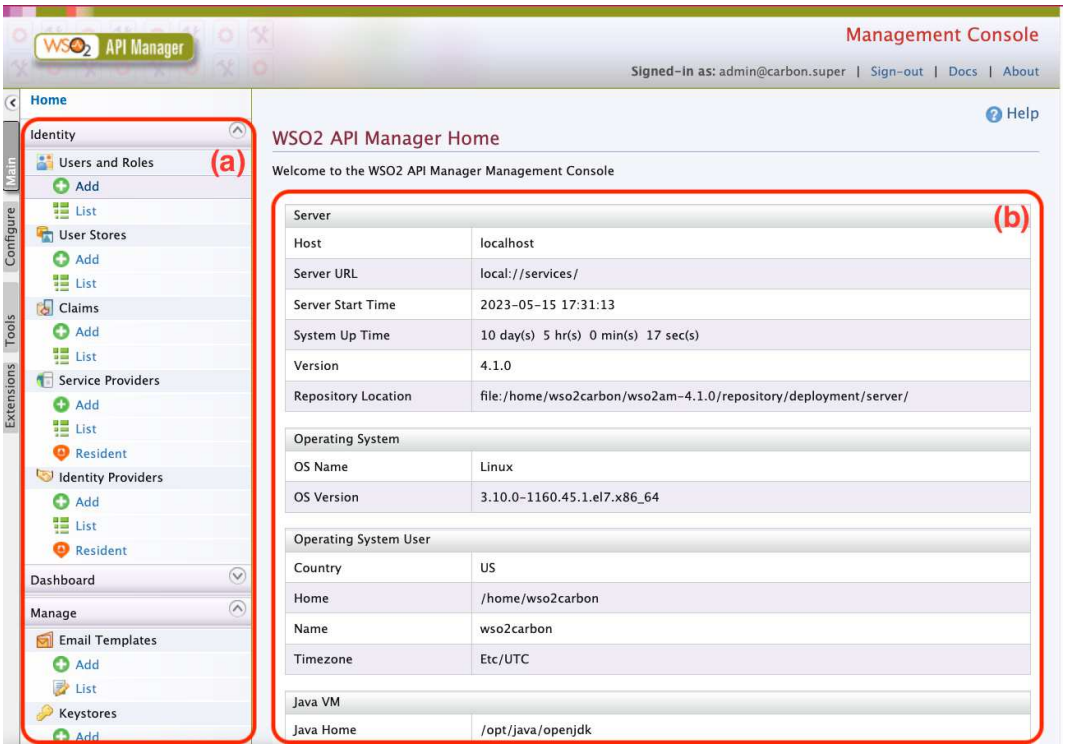
2.5.1 Instalação utilizando Docker

Para agilizar o processo de instalação do WSO2 API Manager, pode ser executado o comando da Listagem 2.5.1 para subir um *container* Docker. Com isso, a imagem é baixada, executada e em alguns instantes a aplicação fica disponível no endereço `https://localhost:9443/`
`carbon`. Como não foi feita nenhuma configuração de usuário, o login padrão é com usuário e senha sendo *admin*.

```
docker run -d -p 8280:8280 -p 8243:8243 -p 9443:9443 --name api-manager
↪  wso2/wso2am:latest
```

Listagem 2.5.1 – Comando para instalação do WSO2 API Manager utilizando Docker

Figura 45 – Tela inicial do WSO2 API Manager



Fonte: Próprio autor

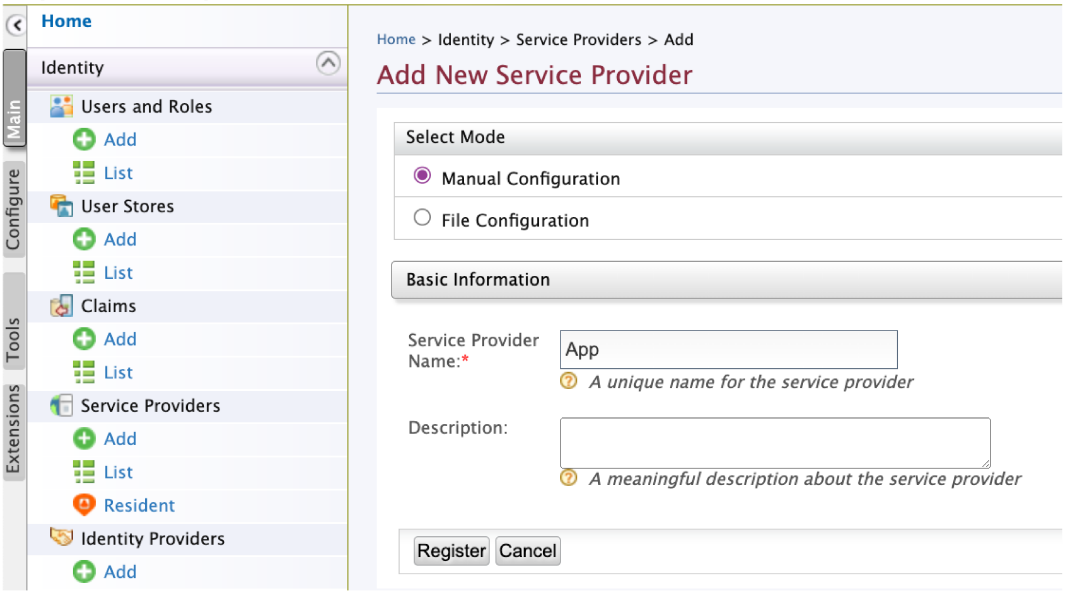
Ao entrar com as credenciais, é possível visualizar a tela inicial do API Manager, que é apresentada na Figura 45. Na imagem, foram destacadas em vermelho as partes **(a)** e **(b)** para discussão. Em **(a)**, existe uma barra lateral por meio do qual se navega nos menus de configuração do API Manager. Já em **(b)**, são mostradas informações relativas ao ambiente que o *container* está rodando, dois exemplos dessas informações são o endereço da máquina e também tempo que o sistema está rodando.

2.5.2 Configurar a conexão com uma aplicação

Qualquer entidade que fornece serviços Web é chamada por um provedor de identidade de *service provider* (WSO2, 2023b). Nesse sentido, para que uma aplicação use o WSO2 Identity Server é necessário realizar o cadastro do *service provider*. Por conseguinte, por meio do menu lateral apresentado previamente, é possível clicar na opção *add* para adicionar um novo *service provider*. Com isso, é acessado uma página para realizar o cadastro do novo *service provider*. Um fragmento dessa página é mostrado pela Figura 46. Nessa tela, deve ser escolhido um nome para o *service provider* e depois clicado para registrar.

Após o cadastro, o usuário é redirecionado para uma tela de configuração do *service provider*, um fragmento dessa página é mostrado na Figura 47. Porém, a imagem mostra o acesso

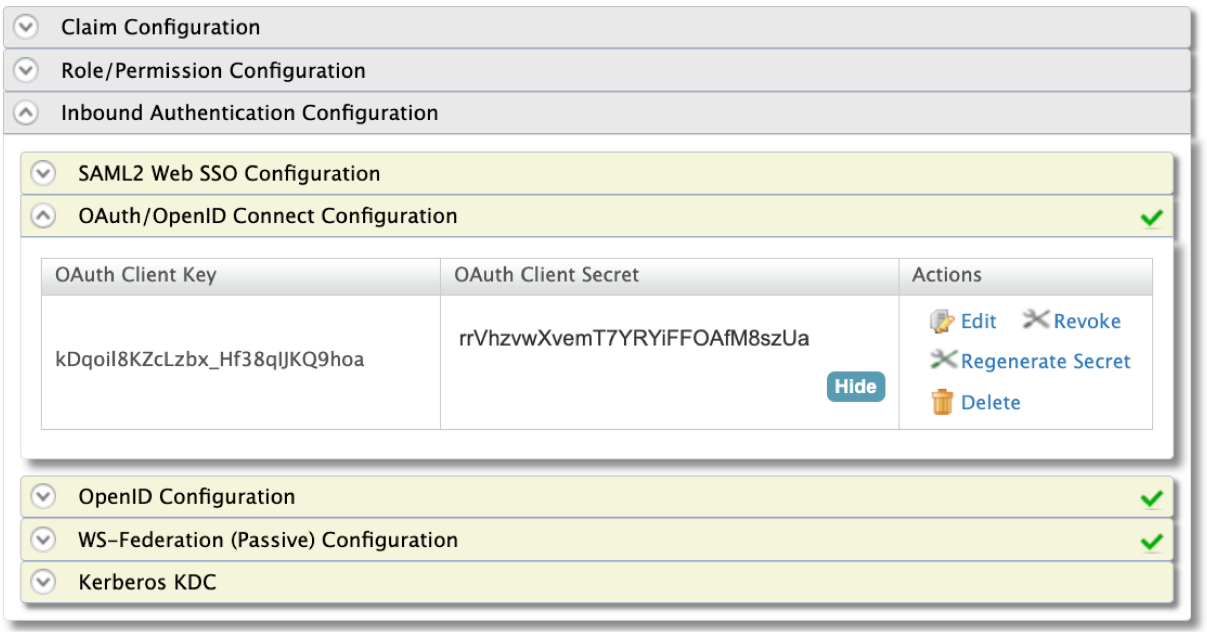
Figura 46 – Registro de *service provider*



Fonte: Próprio autor

ao menu *Inbound Authentication Configuration* e o menu aninhado *OAuth/OpenID Connect Configuration*. Na imagem, são mostradas as credenciais *client_id* e *client_secret* do *service provider*. Ainda na Figura 47, ao clicar *edit*, pode ser acessado as configurações de conexão do protocolo OIDC.

Figura 47 – Fragmento da tela de configuração do *service provider*

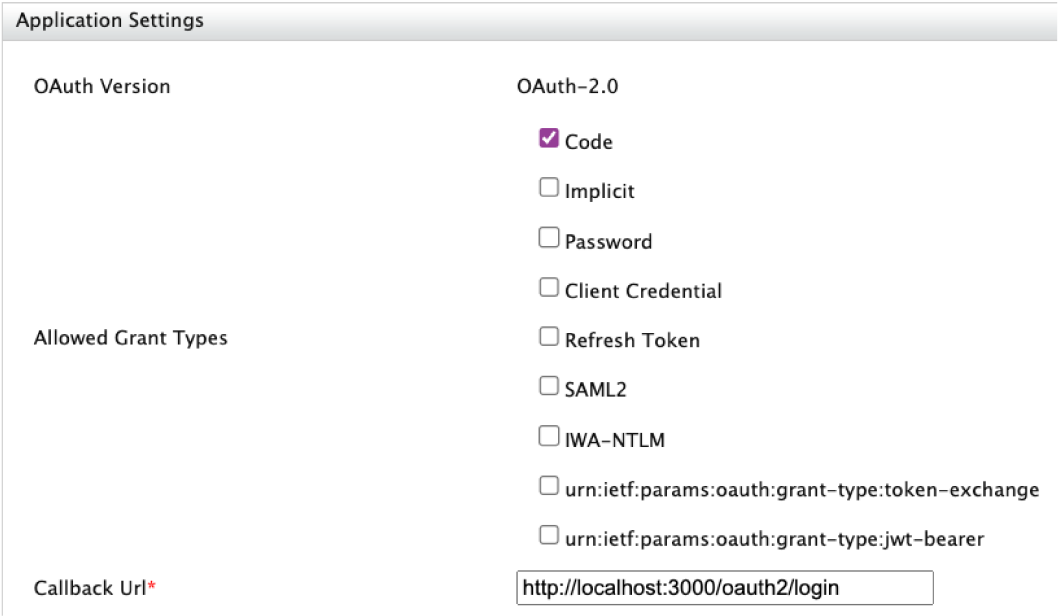


Fonte: Próprio autor

Um recorte da página de configurações de conexão do protocolo OIDC é mostrado na Figura 48. A configuração que será necessária consiste em marcar a versão do protocolo

OAuth como *code* e preencher uma URL de *callback* com uma rota para o *front-end* da aplicação.

Figura 48 – Menu de configuração do protocolo *OpenID Connect*



Fonte: Próprio autor

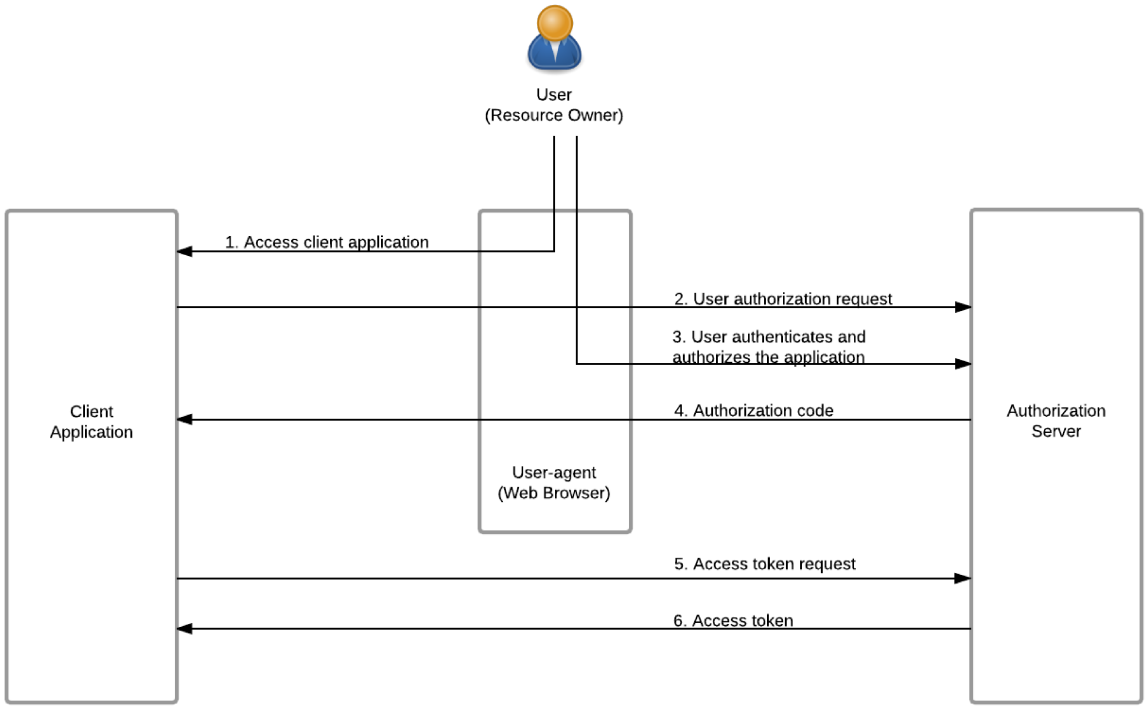
2.5.3 Conectando a uma aplicação

A Figura 49 é um digrama que visa ilustrar o fluxo de autenticação e autorização *authorization code* (WSO2, 2023a). Na imagem, são apresentadas as entidades usuário, aplicação cliente, servidor de autorização e também o navegador do usuário. Além disso, existem setas numeradas que ligam as entidades e cada seta descreve uma etapa no fluxo. As ações que são realizadas por meio de um navegador tem as setas que as descrevem passando pelo navegador do usuário na ilustração. Isto posto, as ações 4 e 5 não são feitas por meio de um navegador, pois suas setas não passam por essa entidade na ilustração.

No âmbito da Figura 49, em 1 o usuário acessa a aplicação por meio de um navegador. Já na seta 2, a aplicação cliente requisita ao servidor de autorização acesso aos dados do usuário. Para obter acesso aos dados do usuário é necessário que o usuário esteja autenticado. Por isso, na etapa 3 o usuário digita suas credenciais para se autenticar em uma tela de login do servidor de autorização. Na etapa 3, o usuário também autoriza acesso da aplicação aos recursos requisitados por meio de uma tela de consentimento. Caso a autenticação e a autorização sejam bem sucedidas, a aplicação recebe um parâmetro *code* que é apresentado pela seta 4. Sendo assim, o parâmetro recebido é utilizado para solicitar um *token* de acesso, como descrito por 5. Por fim, caso o parâmetro *code* seja válido, o servidor de autorização retorna um *token* de acesso,

mostrado em 6.

Figura 49 – Ilustração do fluxo *authorization code*



Fonte: WSO2 (2023a)

Os processos realizados no diagrama da Figura 49 são feitos por meio de requisições HTTP que devem ser feitas diretamente no navegador, por meio de redirecionamentos, ou também por requisições AJAX. A Tabela 1 apresenta as requisições que devem ser feitas, o método HTTP utilizado, a rota que é empregada e também uma breve descrição. As requisições dessa tabela são discutidas a seguir.

Tabela 1 – Requisições HTTP do fluxo *authorization code*

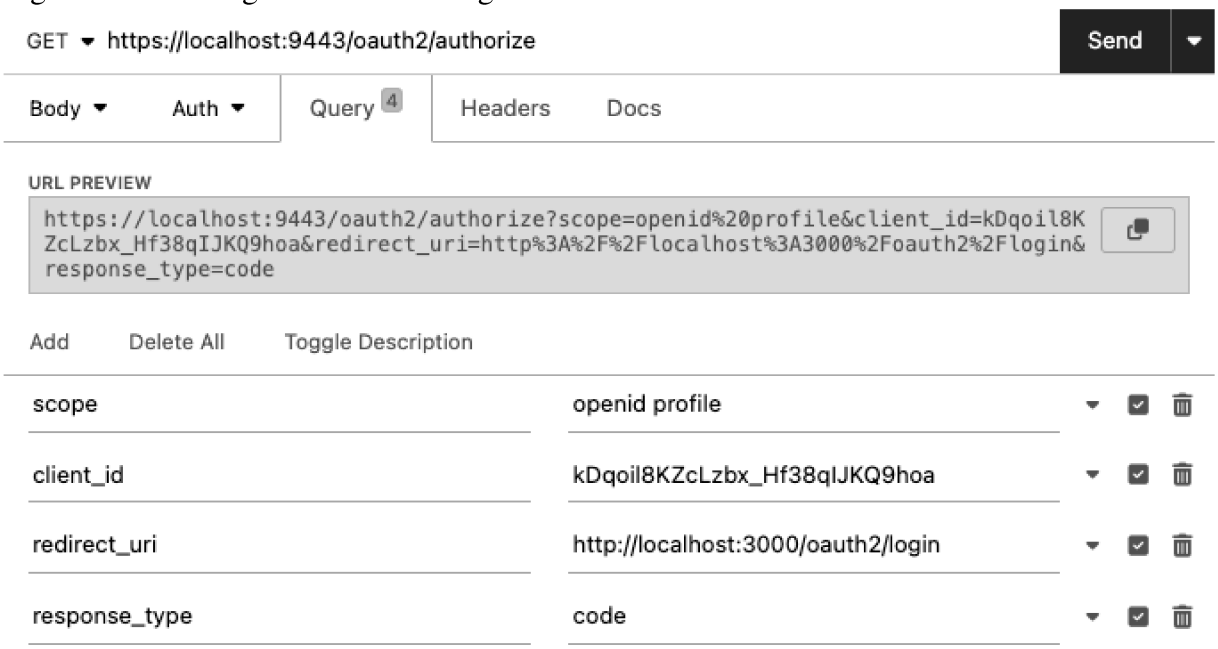
Requisição	Método	Rota	Descrição
1	GET	/oauth2/authorize	Utilizado para acessar a tela de login
2	POST	/oauth2/token	Solicitar o token de acesso
3	POST	/oauth2/userinfo	Requisitar informações do usuário

Fonte: Próprio autor

Com as credenciais do *service provider*, é possível montar uma URL que dá acesso a página de login do WSO2, utilizando a rota da requisição 1 da Tabela 1. A URL de login deve ser acessada por meio de um navegador qualquer e, inicialmente, o login pode ser feito usando o mesmo usuário utilizado para entrar no painel de configuração do WSO2 API Manager. A montagem da URL de login é ilustrada pela Figura 50, em que o *scope* é relativo ao escopo de

dados que se quer obter após a finalização do login. Já o *client_id*, é uma parte das credenciais da aplicação. A *redirect_uri* é a URL de redirecionamento, que deve ser a mesma fornecida na configuração do *service provider*, descrita na Subseção 2.5.2. Por fim, o *response_type* refere-se ao formato da resposta esperada pela aplicação cliente, no caso do fluxo *authorization code* a resposta esperada após o login é o *code*.

Figura 50 – Montagem da URL de login



Fonte: Próprio autor

Ao acessar por meio de um navegador a URL de login montada, o usuário é redirecionado para a página de login, mostrada na Figura 51. Após entrar com as credenciais e consentir com o acesso do escopo de dados solicitado pela aplicação, o usuário é redirecionado para a *redirect_uri*, que é uma rota dentro do *front-end* da aplicação.

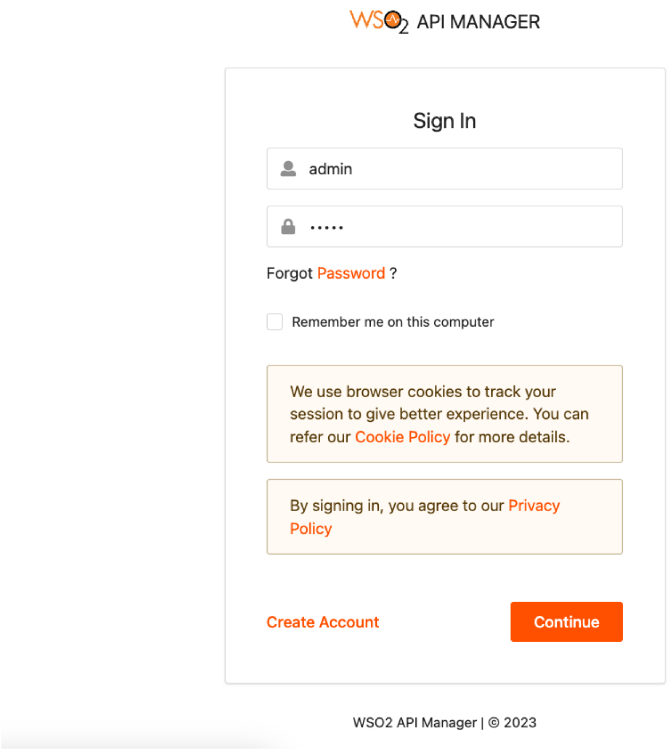
A Listagem 2.5.2 mostra o formato da URL do redirecionamento feito após o login, em que o parâmetro *url-redirecionamento* destacado se refere ao *redirect_uri*. Além disso, também é mostrado que o *front-end* da aplicação recebe por meio da URL os parâmetros *code* e *session_state*.

```
1 <url-redirecionamento>?code=<code>&session_state=<session-state>
```

Listagem 2.5.2 – Formato da URL de retorno após login com WSO2

É ilustrada na Figura 52 a requisição 2 da Tabela 1, que é feita utilizando o parâmetro *code*, as credenciais do *service provider*, *redirect_uri* e *grant_type*. O *grant_type* define o fluxo de

Figura 51 – Página de login da aplicação

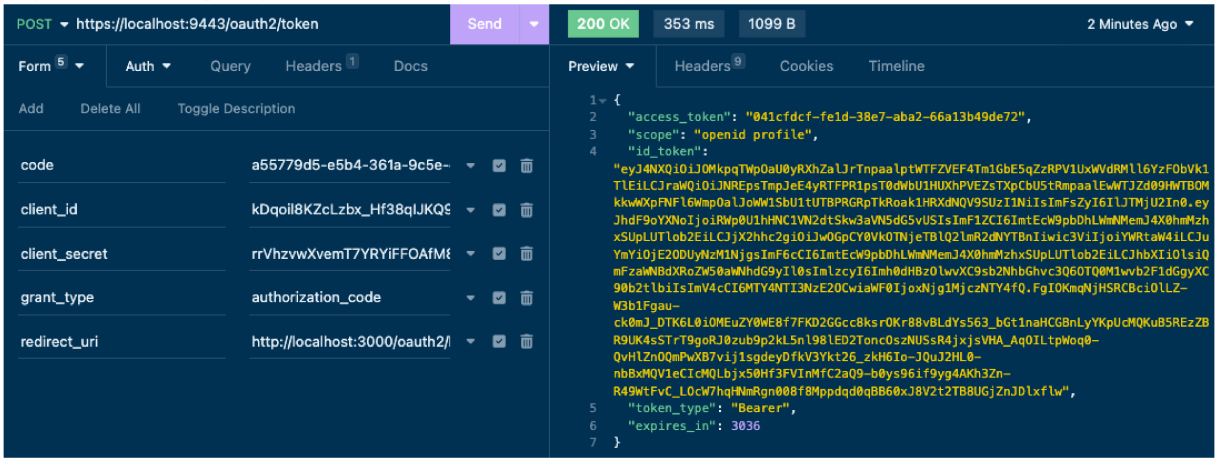


Fonte: Próprio autor

autenticação utilizado. Nesse caso, o fluxo de autenticação é o *authorization_code*, em que se tem um *code* e a partir dele é requisitado um *token* de acesso. Os parâmetros dessa requisição são passados na forma de *Form URL Encoded*. Sendo assim, ao passar os parâmetros pela URL é necessário também enviar o cabeçalho `Content-Type: application/x-www-form-urlencoded`. Após a requisição, o *code* enviado não pode ser utilizado novamente. Em contrapartida, o *token* de acesso recebido como resposta da requisição, que é apresentado na Figura 52 com nome *access_token*, pode ser utilizado várias vezes e sua duração é definida pelo servidor de autorização.

A requisição 3 da Tabela 1 é importante, pois além de mostrar os dados do usuário, é por meio desta que a aplicação consegue saber se o usuário está devidamente autenticado, essa requisição é apresentada na Figura 53. Para ser realizada, é necessário passar um cabeçalho de nome *Authorization* e valor `Bearer <access-token>`. A resposta pode ser maior ou menor dependendo do escopo de dados solicitado durante o processo login (WSO2, 2023c). No caso da Figura 53, a resposta da requisição foi somente o nome de usuário. Se a requisição falhar porque o *token* de acesso está inválido, é necessário fazer o login novamente e conseguir um novo *token* de acesso. Já caso falhe porque o *token* se encontra expirado, é possível implementar o fluxo opcional de *refresh_token* para que a aplicação receba um novo *token* válido sem que o usuário

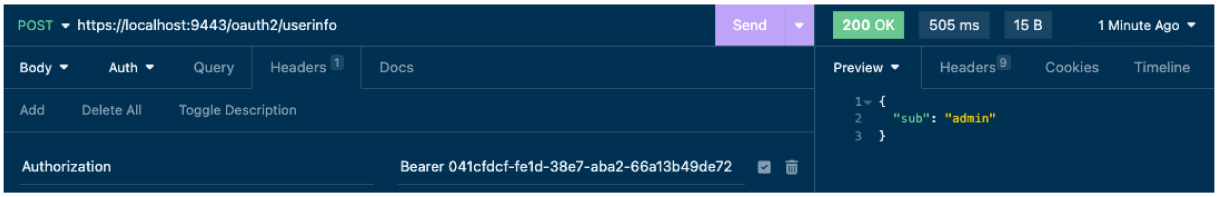
Figura 52 – Requisição do token de acesso



Fonte: Próprio autor

tenha que entrar novamente na aplicação (WSO2, 2023d).

Figura 53 – Requisição de informações do usuário



Fonte: Próprio autor

3 METODOLOGIA

Neste capítulo será apresentada a metodologia empregada para o desenvolvimento deste trabalho. O capítulo está dividido em seções sobre as perguntas de pesquisa (Seção 3.1), etapas de condução do estudo (Seção 3.2), estudo piloto (Seção 3.3), protocolo de entrevistas (Seção 3.4) e configuração das entrevistas (Seção 3.5). Adicionalmente, é feita uma descrição sobre o processamento dos dados (Seção 3.6) e a análise dos dados (Seção 3.7).

3.1 Questões de pesquisa

Conforme descrito na introdução deste trabalho (Capítulo 1), o presente estudo se propõe a preencher a lacuna existente na literatura relacionada com a escolha e caracterização das ferramentas que apoiam o desenvolvimento Web. Para isso, foram formuladas três perguntas de pesquisa alinhadas com esse objetivo. Essas perguntas foram respondidas com base nos dados extraídos no processo de entrevistas com os desenvolvedores, permitindo uma análise das ferramentas utilizadas.

QP1: Quais são as ferramentas que apoiam o desenvolvimento Web? Essa pergunta foi elaborada com o intuito de delimitar o escopo das ferramentas que apoiam o desenvolvimento Web e listar o nome dessas ferramentas com base nas respostas das perguntas de entrevista.

QP2: É possível categorizar as ferramentas de acordo com seu uso? Essa pergunta busca categorizar as ferramentas mencionadas durante as entrevistas conforme suas funcionalidades e áreas de aplicação no desenvolvimento Web. Além disso, investiga-se se algumas ferramentas foram mencionadas em mais de uma categoria de uso, indicando uma versatilidade no emprego dessas ferramentas em diferentes etapas e processos dentro do desenvolvimento de *software*.

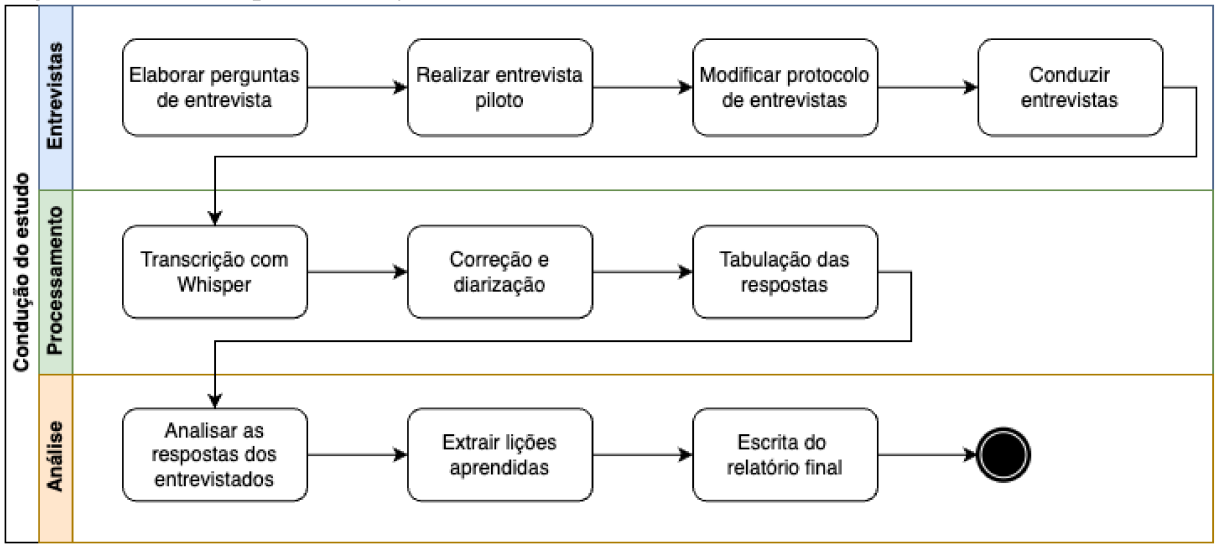
QP3: Quais a ferramentas mais citadas para apoiar as etapas do desenvolvimento Web? Com essa pergunta, procura-se identificar quais a ferramentas mais citadas pelos entrevistados para apoiar as etapas do desenvolvimento Web. Algumas dessas ferramentas foram destacadas na fundamentação teórica (Capítulo 2). Nesse sentido, com os resultados da fase de entrevistas, faz-se necessário avaliar suas frequências de uso, de modo a entender o quão relevante essas ferramentas são para os desenvolvedores entrevistados.

3.2 Etapas de condução do estudo

O digrama apresentado na Figura 54 fornece uma visão geral do processo de entrevistas, que foi dividido em três principais etapas. A primeira etapa foi o das entrevistas, que consiste no processo para a elaboração de um protocolo e realização das entrevistas. Em seguida, o processamento dos dados, que envolveu a transcrição do áudio das entrevistas, correção e tabulação das respostas. Por fim, a análise dos dados coletados, que permitiu extrair as lições aprendidas.

O detalhamento de cada uma dessas etapas é abordado nas seções seguintes deste capítulo. A Seção 3.3 descreve o estudo piloto realizado, enquanto a Seção 3.4 detalha o protocolo de entrevistas e a Seção 3.5 cobre a configuração das entrevistas. A descrição dos passos para o processamento dos dados é apresentada na Seção 3.6, e a análise dos dados é abordada na Seção 3.7.

Figura 54 – Passos para condução do estudo



Fonte: Próprio autor

3.3 Estudo piloto

Inicialmente, foram elaboradas algumas perguntas para direcionar as entrevistas para o estudo. No entanto, antes de conduzir as entrevistas principais, foi realizado um estudo piloto com o objetivo de validar o protocolo de entrevista idealizado. Esse estudo foi importante para avaliar não apenas o tempo necessário para cada entrevista, mas também a clareza das perguntas, a forma de condução das entrevistas e a qualidade dos dados coletados. O estudo piloto contou

com a participação de um Engenheiro de Dados com 3 anos de experiência profissional e experiência em desenvolvimento Web.

Após realizada a entrevista piloto, que durou 15 minutos, estimou-se que as outras entrevistas iriam durar esse mesmo tempo. Com base nos resultados obtidos, as perguntas foram reformuladas. Além disso, foi identificada a necessidade de fornecer uma contextualização inicial sobre o propósito do trabalho para que os entrevistados compreendessem melhor o objetivo das entrevistas e conseguissem responder as perguntas de forma mais objetiva.

3.4 Perguntas de entrevista e formulário de perfil

As perguntas da Tabela 2 foram elaboradas para serem realizadas durante a fase de entrevistas. As perguntas feitas nas entrevistas foram organizadas em quatro tópicos comuns dentro de projetos de software: coordenação de atividades, controle de versão, ferramentas auxiliares e integração contínua e entrega contínua (CI/CD). Cada pergunta foi indexada com um identificador que segue o formato "X.Y", onde "X" representa o número do tópico ao qual a pergunta pertence, e "Y" indica a ordem da pergunta dentro desse tópico.

Tabela 2 – Perguntas de entrevista

Tópico	Identificador	Pergunta
Coordenação de atividades	1.1	Onde fica seu *backlog* e como é feita a organização das tarefas?
	1.2	Quais plataformas sua equipe utiliza para conversar durante o processo de desenvolvimento?
	1.3	Vocês utilizam alguma ferramenta específica de documentação?
Controle de versão	2.1	Vocês utilizam Git para o controle de versão?
	2.2	Qual servidor remoto de Git vocês utilizam?
Ferramentas auxiliares	3.1	Quais *softwares* comerciais ou de código aberto que conversam com a aplicação que sua equipe desenvolve? Se aplicável, cite serviços de mensageria, *engines* de busca, serviços de federação de identidade e outras ferramentas utilizadas por sua equipe.
CI/CD	4.1	Vocês utilizam alguma ferramenta para automatizar o CI/CD?
	4.2	Vocês utilizam contêineres para a implantação da aplicação? Se sim, estes *containers* são Docker *containers*?
	4.3	É utilizado algum serviço de orquestração de *containers*?

Fonte: Próprio autor

Foi elaborado também um formulário de perfil do entrevistado, cujas perguntas foram apresentadas na Tabela 3. Esse formulário foi montado com intuito de extrair informações para caracterizar o perfil do entrevistado, sendo composto de perguntas de múltipla escolha e também perguntas abertas.

Tabela 3 – Formulário de caracterização do entrevistado

Identificador	Pergunta
1	Qual seu cargo profissional no momento? (Pergunta aberta)
2	Quanto tempo de experiência profissional você tem como desenvolvedor para a Web? [ ] Menos de 1 ano [ ] De 1 a 3 anos [ ] De 3 a 5 anos [ ] Mais de 5 anos
3	Quais dos papeis a seguir você já desempenhou em uma equipe? [ ] Desenvolvedor [ ] Gerente de Produto [ ] Scrum Master [ ] Gerente de Projetos [ ] Analista de Qualidade [ ] DevOps [ ] Outros (Campo aberto para adicionar outros papeis)
4	Deseja fazer algum comentário sobre a entrevista realizada?

Fonte: Próprio autor

3.5 Configuração das entrevistas

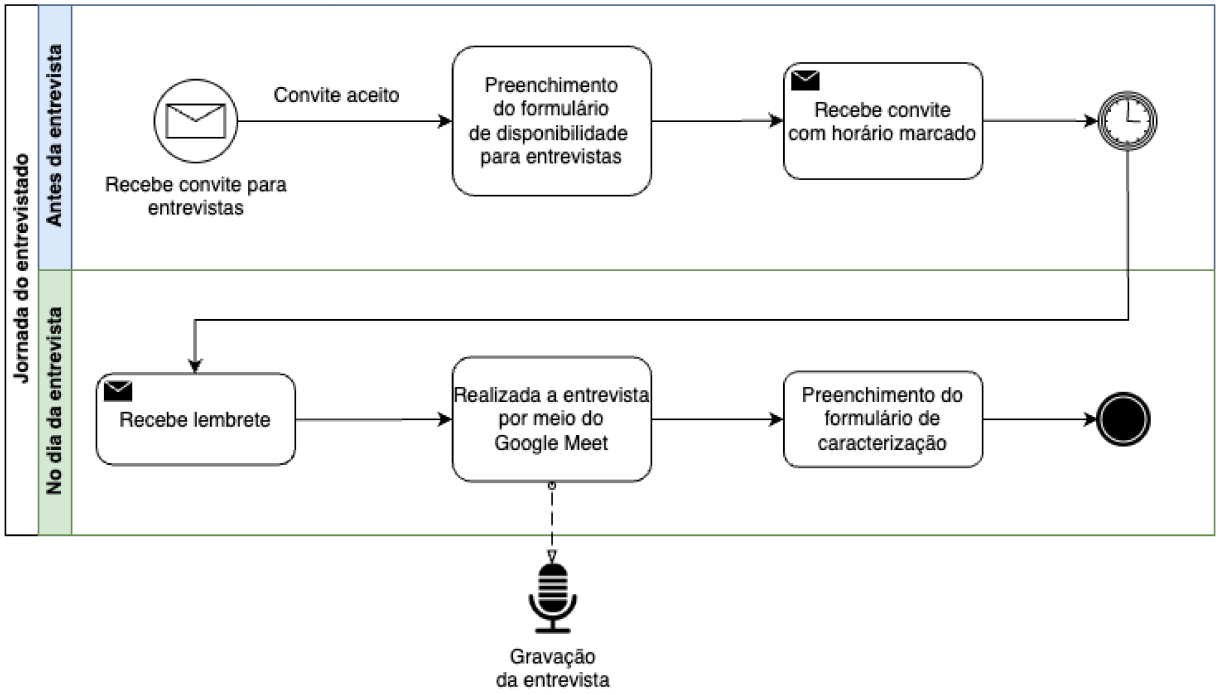
A fase de entrevistas foi iniciada em abril de 2024 e concluída em agosto de 2024. Durante esse período, foram enviados 15 convites e 9 desenvolvedores foram entrevistados. A Figura 55 ilustra a jornada de um entrevistado que participa do presente estudo. Conforme está descrito no diagrama, após confirmar o interesse em participar da entrevista, o entrevistado deveria preencher um formulário de convite, criado por meio do Google Forms[1].

O formulário de convite incluía campos para o nome, *e-mail* e a disponibilidade de horários ao longo da semana. Após o preenchimento do formulário a entrevista era então agendada por meio do Google Agenda[2] e o convite enviado para o *e-mail* fornecido, contendo o link para a sala virtual da entrevista. Para cada entrevistado, foi agendada uma janela de 30 minutos, a fim de acomodar possíveis atrasos e imprevistos.

Ainda na Figura 55, é apresentado que no dia da entrevista o entrevistado recebia um lembrete com o horário da entrevista. Com isso, as entrevistas foram conduzidas de forma completamente remota por meio da plataforma do Google Meet[3] e tiveram seu áudio gravado para análise posterior. Ao fim de cada entrevista, o entrevistado preencheu o formulário de caracterização do participante, apresentado na Seção 3.4.

[1] https://www.google.com/intl/pt-BR/forms/about
[2] https://calendar.google.com
[3] https://meet.google.com

Figura 55 – Ilustração da jornada do entrevistado



Fonte: Próprio autor

3.6 Processamento dos dados

O áudio de cada uma das entrevistas foi transcrito por completo utilizando o modelo Whisper Large V2 da OpenAI (RADFORD *et al.*, 2022). Após isso, a transcrição passou por um processo manual de diarização e também de correção de erros de transcrição. O processo de diarização é necessário para separar o que cada interlocutor da entrevista disse. Durante esse processo, cada participante foi associado a um identificador único sequencial, que foi utilizado para identificar tanto as falas do participante na transcrição quanto os demais dados relacionados ao entrevistado.

Em uma análise posterior das transcrições, as respostas das entrevistas foram tabuladas de modo que fosse possível identificar quais ferramentas cada entrevistado citou como resposta para uma dada pergunta. Esse processo foi realizado para facilitar a análise das respostas. Além disso, as informações pessoais coletadas para identificar o entrevistado durante o processo das entrevistas e no formulário de caracterização do participante (como nome e *e-mail*) foram substituídas pelo um número identificador atribuído ao participante. Dessa forma, na fase de análise dos dados os participantes que colaboraram com as entrevistas têm sua identidade preservada.

3.7 Análise dos dados

Para realizar a análise dos dados, foram utilizados Notebooks dentro do ambiente do Google Colab[4], uma plataforma baseada em nuvem que facilita o trabalho colaborativo e a execução interativa de código Python. Além disso, foram utilizadas as bibliotecas Matplotlib[5] e Seaborn[6] para a criação de gráficos, enquanto a biblioteca Wordcloud[7] foi empregada para gerar nuvens de palavras e a biblioteca Plotly[8] foi utilizada para a criação da visualização do tipo *Sunburst*.

Em um primeiro Notebook, foi realizada uma análise exploratória dos dados estruturados em tabelas, incluindo as respostas dos entrevistados para as perguntas de entrevista e as informações do formulário de caracterização dos participantes. Essa etapa permitiu identificar padrões de uso das ferramentas e entender as características dos desenvolvedores entrevistados.

O segundo Notebook foi dedicado ao processamento das transcrições brutas das entrevistas. Com o auxílio da biblioteca NLTK[9], foi feita a remoção de *stopwords* e selecionados apenas os substantivos, além de serem normalizadas as palavras no plural. Esse processamento permitiu analisar as falas dos participantes e criar visualizações relevantes, destacando os termos mais recorrentes nas entrevistas.

[4] https://colab.research.google.com
[5] https://matplotlib.org
[6] https://seaborn.pydata.org
[7] https://pypi.org/project/wordcloud
[8] https://plotly.com
[9] https://www.nltk.org

4 RESULTADOS

O presente estudo foi conduzido por meio de entrevistas realizadas entre abril e agosto de 2024 com 9 desenvolvedores que atuam no mercado de desenvolvimento Web no Brasil. Após a condução do estudo e a subsequente análise dos dados coletados por meio das entrevistas, como detalhado no Capítulo 3, os resultados obtidos serão apresentados neste capítulo. Inicialmente é discutido o perfil do participante das entrevistas (Seção 4.1), seguido pela discussão das perguntas de pesquisa nas Seções 4.2, 4.3 e 4.4. Por fim, é feito um comparativo com uma pesquisa de mercado (Seção 4.5) e são listadas as lições aprendidas com os resultados das entrevistas (Seção 4.6).

4.1 Perfil do participante

O perfil dos entrevistados do presente estudo é diversificado, sendo todos desenvolvedores com experiência profissional em desenvolvimento para a Web. A Tabela 4 apresenta uma visão geral do perfil dos entrevistados, em que cada linha contém um número identificador do entrevistado, o cargo profissional que ocupava no momento da entrevista, o tempo de experiência profissional com desenvolvimento Web e os papeis que o entrevistado já desempenhou em uma equipe de desenvolvimento de *software*.

Conforme mencionado anteriormente, após a entrevista, cada entrevistado preencheu um fomulário de perfil, (apresentado na Seção 3.4). Os dados exibidos na Tabela 4 foram retirados das respostas do formulário de caracterização do entrevistado. Vale destacar que as siglas utilizadas na coluna “Papeis desempenhados” da Tabela 4 representam: Desenvolvedor (DEV), Gerente de Produto (GP), Scrum Master (SM), Gerente de Projetos (GPJ), Analista de Qualidade (AQ) e DevOps (DO).

Tabela 4 – Formulário de caracterização do entrevistado

Identificador	Cargo profissional	Tempo de experiência	Papeis desempenhados
E01	Desenvolvedor FullStack	De 3 a 5 anos	DEV,AQ
E02	Programador	De 1 a 3 anos	DEV,GPJ
E03	Desenvolvedor Web	Menos de 1 ano	DEV
E04	Desenvolvedor Júnior	De 1 a 3 anos	DEV
E05	Gerente de Projetos	Mais de 5 anos	DEV,GPD,SM,GPJ
E06	Desenvolvedor Back-End	De 3 a 5 anos	DEV,GPJ
E07	Desenvolvedor FullStack	De 1 a 3 anos	DEV,DO
E08	Desenvolvedor Web	De 1 a 3 anos	DEV
E09	Desenvolvedor Web	De 1 a 3 anos	DEV

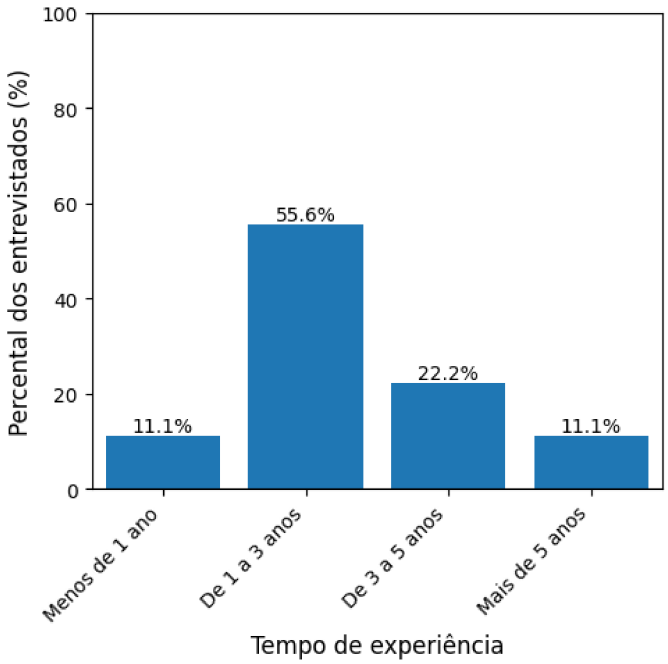
Fonte: Próprio autor

Os entrevistados, cujos dados foram apresentados na Tabela 4, trabalham em empresas distintas, exceto nos casos do “E05”, “E08” e um dos projetos mencionados pelo “E07”, que pertencem à mesma organização. Observa-se que a maioria dos entrevistados ocupam posições diretamente relacionadas ao desenvolvimento de *software*, como Desenvolvedor FullStack, Desenvolvedor Web e Programador.

Ainda no contexto da Tabela 4, é interessante notar a presença de um entrevistado que atua como Gerente de Projetos (E05), que tem mais de cinco anos de experiência profissional, indicando que o estudo também inclui a perspectiva de um profissional em posição de liderança. Além disso, é importante destacar que, exceto o desenvolvedor cujo cargo é especificamente voltado para o *back-end*, os outros entrevistados trabalhavam tanto com *front-end* quanto com *back-end*.

O gráfico da Figura 56 apresenta a distribuição do tempo de experiência dos entrevistados. Sabe-se que dos nove entrevistados, aproximadamente 55,6% possuem entre um e três anos de experiência profissional, indicando que a maioria está em um momento intermediário de sua carreira. Além disso, apenas um entrevistado (11,1% do total) ingressou no mercado de trabalho há menos de um ano, o que sugere que a maior parte dos participantes já possui experiência prática consolidada no desenvolvimento Web.

Figura 56 – Tempo de experiência dos entrevistados



Fonte: Próprio autor

Por outro lado, o gráfico da Figura 56 também revela que aproximadamente 33,3%

dos entrevistados possuem mais de três anos de experiência, sendo que 22,2% têm entre três e cinco anos, e um entrevistado (11,1%) tem mais de cinco anos de atuação no mercado. Com a entrevista desses profissionais mais experientes, espera-se obter uma perspectiva consolidada, proveniente do tempo extra de prática no desenvolvimento Web.

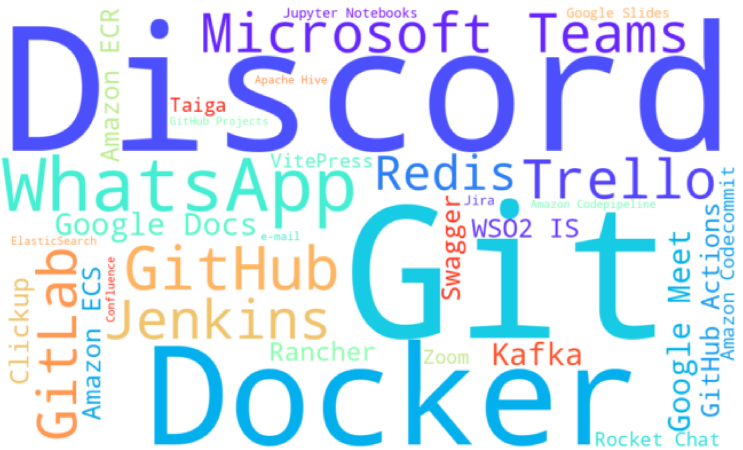
É importante destacar que todos os desenvolvedores entrevistados estavam atuando no mercado de trabalho. Embora a maioria esteja em um estágio intermediário de sua carreira, essa diversidade de experiências pode influenciar as ferramentas utilizadas e as práticas adotadas, uma vez que as escolhas e percepções tendem a variar conforme o tempo de atuação profissional.

4.2 QP1: Quais são as ferramentas que apoiam o desenvolvimento Web?

Uma nuvem de palavras montada por meio das ferramentas citadas pelos entrevistados é apresentada na Figura 57. Nessa visualização, quanto maior a palavra, mais vezes essa ferramenta foi citada pelos entrevistados. Dessa forma, espera-se destacar as principais ferramentas citadas durante as entrevistas, oferecendo uma visão geral das tendências e preferências de uso das ferramentas.

Ao todo, foram mencionadas pelos entrevistados 34 ferramentas distintas, refletindo a diversidade de soluções utilizadas no desenvolvimento Web. Analisando a Figura 57, percebe-se que as ferramentas com mais recorrência são Git, Discord e Docker. Essa predominância se deve à sua grande importância nos ecossistemas de desenvolvimento dos entrevistados, o que refle sua ampla adoção e relevância para esses profissionais.

Figura 57 – Nuvem de palavras das ferramentas citadas pelos entrevistados



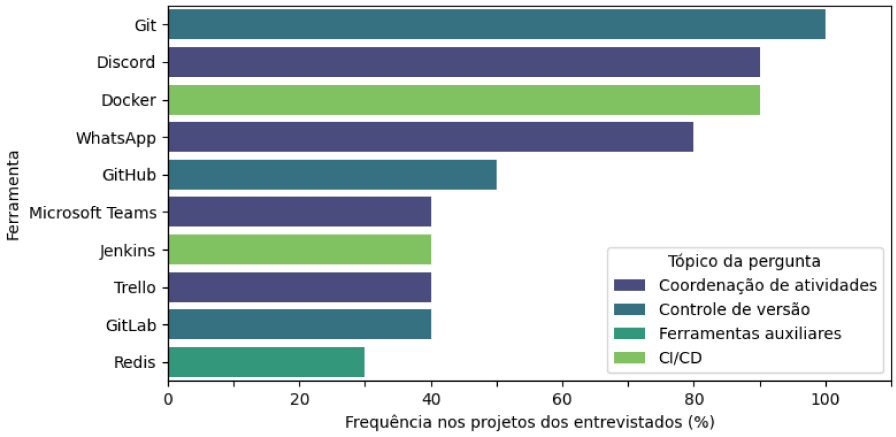
Fonte: Próprio autor

É apresentado na Figura 58 um gráfico de barras que ilustra as 10 ferramentas mais frequentes nas respostas dos entrevistados. É importante notar que, apesar de ter entrevistado 9 desenvolvedores, as respostas foram segmentadas de acordo com os projetos que eles estavam envolvidos. Isso acontece, porque o entrevistado "E07" forneceu informações sobre dois projetos distintos. Como resultado, a frequência total das menções ultrapassa o número de entrevistados, totalizando 10 projetos.

A ampla maioria dos projetos dos entrevistado (com 80% ou mais citações entre os 10 projetos citados) utiliza Git, Discord, Docker e WhatsApp, reiterando a importância dessas ferramentas para os entrevistados. Além das frequências de menção, isso revela uma aceitação generalizada entre a amostra selecionada. O destaque dessas ferramentas com elevado número de citações indica uma uniformidade no uso e na preferência entre os profissionais, fato que evidencia seu impacto nas práticas de desenvolvimento e colaboração.

Ainda na Figura 58, foi apresentada uma legenda que divide as ferramentas de acordo com o tópico da pergunta para a qual ela foi citada como resposta. As perguntas que fazem parte dos tópicos presentes na legenda foram apresentadas na Seção 3.4, que aborda o protocolo de entrevistas utilizado. Nesse protocolo, é apresentado as perguntas de entrevistas, que foram divididas nas categorias de coordenação de atividades, controle de versão, ferramentas auxiliares e integração contínua e entrega contínua (CI/CD).

Figura 58 – Gráfico das 10 ferramentas mais usadas nos projetos



Fonte: Próprio autor

A Figura 59 apresenta um gráfico de barras que ilustra a distribuição das 34 ferramentas únicas mencionadas nas respostas às perguntas de entrevista, categorizadas entre os quatro tópicos principais. Essa visualização tem como objetivo extrair informações sobre o escopo das ferramentas citadas pelos entrevistados, destacando como elas são utilizadas para

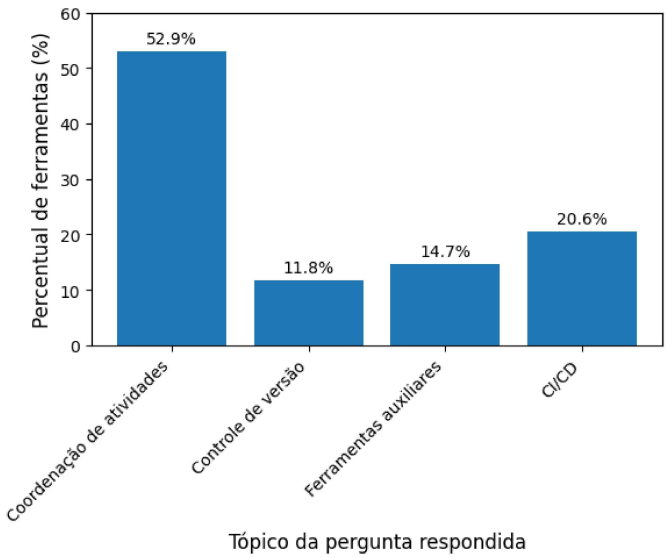
apoiar e otimizar o processo de desenvolvimento Web.

Em relação aos dados apresentados na Figura 59, cerca de 52,9% das ferramentas apresentadas pelos entrevistados concentram-se na coordenação de atividades. Esse alto percentual reflete a ênfase na gestão de projetos e na organização de tarefas no contexto do desenvolvimento Web. Além disso, 20,6% das ferramentas citadas estão associadas à práticas de integração contínua e entrega contínua (CI/CD). Ferramentas auxiliares representam 14,7% das menções, enquanto 11,8% referem-se a ferramentas relacionadas com o controle de versão.

Para finalizar a análise dos dados da Figura 59, é necessário enfatizar que um pequeno percentual de ferramentas únicas mencionadas em um tópico de ferramentas não necessariamente indica que têm uma baixa importância para os desenvolvedores entrevistados. Na verdade, isso pode refletir a presença de um mercado consolidado com um número reduzido de opções predominantes. A consolidação e a ampla adoção de ferramentas específicas podem resultar em uma menor diversidade mencionada pelos entrevistados.

Apesar da baixa quantidade de ferramentas únicas mencionadas que tratam sobre controle de versão, o entendimento de que o mercado dessas ferramentas é consolidado é corroborado pelo dado apresentado anteriormente na Figura 58, onde o Git foi mencionado em 100% dos projetos comentados pelos entrevistados, e GitHub e GitLab, combinados, foram utilizados em 90% dos projetos. Assim, para entender melhor essas informações e a importância dada pelos desenvolvedores para as ferramentas em cada tópico, é essencial analisar a frequência de uso por projeto, o que será detalhado na Seção 4.4.

Figura 59 – Percentual de ferramentas únicas respondidas por tópico



Fonte: Próprio autor

A Figura 60 apresenta uma nuvem de palavras criada com base nas transcrições das entrevistas. Para a construção dessa nuvem, foram selecionadas exclusivamente as falas dos entrevistados, destacando os substantivos mais frequentemente mencionados. Nomes de ferramentas foram excluídos para refletir somente os pontos centrais ainda não avaliados sobre as entrevistas. Esta visualização oferece uma representação gráfica das palavras mais comuns, permitindo identificar os termos e conceitos que mais se destacaram nas discussões dos participantes.

Na nuvem de palavras Figura 60, observa-se um predomínio de palavras relacionadas ao produto final, como “projeto”, “sistema” e “ ferramenta”, indicando que os entrevistados colocam uma ênfase considerável no aspecto do produto e sua gestão. Além disso, a presença de termos relacionados a pessoas, como “equipe” e “pessoal”, sugere que as dinâmicas de equipe e a colaboração são aspectos fundamentais para os entrevistados. Os termos com maior destaque refletem um foco significativo nas práticas corporativas, evidenciando uma preocupação com a organização e os processos dentro dos projetos.

Figura 60 – Nuvem de palavras das falas dos entrevistados

Fonte: Próprio autor

Nesse contexto, ambos os aspectos se alinham com as metodologias ágeis, no sentido de ter o foco no produto e desenvolvimento de forma eficiente, promovendo uma adaptação contínua e a entrega de soluções que atendam às necessidades do cliente. A ênfase em práticas corporativas e a integração das metodologias ágeis indicam que as ferramentas e processos discutidos são escolhidos não apenas para a eficiência técnica, mas também para otimizar a colaboração e a gestão dentro das equipes de desenvolvimento.

A presença de palavras frequentes como “documentação”, “deploy” e “kanban” na Figura 60 reiteram a importância que os entrevistados atribuem à gestão eficiente do produto e na adoção de processos ágeis. Além disso, aparecem termos relacionados às ferramentas de

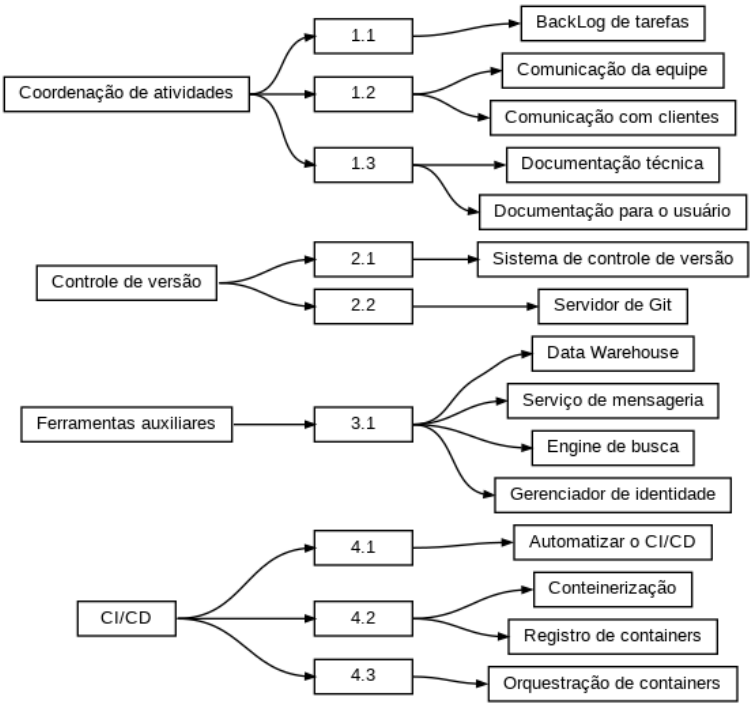
controle de versão, como “branch” e “versão”, embora com menor frequência. Apesar de sua citação ser menos recorrente, essas ferramentas são fundamentais também para a integração contínua e entrega contínua (CI/CD). Sua menor frequência nas respostas sugere que as perguntas relacionadas a esse tema foram abordadas de forma mais pontual, sem uma exploração aprofundada.

4.3 QP2: É possível categorizar as ferramentas de acordo com seu uso?

Para categorizar as ferramentas de acordo com seu uso, foi realizada uma análise detalhada das respostas dos entrevistados por meio da leitura das transcrições das entrevistas. Durante essa análise, observou-se que, ao responderem às perguntas, os entrevistados mencionaram não apenas o nome das ferramentas utilizadas, mas também o papel que as ferramentas desempenhavam no processo de desenvolvimento de *software*.

Este trabalho propôs a organização das ferramentas em 15 categorias distintas de uso, com base nas respostas dos entrevistados. A Figura 61 apresenta um diagrama que mostra as diferentes categorias de ferramentas informadas como respostas para cada pergunta. Isso permite uma visão geral das categorias de uso, separadas de acordo com o número da pergunta correspondente e também o tópico referente a cada pergunta.

Figura 61 – Diagrama de categorias das ferramentas respondidas



Fonte: Próprio autor

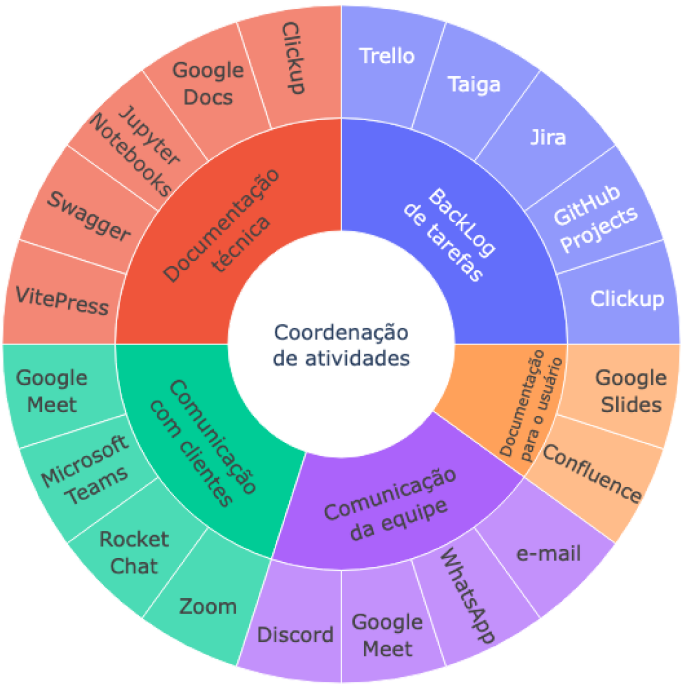
Da Figura 62 até a Figura 65 são apresentadas visualizações do tipo *Sunburst*, que ilustram as categorias de uso das ferramentas mencionadas pelos entrevistados em resposta a cada tópico de perguntas. Essas figuras são organizadas em três anéis concêntricos. No anel interior, está indicado o tópico específico das perguntas em que as ferramentas apresentadas na visualização foram citadas. O anel do meio descreve as categorias de uso propostas por este trabalho. No anel mais exterior, encontram-se as ferramentas citadas, distribuídas conforme suas respectivas categorias de uso.

As cores, nas imagens das Figura 62 até a Figura 65, foram aplicadas para agrupar visualmente as ferramentas dentro de suas respectivas categorias de uso, facilitando a distinção e comparação entre elas. Além disso, a organização hierárquica na visualização do tipo *Sunburst* permite uma distinção entre as categorias de uso, destacando como cada ferramenta se insere nessas categorias.

Na Figura 62 são apresentadas todas as ferramentas que foram citadas nas respostas para as perguntas sobre coordenação de atividades. Conforme ilustrado na imagem, são apresentadas ferramentas utilizadas para comunicação da equipe, organização do *backlog* de tarefas, documentação técnica, comunicação com o cliente e documentação para o usuário. Exemplos incluem Discord e WhatsApp para comunicação da equipe, Microsoft Teams e Google Meet para comunicação com clientes, Trello e GitHub Projects para organização do *backlog* de tarefas, e Confluence e Google Slides para documentação para o usuário.

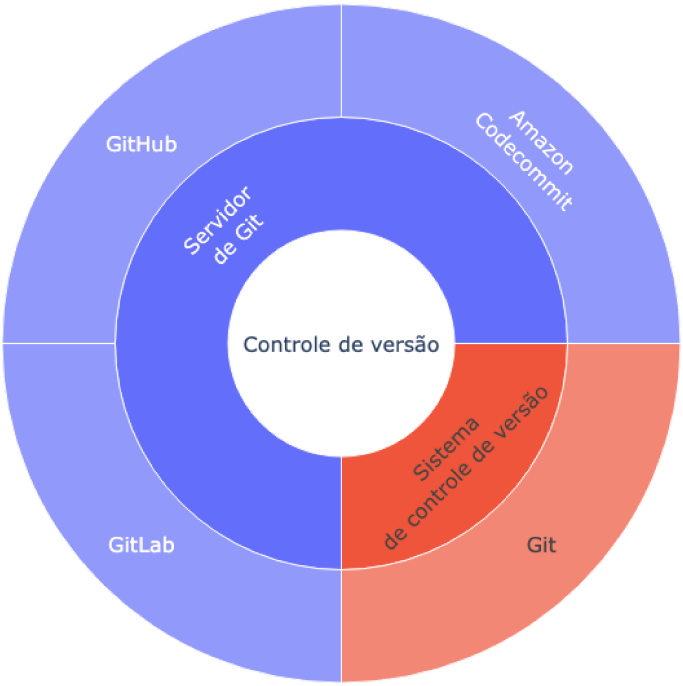
A Figura 63 apresenta a categorização das ferramentas mencionadas como respostas para as perguntas sobre controle de versão. A categorização realizada segue o que foi apresentado nas perguntas desse tema, que distingue entre sistemas de controle de versão e servidores remotos de Git. No entanto, o único sistema de controle de versão mencionado foi o Git. Já os servidores remotos de Git citados foram GitHub, GitLab e Amazon Codecommit.

Figura 62 – Categorização das ferramentas de coordenação de atividades



Fonte: Próprio autor

Figura 63 – Categorização das ferramentas de controle de versão

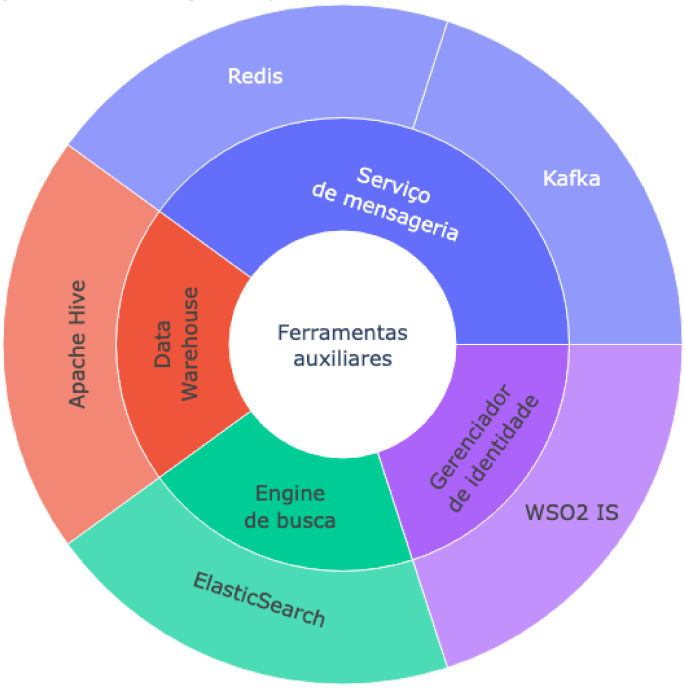


Fonte: Próprio autor

Na Figura 64 são apresentadas as ferramentas citadas como resposta para a pergunta sobre ferramentas auxiliares. Os entrevistados destacaram serviços de mensageria como Kafka e Redis. Para gerenciamento de identidade, foi citado exclusivamente, o WSO2 Identity Server. O

Apache Hive foi mencionado como ferramenta para construção de *data warehouses*, enquanto o ElasticSearch foi indicado como *engine* de busca.

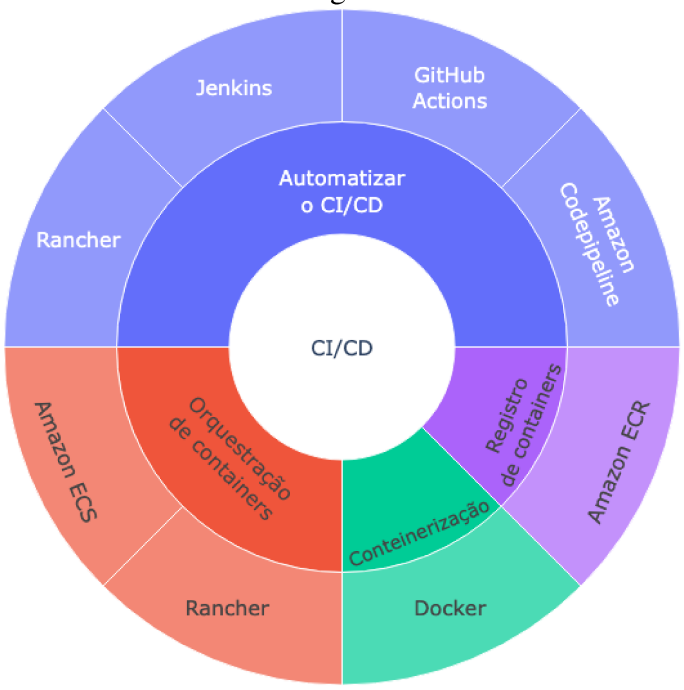
Figura 64 – Categorização das ferramentas auxiliares



Fonte: Próprio autor

São apresentadas na Figura 65, as ferramentas comentadas durante as respostas das perguntas sobre integração contínua e entrega contínua (CI/CD). O Docker foi a única ferramenta citada para conteinerização de aplicação, juntamente com o Amazon Elastic Container Registry (ECR), que foi o único *registry* de *containers* citado. Foram apresentados o Jenkins, GitHub Actions, Amazon Codepipeline e Rancher como alternativas para criar fluxos de integração contínua e entrega contínua (CI/CD). E por fim, o Amazon Elastic Cluster Service (ECS) e o Rancher utilizados para orquestração de *containers*.

Figura 65 – Categorização das ferramentas de integração contínua e entrega contínua



Fonte: Próprio autor

A categorização das ferramentas e a análise das categorias propostas, revelam uma clara distinção entre as funções desempenhadas por cada ferramenta no processo de desenvolvimento Web. As 34 ferramentas únicas mencionadas foram separadas em 15 categorias distintas, evidenciando a diversidade de soluções utilizadas e possivelmente refletindo diferentes domínios e necessidades específicas dos projetos. Essa variedade de ferramentas pode estar relacionada às demandas particulares de cada equipe e projeto, demonstrando a adaptação das ferramentas às necessidades específicas e às características dos diferentes contextos de desenvolvimento.

A categorização revela que, enquanto tópicos como coordenação de atividades são amplamente representados, outros, como controle de versão, são dominadas por um número pequeno de soluções. Ferramentas auxiliares e de integração contínua e entrega contínua (CI/CD), por sua vez, destacam a especialização e a necessidade de soluções direcionadas para funções específicas.

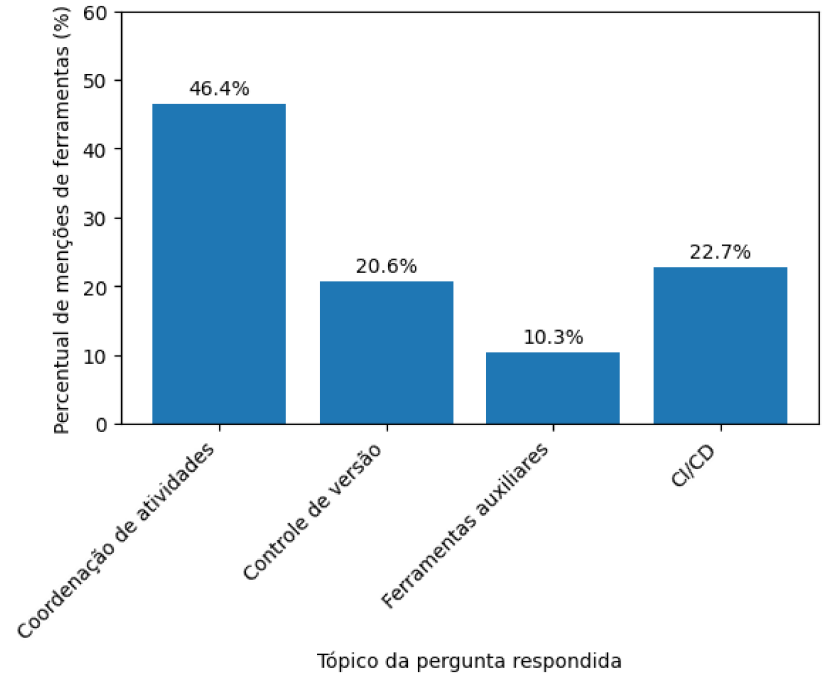
4.4 QP3: Quais a ferramentas mais citadas para apoiar as etapas do desenvolvimento Web?

Conforme apresentado anteriormente, foram definidos quatro tópicos principais para as perguntas no protocolo de entrevistas. Além disso, na Seção 4.3, foram apresentadas as várias

ferramentas que foram mencionadas pelos entrevistados. Nesse sentido, o foco desta questão de pesquisa é na popularidade das ferramentas, avaliando o percentual de menções de cada uma das ferramentas citadas.

A Figura 66 apresenta um gráfico que relaciona o percentual de menções de ferramentas por tópico de perguntas respondidas. Esse gráfico permite identificar quais tópicos tiveram mais ferramentas mencionadas. O gráfico revela que a coordenação de atividades é o tópico mais frequentemente citado, com 46,4% das menções de ferramentas, sendo seguido por ferramentas de integração contínua e entrega contínua (CI/CD) com 22,7%. Já as ferramentas de controle de versão foram citadas em 20,6% do total de respostas, enquanto as ferramentas auxiliares representaram 10,3%.

Figura 66 – Percentual das menções por tópico das perguntas respondidas



Fonte: Próprio autor

Cada tópico aborda aspectos distintos, mas essenciais, do processo de desenvolvimento para a Web, e a respostas das perguntas fornecem informações específicas sobre as ferramentas citadas pelos entrevistados. Nesse sentido, a seguir, é apresentada uma discussão sobre as perguntas relacionadas com coordenação de atividades (Subseção 4.4.1), controle de versão (Subseção 4.4.2), ferramentas auxiliares (Subseção 4.4.3) e integração contínua e entrega contínua (Subseção 4.4.4). Por fim, é feita uma listagem das principais ferramentas avaliadas (Subseção 4.4.5).

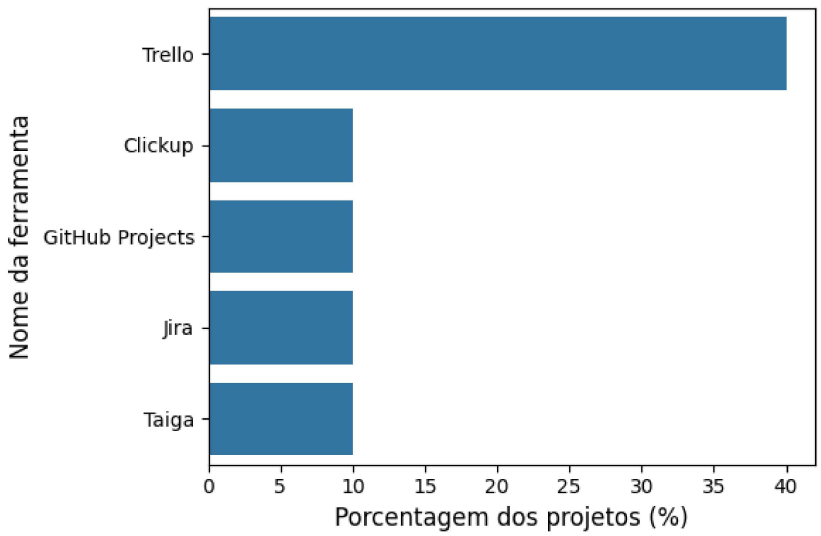
4.4.1 Coordenação de atividades

A primeira pergunta de entrevista (pergunta 1.1) busca entender quais as ferramentas os desenvolvedores entrevistados utilizam para organizar o *backlog* de tarefas em seus projetos. Nesse sentido, a Figura 67 apresenta um gráfico de barras com a frequência de uso dessas ferramentas nos projetos dos entrevistados. Vale ressaltar que os dados dos entrevistados "E03" e "E09" não foram incluídos nessa análise. O projeto do entrevistado "E03" utiliza um sistema interno para organizar suas tarefas. Já o entrevistado "E09" relatou que sua equipe não utiliza uma ferramenta com essa finalidade específica.

Ainda se tratando do gráfico da Figura 67, observa-se que o Trello foi mencionado em 40% dos projetos avaliados, destacando-se como a ferramenta mais utilizada para organização de tarefas entre os entrevistados. Essa preferência pelo Trello pode estar relacionada à sua interface intuitiva e flexível, que permite uma adaptação fácil às necessidades variadas das equipes de desenvolvimento.

Por outro lado, outras ferramentas, como Clickup, GitHub Projects, Jira e Taiga, foram citadas em 10% dos projetos, conforme apresentado pela Figura 67. Isso pode indicar que algumas equipes optam por soluções que oferecem funcionalidades específicas, integração com outras plataformas, ou que se alinham melhor com os processos internos já estabelecidos. Essa diversidade de ferramentas reflete a busca por alternativas que atendam a particularidades e preferências específicas de cada equipe.

Figura 67 – Porcentagem dos projetos por ferramentas citadas para organização de tarefas

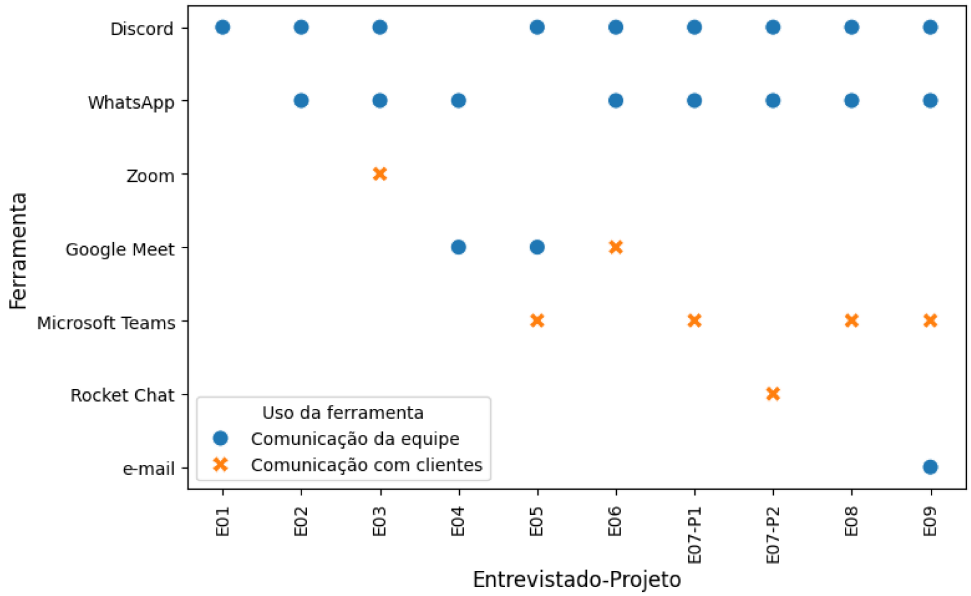


Fonte: Próprio autor

A pergunta 1.2 foi formulada com o objetivo de identificar as plataformas que as equipes dos entrevistados utilizam para se comunicar durante o processo de desenvolvimento. O gráfico apresentado pela Figura 68 apresenta as ferramentas citadas como resposta para essa pergunta. Além disso, na legenda do gráfico foi feita uma categorização de acordo com o uso que cada entrevistado mencionou fazer, sendo separadas entre ferramentas para comunicação da equipe e comunicação com clientes.

Conforme os dados apresentados na Figura 68, para a comunicação interna, o Discord foi a ferramenta mais mencionada, aparecendo em 9 dos 10 projetos. Já o WhatsApp é utilizado por equipes de 8 projetos. O Google Meet também foi mencionado duas vezes para comunicação interna, enquanto o e-mail foi citado uma vez. Já no quesito de comunicação com clientes, o Microsoft Teams foi a ferramenta mais utilizada, sendo citado em 4 dos 10 projetos. Outras ferramentas, como Google Meet, Zoom e Rocket Chat, foram mencionadas de forma isolada para comunicação com clientes, cada uma sendo citada uma única vez.

Figura 68 – Uso das ferramentas de comunicação por projeto

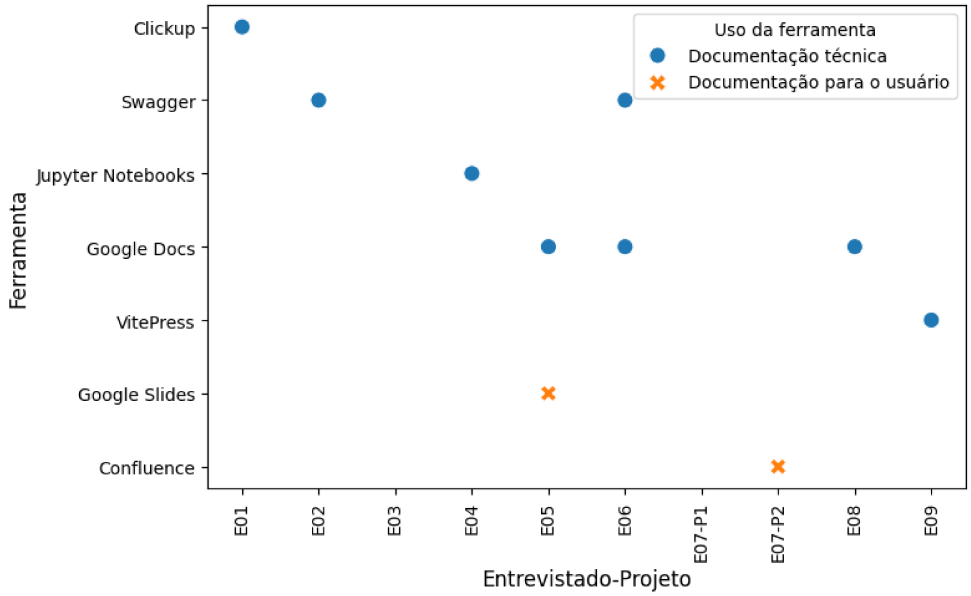


Fonte: Próprio autor

A pergunta 1.3 foi formulada para identificar as ferramentas que os entrevistados utilizam para fazer a documentação de seus projetos. O gráfico da Figura 69 mostra as ferramentas citadas como respostas dessa pergunta. Além disso, é importante citar que as ferramentas foram organizadas de acordo com o uso mencionado pelos entrevistados, o que inclui documentação técnica e para o usuário final. Ao verificar a Figura 69, percebe-se que o entrevistado “E03“ não citou ferramentas de documentação, assim como o projeto de número um do entrevistado “E07”.

Observa-se por meio do gráfico apresentado na Figura 69, que a ferramenta mais citada para documentação técnica foi o Google Docs, mencionada em 3 de 10 projetos. Logo em seguida, o Swagger, que foi mencionado em 2 projetos. Para a documentação técnica, também foram citados de forma pontual o Jupyter Notebooks, Clickup e VitePress. Já para a documentação voltada para o usuário, foram citados o Google Slides e o Confluence, ambos uma única vez.

Figura 69 – Uso das ferramentas de documentação por projeto



Fonte: Próprio autor

4.4.2 Controle de versão

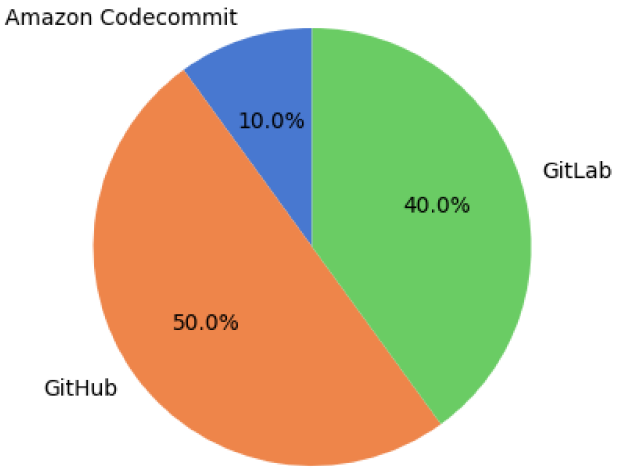
A pergunta 2.1 foi elaborada com o objetivo de avaliar o uso dos sistemas de controle de versão. Entretanto, o Git foi citado como solução adotada em todos os 10 projetos analisados. Esse dado não apenas confirma a importância do Git como ferramenta de controle de versão, mas também destaca o papel central no processo de colaboração e gerenciamento de configurações entre os desenvolvedores da amostra selecionada. A unânime adoção reforça sua posição como uma das principais ferramentas no cenário de desenvolvimento de *software* para a Web.

Como já era esperado, dado o uso unânime do Git como sistema de controle de versão, a pergunta 2.2 investigou os diferentes servidores remotos de Git utilizados nos projetos dos entrevistados. A Figura 70 apresenta um gráfico com os servidores remotos utilizados nos projetos dos entrevistados. Os resultados mostraram que 50% dos projetos avaliados utilizam o GitHub, enquanto 40% mencionaram o GitLab. Por fim, o Amazon Codecommit foi citado em

10% dos casos.

Esses dados refletem a ampla adoção do GitHub, reafirmando sua popularidade na comunidade de desenvolvedores. No entanto, o GitLab também se destaca como uma escolha significativa, especialmente por sua natureza de código aberto e a capacidade de ser hospedado *on-premises*, oferecendo maior controle sobre o ambiente de desenvolvimento. O uso do Amazon Codecommit, embora menos comum, sugere sua aplicação em cenários específicos que exigem integração com outros serviços da Amazon. Dessa forma, a análise mostra uma predominância do GitHub, mas também revela a robustez e relevância do GitLab no mercado.

Figura 70 – Distribuição dos servidores remotos de Git



Fonte: Próprio autor

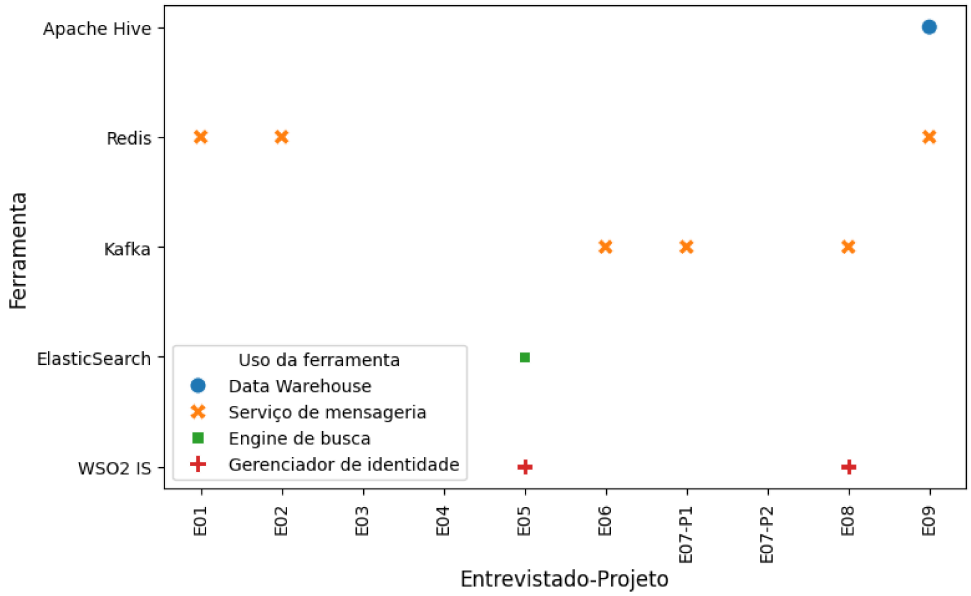
4.4.3 Ferramentas auxiliares

A pergunta 3.1 foi elaborada para investigar quais *softwares* comerciais ou de código aberto interagem com as aplicações desenvolvidas pelas equipes dos entrevistados. A questão explora o uso de serviços complementares, como mensageria, *engines* de busca, serviços de federação de identidade e outras ferramentas que desempenham papéis importantes no ecossistema das aplicações. A Figura 71 apresenta um gráfico que relaciona as ferramentas citadas com os projetos nos quais são utilizadas, apresentado uma visão clara das integrações adotadas nos diferentes contextos de desenvolvimento analisados.

Além das ferramentas mencionadas, o Kafka e o Redis foram citados como serviços de mensageria, cada um sendo empregado em 3 dos 10 projetos analisados. O WSO2 Identity Server, presente em dois projetos, destacou-se como uma solução para a federação de identidade,

sendo usado para o gerenciamento de identidades em organizações. O Elastic Search e o Apache Hive foram mencionados uma vez cada, respectivamente como *engine* de busca e para a construção de *data warehouses*, logo ambos podendo ser usados para a busca e armazenamento de dados.

Figura 71 – Uso das ferramentas auxiliares por projeto



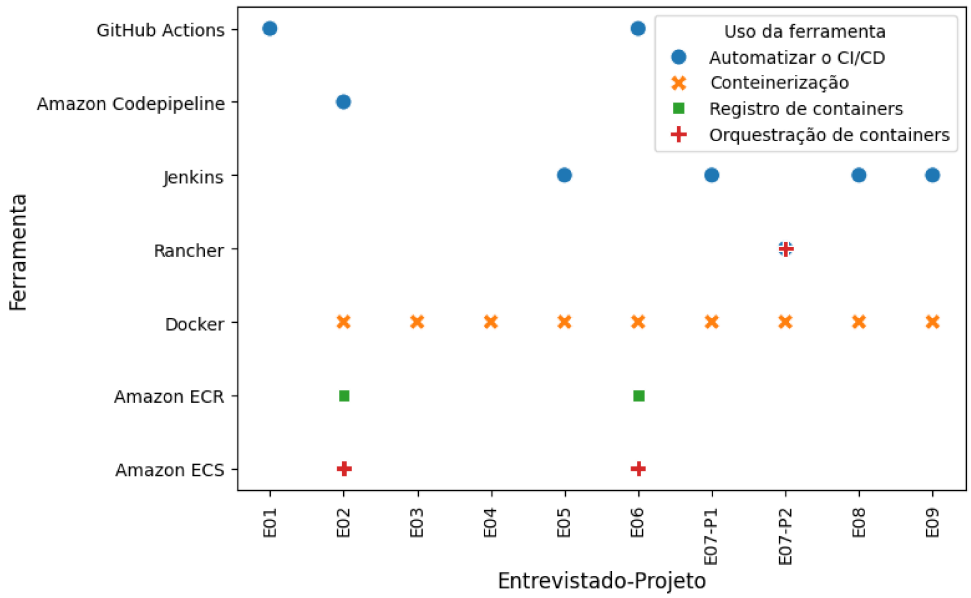
Fonte: Próprio autor

4.4.4 Integração contínua e entrega contínua (CI/CD)

A análise dos dados da Figura 72 revela uma quase unanimidade no uso do Docker para conteinerização, sendo utilizado em todos os projetos, exceto no projeto do entrevistado "E01". Isso reflete a ampla adoção do Docker como padrão na criação de ambientes isolados e consistentes para desenvolvimento e implantação de *software*.

Além disso, foram mencionadas diversas ferramentas destinadas à automação de processos de CI/CD, como Jenkins, GitHub Actions e Amazon Codepipeline, demonstrando a importância da integração contínua e da entrega contínua no fluxo de trabalho das equipes. Amazon ECS e Amazon ECR foram sempre citadas em conjunto, sendo utilizadas, respectivamente, para a orquestração e registro de *containers*. A ferramenta Rancher destaca-se no que quesito versatilidade, visto que foi citada tanto para orquestração de *containers* quanto para automação de CI/CD.

Figura 72 – Uso das ferramentas relacionadas com CI/CD por projeto



Fonte: Próprio autor

4.4.5 Principais ferramentas avaliadas

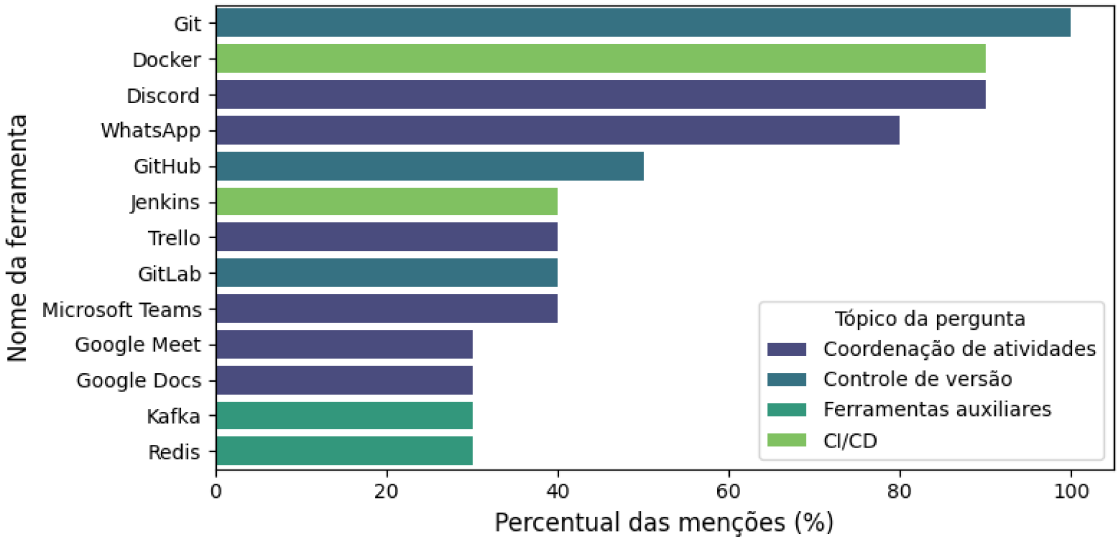
Para identificar as ferramentas mais relevantes nos projetos analisados, foram consideradas todas as ferramentas presentes, com o objetivo de destacar as mais utilizadas de acordo com o percentual de uso nos projetos. Inicialmente, pretendia-se limitar a lista às 10 principais ferramentas. No entanto, devido a um empate nas últimas posições, foi necessário expandir a lista para incluir 13 ferramentas no total. A Figura 73 apresenta essas ferramentas, destacando o nome de cada ferramenta e o percentual de projetos que a utilizam. Além disso, é apresentado uma legenda que identifica em qual tópico de perguntas a ferramenta foi mencionada.

A Figura 73 revela que o Git, Docker e Discord são as ferramentas mais utilizadas, sendo empregadas em 90% ou mais dos projetos avaliados. Além disso, GitHub e GitLab somados são utilizados em 90% dos projetos. Esses resultados evidenciam a relevância das ferramentas em suas categorias de uso, que são o controle de versão, conteinerização e comunicação de equipes. A elevada frequência em seu uso reflete sua importância para os projetos dos entrevistados.

Ferramentas como WhatsApp, Microsoft Teams, Trello e Jenkins também se mostraram essenciais, sendo usadas em 40% a 80% dos projetos. Google Meet e Google Docs, embora menos frequentes, também desempenharam um papel importante em 30% dos projetos, destacando-se na coordenação de atividades e na documentação técnica, respectivamente. Outros recursos, como Redis e Kafka para mensageria, completam o grupo, evidenciando a diversidade

e especialização das ferramentas empregadas. Em conjunto, essas 13 ferramentas formam a base tecnológica que sustenta a maioria dos projetos estudados, demonstrando seu importante papel no processo de desenvolvimento de *software* para a Web.

Figura 73 – As 13 principais ferramentas citadas nos projetos dos entrevistados



Fonte: Próprio autor

4.5 Comparativo com uma pesquisa de mercado

Com o intuito de avaliar se os resultados deste estudo estão condizentes com pesquisas de mercado, comparamos os resultado obtidos com uma pesquisa do Stack Overflow (Stack Overflow, 2024). A análise do percentual de uso foi feita por desenvolvedor, alinhando-se ao método adotado pela pesquisa do Stack Overflow. Isso difere do que foi apresentado na Seção 4.4, que considerou o percentual de uso por projetos.

É importante notar que a pesquisa do Stack Overflow apresenta uma perspectiva ligeiramente diferente quando aborda as ferramentas utilizadas pelos desenvolvedores. Ao invés de apresentar perguntas livres, como nas entrevistas realizadas neste trabalho, a pesquisa apresenta uma lista com várias ferramentas e pede ao respondente para dizer se já utilizou aquela ferramenta no último ano. Em contraste, o presente estudo se concentra nas ferramentas que os desenvolvedores estão utilizando atualmente nos projetos em que trabalham.

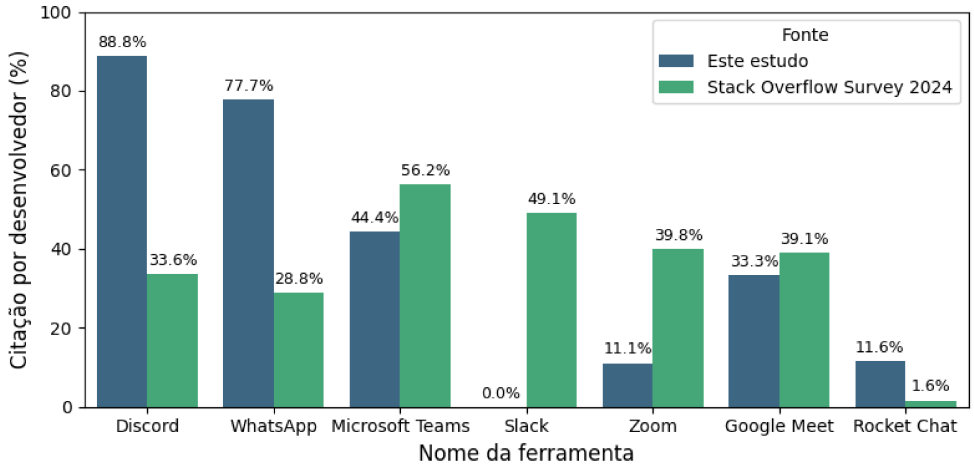
Para garantir uma comparação mais precisa, foram selecionados os dados referentes aos desenvolvedores profissionais na pesquisa do Stack Overflow. Essa abordagem permite identificar tanto convergências quanto divergências entre as práticas observadas neste estudo e as

tendências globais, proporcionando uma visão mais detalhada sobre a adoção de ferramentas no mercado atual.

A pesquisa do Stack Overflow categoriza as ferramentas de coordenação de atividades em comunicação síncrona e assíncrona, enquanto o presente estudo adota uma divisão com base no uso que os desenvolvedores deram às ferramentas: *backlog* de tarefas, comunicação da equipe, comunicação com clientes, documentação técnica e documentação para o usuário. Além disso, a pesquisa do Stack Overflow não avaliou os sistemas de controle de versão nem os servidores de Git utilizados pelos desenvolvedores.

A Figura 74 apresenta um comparativo das ferramentas de comunicação síncrona, em que a pesquisa do Stack Overflow revela que as mais utilizadas globalmente são Microsoft Teams (56,2%), Slack (49,1%), Zoom (39,8%), Google Meet (39,1%), Discord (33,6%) e WhatsApp (28,8%). No presente estudo, todas essas ferramentas, exceto o Slack, foram mencionadas. A distribuição observada foi: Discord (88,8%), WhatsApp (77,7%), Microsoft Teams (44,4%), Google Meet (33,3%) e Zoom (11,1%). O Zoom, que é a terceira ferramenta mais utilizada globalmente, foi citado apenas uma vez neste estudo, enquanto o Rocket Chat, que segundo o Stack Overflow, ocupa a 16ª posição entre as ferramentas de comunicação síncrona, foi mencionado apenas uma vez neste estudo, representando 11,1% dos entrevistados.

Figura 74 – Comparativo do uso das ferramentas de comunicação síncrona

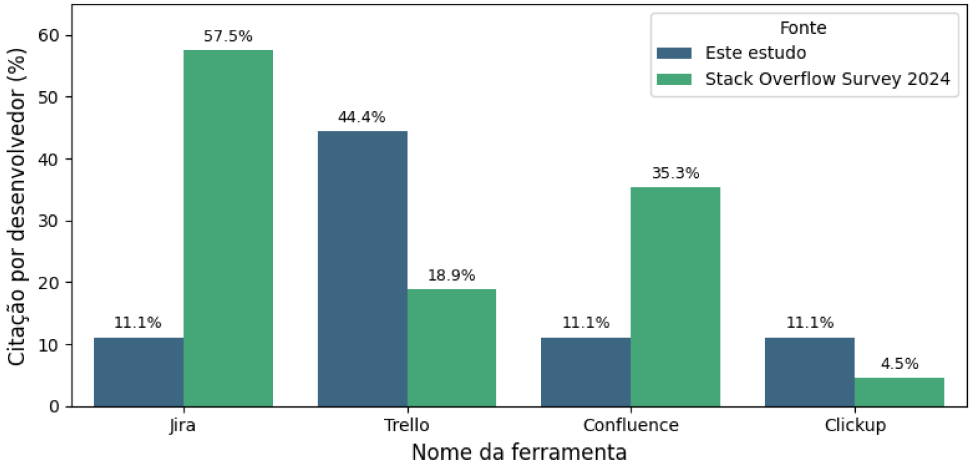


Fonte: Próprio autor

Já a Figura 75 apresenta o comparativo das ferramentas de comunicação assíncrona. As ferramentas mais utilizadas por desenvolvedores profissionais, segundo a pesquisa do Stack Overflow, são Jira (57,5%), Confluence (35,3%) e Trello (18,9%). Dentre essas ferramentas, o Trello foi identificado como uma das principais neste estudo, com uma utilização de 44,4%. Jira

e Confluence foram mencionados somente por um entrevistado, sendo 11,1% do total. O Clickup, que foi citado por 4,5% dos participantes na pesquisa do Stack Overflow, foi mencionado somente por um dos entrevistados (11,1%) deste estudo.

Figura 75 – Comparativo do uso das ferramentas de comunicação assíncrona



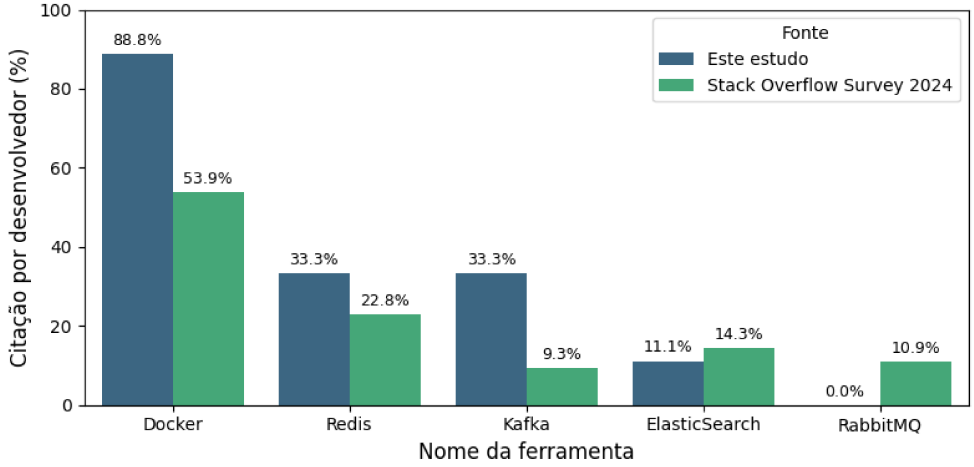
Fonte: Próprio autor

Conforme a Figura 76, a pesquisa de mercado revela que Redis foi utilizado por 22,8% dos desenvolvedores, ElasticSearch por 14,3% e Kafka por 9,4%. Em contraste, neste trabalho, a análise revelou que tanto Kafka quanto Redis foram utilizados por 33,3% dos desenvolvedores entrevistados, mostrando uma adoção mais ampla dessas tecnologias em comparação com o estudo global. ElasticSearch, por outro lado, foi mencionado pontualmente por apenas um dos entrevistados neste estudo, sendo isso 11,1% dos entrevistados.

Ainda no contexto da Figura 76, o RabbitMQ, que é um concorrente significativo de Kafka e Redis no quesito mensageria e foi utilizado por 10,9% dos desenvolvedores na pesquisa do Stack Overflow, não foi mencionado por nenhum dos entrevistados neste estudo.

Na pesquisa de mercado, a abordagem de ferramentas de CI/CD foi limitada. Nesse sentido, das ferramentas relacionadas com CI/CD avaliadas no presente estudo, somente o Docker foi mencionado na pesquisa. O Docker foi utilizado por 88,8% dos entrevistados, enquanto a pesquisa do Stack Overflow revelou que 53,9% dos desenvolvedores profissionais utilizam Docker, conforme apresentado na Figura 76.

Figura 76 – Comparativo do uso das demais ferramentas



Fonte: Próprio autor

As diferenças observadas entre as ferramentas citadas na pesquisa do Stack Overflow e as mencionadas no presente estudo podem indicar variações regionais e de custo. A pesquisa do Stack Overflow, ao avaliar tendências globais, reflete uma perspectiva mais ampla que pode não capturar completamente as especificidades locais. Portanto, as discrepâncias nas preferências de ferramentas discutidas podem sugerir que as soluções adotadas no Brasil atendem a necessidades específicas e restrições econômicas que não são tão evidentes em uma análise global.

4.6 Lições aprendidas

Ao longo desse estudo, foram exploradas as principais categorias de ferramentas que apoiam o desenvolvimento Web. Com base na análise das falas dos entrevistados, foi possível listar ferramentas utilizadas em projetos reais de *software* para a Web e identificar as tendências no uso dessas tecnologias entre os entrevistados. Com isso, foram identificas as seguintes lições aprendidas.

I. **Diversidade de ferramentas para coordenação de atividades:** A escolha de ferramentas para comunicação e coordenação revelou uma diversidade maior, com Discord e WhatsApp sendo amplamente utilizados para comunicação interna e Microsoft Teams predominando na comunicação com clientes. A diversificação de ferramentas mostra a adaptação das equipes às necessidades específicas de cada projeto. Enquanto Discord e WhatsApp são ferramentas rápidas e acessíveis para comunicação diária, Microsoft Teams é o preferido para reuniões mais formais e interações com clientes, destacando a importância de escolher ferramentas que melhor se alinhem às necessidades da equipe e do projeto.

II. **Consolidação do mercado e a importância das ferramentas de controle de versão:** Durante o estudo, foi observado que a quantidade de soluções relacionadas ao controle de versão foi limitada, com a maioria dos desenvolvedores se concentrando em um pequeno conjunto de ferramentas. O Git foi mencionado em todos os projetos dos entrevistados, sendo o único sistema de controle de versão utilizado pela amostra de entrevistados. Além disso, os servidores remotos de Git também foram utilizados em todos os projetos. GitHub e GitLab foram usados em 90% dos projetos. Notavelmente, não foram apresentados concorrentes ao Git durante as entrevistas, sugerindo uma consolidação no mercado de controle de versão. Os resultados podem indicar que o uso de Git e de um servidor de Git, como GitHub ou GitLab, são essenciais para a prática cotidiana do desenvolvimento de *software*. Além disso, os projetos mencionados demonstraram uma preferência notável por GitHub e GitLab, o que pode reforçar sua importância na colaboração e no versionamento em projetos de *software*.

III. **Especificidade das ferramentas auxiliares:** O estudo revelou que o uso de ferramentas auxiliares é bastante específico, concentrando essas ferramentas em categorias como serviços de mensageria, *engines* de busca, *data warehouses* e gerenciador de identidade. Dentro dessas quatro categorias, apenas cinco ferramentas únicas foram mencionadas. Embora 70% dos projetos tenham empregado pelo menos uma ferramenta auxiliar, a variedade de ferramentas usadas foi limitada. Dos projetos avaliados, 60% utilizaram serviços de mensageria, com Redis e Kafka sendo empregados em 30% dos projetos cada, destacando a relevância e possível consolidação dessas ferramentas no mercado. Ferramentas das outras categorias foram citadas de forma pontual e em menor número. Isso sugere que as ferramentas auxiliares são escolhidas para atender a demandas específicas de organizações e projetos, refletindo uma abordagem especializada e focada na seleção de soluções adequadas para necessidades particulares.

IV. **Adoção de práticas de CI/CD e automação de processos:** O estudo revelou que o Docker é amplamente utilizado em projetos, sendo empregado por 90% dos projetos avaliados. Além disso, 80% dos projetos utilizam serviços para automatizar processos de integração contínua e entrega contínua (CI/CD), evidenciando a importância desses serviços na automação e eficiência das operações de desenvolvimento. Entre os serviços citados para automação de CI/CD, o Jenkins foi o mais utilizado, aparecendo em 40% dos projetos, seguido pelo GitHub Actions com 20%. Esses resultados sugerem que a

automação de processos e a conteinerização estão amplamente integrados nas práticas de desenvolvimento, com um grupo de ferramentas desempenhando um papel central na integração e entrega contínua de *software*.

V. **Escolha das ferramentas e variabilidade conforme o contexto:** A escolha das ferramentas varia significativamente de acordo com o contexto de uso, incluindo fatores regionais, organizacionais e de integração com outras plataformas. Ferramentas que podem ser utilizadas de forma gratuita para negócios, como o Discord e o Trello, são frequentemente preferidas para comunicação e gerenciamento de tarefas devido à sua flexibilidade. Além disso, a preferência por ferramentas que podem ser hospedadas localmente, como o GitLab, Jenkins, Redis e Kafka, podem refletir uma estratégia de minimizar custos e aumentar o controle sobre os ambientes de desenvolvimento. Essas ferramentas são vistas como alternativas viáveis a serviços pagos, oferecendo soluções robustas e customizáveis que atendem às necessidades específicas de projetos e organizações. Essa variação contextual sugere que a seleção de ferramentas é muitas vezes orientada pelas especificidades do ambiente de trabalho e pelas necessidades particulares de cada projeto, destacando a importância de uma abordagem flexível e adaptada na escolha das tecnologias utilizadas no desenvolvimento de *software*.

5 TRABALHOS RELACIONADOS

Dos estudos selecionados, dois se concentram na listagem e comparação de *frameworks* para o desenvolvimento Web (XING *et al.*, 2019; FERNÁNDEZ-VILLAMOR *et al.*, 2008), enquanto o terceiro coleta percepções de desenvolvedores por meio de entrevistas e um *survey* (CONSTANTINO *et al.*, 2023). Os estudos que analisam desempenho e funcionalidades de ferramentas específicas (XING *et al.*, 2019; FERNÁNDEZ-VILLAMOR *et al.*, 2008) não consideram a perspectiva prática dos desenvolvedores. Por outro lado, (CONSTANTINO *et al.*, 2023) adota uma abordagem focada nas percepções de desenvolvedores sobre colaboração em projetos de código aberto, mas não aborda ferramentas diretamente.

O primeiro estudo, (CONSTANTINO *et al.*, 2023), adota uma abordagem qualitativa, coletando percepções de desenvolvedores através de entrevistas e um *survey*. O público avaliado foi o dos desenvolvedores que contribuem para projetos de código aberto. As entrevistas foram realizadas em português com desenvolvedores brasileiros. Já o *survey* foi respondido por desenvolvedores de vários países. Ao final, foram apresentadas lições aprendidas com base nas entrevistas e nas respostas do *survey*.

O segundo estudo, (XING *et al.*, 2019), foca na análise de desempenho e funcionalidades entre os *frameworks front-end* React, Angular 2 e Vue, do JavaScript. A análise incluiu tanto a descrição técnica desses *frameworks*, feita com base na revisão da literatura, quanto a avaliação de seu desempenho. Com isso, concluiu-se que Angular 2 é mais adequado para grandes projetos comerciais e React e Vue são adequados para criar interfaces que precisam ser atualizadas em tempo real.

O terceiro estudo, (FERNÁNDEZ-VILLAMOR *et al.*, 2008), propõe um modelo comparativo para avaliar os *frameworks* Rails, Grails, Trails e Roma da linguagem Ruby. Esse modelo foi criado por meio de uma revisão da literatura sobre esses *frameworks* e feito com base nas funcionalidades oferecidas. Sabendo disso, o modelo comparativo ajuda a identificar os pontos fortes e fracos de cada *framework*, sendo sugerido pelo autor do estudo como uma ferramenta útil para a escolha de tecnologias para iniciar novos projetos.

O estudo de (CONSTANTINO *et al.*, 2023) conduziu entrevistas com 12 desenvolvedores brasileiros, focando em colaboradores de projetos de código aberto, e também obteve 121 respostas em um *survey*, envolvendo desenvolvedores de diferentes países. Já o presente estudo, por sua vez, entrevistou um número similar de participantes, com 9 desenvolvedores que atuam no mercado brasileiro de desenvolvimento de *software*.

(XING *et al.*, 2019) apresentou a descrição e também uma análise de performance entre *frameworks front-end*. Ao final foram feitas conclusões com vantagens e desvantagens das três tecnologias avaliadas. Em contraste, o presente trabalho categorizou as ferramentas conforme seu uso, oferecendo uma visão abrangente sobre como as ferramentas que apoiam o desenvolvimento Web podem ser utilizadas, com base nas respostas dos desenvolvedores entrevistados.

(FERNÁNDEZ-VILLAMOR *et al.*, 2008) concluiu que o modelo comparativo desenvolvido é importante para ajudar na escolha de um *framework* ao iniciar um novo projeto. Em outra perspectiva, o presente estudo fornece dados sobre o uso das ferramentas que apoiam o desenvolvimento Web em projetos reais. Isso permite direcionar desenvolvedores em relação à opções populares e adequadas para suas necessidades práticas.

A Tabela 5 sumariza os estudos de propósitos similares. As colunas da tabela apresentam características dos estudos, como a metodologia utilizada, objeto de estudo e alguns resultados, enquanto as linhas listam os estudos avaliados, incluindo este trabalho e três outros publicados em conferências.

Tabela 5 – Comparativo de trabalhos relacionados

Trabalho	Metodologia	Objeto de estudo	Resultados
Presente estudo (2024)	Entrevistas com desenvolvedores	Ferramentas que apoiam o desenvolvimento Web	Extração de lições aprendidas com base em 34 ferramentas citadas por 9 entrevistados
(CONSTANTINO *et al.*, 2023)	Entrevistas e *survey*	Colaboração em projeto de código abertos	Lições sobre o objeto de estudo com base em 121 respostas do *survey* e 12 entrevistados
(XING *et al.*, 2019)	Revisão da literatura e análise de desempenho	*Frameworks front-end*	Dados de performance de 3 *frameworks front-end* do JavaScript
(FERNÁNDEZ-VILLAMOR *et al.*, 2008)	Revisão da literatura e comparação de funcionalidades	*Frameworks* Web	Um modelo comparativo entre 4 *frameworks* de Ruby

Fonte: Próprio autor

6 CONCLUSÕES E TRABALHOS FUTUROS

O presente estudo teve como principal objetivo investigar ferramentas que apoiam o desenvolvimento Web e seu uso por profissionais de Tecnologia da Informação. A metodologia adotada baseou-se em entrevistas realizadas entre abril e agosto de 2024 com 9 desenvolvedores que atuam no mercado de desenvolvimento Web, focando em coletar percepções detalhadas sobre o uso dessas ferramentas em diferentes contextos de desenvolvimento.

Os resultados obtidos destacam a predominância de algumas ferramentas no cenário do desenvolvimento Web, conforme indicado pela análise dos projetos avaliados. O Git, Docker e Discord emergem como as ferramentas mais amplamente adotadas, com o Git sendo citado em 100% dos projetos. Essa alta taxa de utilização ressalta a importância do Git para o controle de versão e a centralização do código-fonte. Docker e Discord, também empregados em 90% dos projetos, evidenciam a relevância também da conteinerização e de ferramentas para a comunicação para o sucesso dos projetos.

Apesar da divisão entre GitHub e GitLab, todos os projetos utilizam um servidor de Git, o que enfatiza a importância dessas ferramentas para o controle de versão e a centralização do código-fonte. GitHub e GitLab, quando considerados em conjunto, aparecem em 90% dos projetos avaliados, reforçando a importância das ferramentas de gerenciamento de código e colaboração no desenvolvimento Web. Outras ferramentas, como WhatsApp, Microsoft Teams, Trello e Jenkins, desempenham papéis importantes, sendo utilizadas em 40% a 80% dos projetos. Google Meet e Google Docs, embora menos frequentes, são mencionadas em 30% dos projetos, para comunicação e documentação técnica. Ferramentas como Redis e Kafka também aparecem em 30% dos projetos cada, demonstrando a especialização necessária em diferentes contextos de desenvolvimento.

As implicações dos resultados do presente estudo são significativas para compreender o cenário atual do desenvolvimento Web. A predominância de Git e servidores de Git, como GitHub e GitLab, indica uma consolidação do mercado de controle de versão, evidenciando sua importância crucial para o versionamento e colaboração em projetos de software. A diversidade nas ferramentas de coordenação, com o uso extensivo de Discord e WhatsApp para comunicação interna e Microsoft Teams para interações com clientes, revela uma adaptação às necessidades específicas de cada projeto, destacando a importância de selecionar ferramentas adequadas para diferentes contextos.

Além disso, a especificidade no uso de ferramentas auxiliares, como Redis e Kafka,

aponta para uma escolha focada em atender a demandas particulares dos projetos. A alta adoção de práticas de CI/CD e a ampla utilização de Docker demonstram a importância dada à integração contínua e entrega contínua e da conteinerização no desenvolvimento de *software*. A variabilidade na escolha de ferramentas, influenciada por fatores regionais e organizacionais, acentua a necessidade de uma abordagem flexível e adaptada para a seleção de tecnologias. Essas observações destacam a importância de entender o contexto específico de cada projeto para otimizar o uso das ferramentas e práticas de desenvolvimento.

O presente estudo oferece contribuições para profissionais e estudantes envolvidos no desenvolvimento Web. Ao apresentar ferramentas utilizadas em projetos reais de *software* para a Web, o estudo proporciona um panorama claro das tecnologias mais relevantes e suas aplicações práticas. Com as informações documentadas, profissionais e estudantes têm a oportunidade de tomar decisões informadas ao escolher as ferramentas adequadas para seus projetos, alinhando suas escolhas às necessidades e tendências atuais do mercado.

Com trabalhos futuros é proposta uma expansão do estudo com mais entrevistas para contemplar desenvolvedores que atuam em diferentes países, a fim de obter uma perspectiva mais global sobre as ferramentas Web utilizadas por esses desenvolvedores. Além disso, outra abordagem seria realizar uma mineração de dados em fóruns de discussão, grupos de desenvolvedores, e outras plataformas de comunicação abertas e online, onde as ferramentas que apoiam o desenvolvimento Web são frequentemente discutidas. Isso vai permitir identificar padrões de uso de ferramentas com um conjunto maior de dados para serem explorados. Finalmente, pode também ser proposto um catálogo de ferramentas recomendadas, com intuito de atender diferentes necessidades e propostas de sistemas Web.

REFERÊNCIAS

ATLASSIAN. **BitButcket Tutorial - git pull**. 2023. Disponível em: <https://www.atlassian.com/git/tutorials/syncing/git-pull>. Acesso em: 09 jun. 2023.

AUTH0. **Auth0 Documentation - Introduction to Identity and Access Management (IAM)**. 2023. Disponível em: <https://auth0.com/docs/get-started/identity-fundamentals/identity-and-access-management>. Acesso em: 28 mai. 2023.

BAHRINI, R.; QAFFAS, A. A. Impact of information and communication technology on economic growth: Evidence from developing countries. **Economies**, College of Business, University of Jeddah, Asfan Road 21595, Saudi Arabia, v. 7, n. 1, 2019. ISSN 2227-7099. Disponível em: <https://www.mdpi.com/2227-7099/7/1/21>.

BEER, B. **Introducing GitHub: A Non-Technical Guide**. [S.l.]: O'Reilly Media, 2018.

CHACON, S. **Why Git is Better than X**. 2009. Disponível em: <http://web.archive.org/web/20090210020404id_/http://whygitisbetterthanx.com/#the-staging-area>. Acesso em: 04 jun. 2023.

CHACON, S.; STRAUB, B. **Pro git**. [S.l.]: Apress, 2014. Acesso em: 03 jun. 2023.

CONSTANTINO, K.; SOUZA, M.; ZHOU, S.; FIGUEIREDO, E.; KÄSTNER, C. Perceptions of open-source software developers on collaborations: An interview and survey study. **Journal of Software: Evolution and Process**, Wiley Online Library, v. 35, n. 5, p. e2393, 2023.

CONSTANTINO, K.; ZHOU, S.; SOUZA, M.; FIGUEIREDO, E.; KÄSTNER, C. Understanding collaborative software development. In: **Proceedings of the 15th International Conference on Global Software Engineering**. New York, NY, USA: ACM, 2020.

DOCKER. **Guides - Overview**. 2020. Disponível em: <https://docs.docker.com/get-started/docker-overview>. Acesso em: 30 mai. 2023.

FERNÁNDEZ-VILLAMOR, J. I.; DÍAZ-CASILLAS, L.; IGLESIAS, C. Á. A comparison model for agile web frameworks. In: **Proceedings of the 2008 Euro American Conference on Telematics and Information Systems**. [S.l.: s.n.], 2008. p. 1–8.

FRATERNALI, P. Tools and approaches for developing data-intensive web applications: a survey. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 31, n. 3, p. 227–263, 1999.

GITHUB. **GitHub Importer**. 2023. Disponível em: <https://docs.github.com/en/migrations/importing-source-code/using-github-importer/about-github-importer>. Acesso em: 18 jun. 2023.

GITKRAKEN. **Best Practices - Writing a Good Git Commit Message**. 2023. Disponível em: <https://www.gitkraken.com/learn/git/best-practices/git-commit-message>. Acesso em: 05 jun. 2023.

KARUNARATHNA, I.; KARUNARATNE, I. **Biblioteca - O que é o WSO2 Identity Server?** 2017. Disponível em: <https://wso2.com/library/articles/2017/08/o-que-e-o-wso2-identity-server>. Acesso em: 25 mai. 2023.

LAZAR, J. User-centered web development. In: . [S.l.]: Jones & Bartlett Learning, 2001. p. "12–16".

MOZILLA. **The web and web standards**. 2021. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/The_web_and_web_standards>. Acesso em: 27 abr. 2023.

RADFORD, A.; KIM, J. W.; XU, T.; BROCKMAN, G.; MCLEAVEY, C.; SUTSKEVER, I. **Robust Speech Recognition via Large-Scale Weak Supervision**. 2022. Disponível em: <https://arxiv.org/abs/2212.04356>.

RAJENDRAN, L.; VEILUMUTHU, R. A comparative study on internet application development tools. **International Journal of Engineering Science and Technology**, 2010.

ROSSON, M. B.; BALLIN, J.; NASH, H. Everyday programming: Challenges and opportunities for informal web development. In: IEEE. **2004 IEEE Symposium on Visual Languages-Human Centric Computing**. [S.l.], 2004. p. 123–130.

Stack Overflow. **Stack Overflow Developer Survey 2024**. 2024. Disponível em: <https://survey.stackoverflow.co/2024>.

TKACHEV, D. The evolution of software engineering and web development: tools, methods, approaches. 2023.

TZAFILKOU, K.; PROTOGEROS, N.; CHOULIARA, A. Experiential learning in web development courses: Examining students' performance, perception and acceptance. **Education and Information Technologies**, Springer, v. 25, p. 5687–5701, 2020.

VALENTE, M. T. **Engenharia de Software Moderna**. [*S. l.*]: Independente, 2022. ISBN 9786500019506.

WSO2. **API Manager Documentation - Authorization Code Grant**. 2023. Disponível em: <https://apim.docs.wso2.com/en/latest/design/api-security/oauth2/grant-types/authorization-code-grant>. Acesso em: 25 mai. 2023.

WSO2. **Identity Server Documentation - Architecture**. 2023. Disponível em: <https://is.docs.wso2.com/en/latest/get-started/architecture/#service-provider-section>. Acesso em: 25 mai. 2023.

WSO2. **Identity Server Documentation - OpenID Connect Scopes and Claims**. 2023. Disponível em: <https://is.docs.wso2.com/en/latest/references/concepts/authentication/scopes-claims>. Acesso em: 28 mai. 2023.

WSO2. **Identity Server Documentation - Refresh Token Grant**. 2023. Disponível em: <https://is.docs.wso2.com/en/latest/learn/refresh-token-grant>. Acesso em: 28 mai. 2023.

WSO2. **Identity Server Documentation - Single Sign-On**. 2023. Disponível em: <https://is.docs.wso2.com/en/latest/references/concepts/single-sign-on>. Acesso em: 25 mai. 2023.

XING, Y.; HUANG, J.; LAI, Y. Research and analysis of the front-end frameworks and libraries in e-business development. In: **Proceedings of the 2019 11th International Conference on Computer and Automation Engineering**. [S.l.: s.n.], 2019. p. 68–72.