UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

DEPARTAMENTO DA COMPUTAÇÃO

GABRIEL VIEGAS BUSATO

ESTUDO E ANÁLISE DA PLATAFORMA CLOUDFLARE PARA OTIMIZAÇÃO E PROTEÇÃO DE APLICAÇÕES WEB ATRAVÉS DE UM ESTUDO DE CASO

ALEGRE - ES

JANEIRO DE 2026

GABRIEL VIEGAS BUSATO

Estudo e análise da plataforma cloudflare para otimização e proteção de aplicações web através de um estudo de caso

Trabalho de conclusão de curso apresentado à Universidade Federal do Espírito Santo, como parte dos requisitos parciais para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Giuliano Prado de Morais Giglio

Universidade Federal do Espirito Santo - UFES

Departamento da Computação

Alegre - ES

Janeiro de 2026

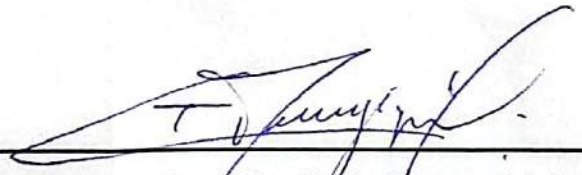
Gabriel Viegas Busato

ESTUDO E ANÁLISE DA PLATAFORMA CLOUDFLARE PARA OTIMIZAÇÃO E PROTEÇÃO DE APLICAÇÕES WEB ATRAVÉS DE UM ESTUDO DE CASO

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

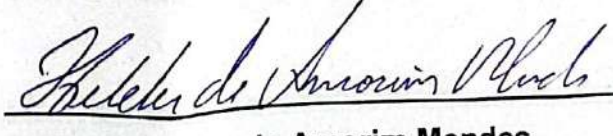
Aprovado em 04 de fevereiro de 2026.

COMISSÃO EXAMINADORA



Prof. Msc. Giuliano Prado de Morais Giglio
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Marcelo Otone Aguiar
Universidade Federal do Espírito Santo



Prof. Dr. Helder de Amorim Mendes
Universidade Federal do Espírito Santo

RESUMO

Com a rápida evolução da internet e do desenvolvimento de aplicações web, muitas aplicações emergem simultaneamente, junto com novos desenvolvedores, porém, surgem problemas de segurança, custo de infraestrutura, desempenho e otimização. O presente trabalho apresenta um estudo de caso relacionado à implementação da Cloudflare em uma aplicação web. Com o intuito de resolver os problemas citados acima foi utilizado ferramentas como CDN, WAF, Workers, Pages e o images. Além disso, foram analisadas métricas como o Core WEB Vitals, SEO e o desempenho geral com técnicas de *cache*. Junto com a *interface* de Web Analytics do Cloudflare, foi utilizado a plataforma Loader, como teste de carga, garantindo também um painel detalhado de informações. Os resultados revelaram redução de até 89% no LCP, economia de banda de 95% e queda no uso de CPU na AWS para menos de 4%, com 100% de taxa de sucesso nos testes de carga.

A aplicação desenvolvida consiste em um Cardápio Eletrônico simples e intuitivo, apresentando uma página principal com listagem de produtos, categorização e itens adicionais.

Palavras-chave: Cloudflare, computação em nuvem, desempenho, segurança, Core Web Vitals, SEO, aplicações web.

LISTA DE ABREVIAÇÕES E SIGLAS

ACM AWS Certificate Manager

AWS Amazon Web Services

CDN Content Delivery Network (Rede de Distribuição de Conteúdo)

CI/CD Continuous Integration / Continuous Deployment (Integração e Entrega Contínuas)

CLS Cumulative Layout Shift (Mudança de Layout Cumulativa)

CrUX Chrome User Experience Report (Relatório de Experiência do Usuário do Chrome)

CWV Core Web Vitals

DDoS Distributed Denial of Service (Negação de Serviço Distribuída)

EC2 Elastic Compute Cloud (Serviço de instância da AWS)

FCP First Contentful Paint (Primeira Pintura com Conteúdo)

FID First Input Delay (Primeiro Atraso na Interação)

FLT Fully Loaded Time (Tempo Totalmente Carregado)

IAM Identity and Access Management (Gerenciamento de Identidade e Acesso)

KV Key Value Store (Armazenamento de Chave-Valor)

LCP Largest Contentful Paint (Maior Pintura de Conteúdo)

OLT Onload Time (Tempo de Carregamento)

PWA Progressive Web Application (Aplicação Web Progressiva)

S3 Simple Storage Service (Serviço de Armazenamento Simples da AWS)

SEO Search Engine Optimization (Otimização para Mecanismos de Busca)

SI Speed Index (Índice de Velocidade)

SPA Single Page Application (Aplicação de Página Única)

SSL/TLS Secure Socket Layer / Transport Layer Security

TBT Total Blocking Time (Tempo Total de Bloqueio)

TTL Time to Live

WAF Web Application Firewall (Firewall de Aplicações Web)

XSS Cross-Site Scripting

LISTA DE ILUSTRAÇÕES

Figura 1 – Definição de Cloud 17
Figura 2 – Ataque DDoS 18
Figura 3 – Tela inicial AWS 19
Figura 4 – Tela inicial Akamai 20
Figura 5 – Comparação entre CDN's do mercado 21
Figura 6 – Demonstração Google Lighthouse 24
Figura 7 – Página de Web Analytics da Cloudflare 25
Figura 8 – Exemplo de criação de Pages no Cloudflare 26
Figura 9 – Exemplo de *deploy* da Cloudflare pages 27
Figura 10 – Fluxo básico loader 29
Figura 11 – Métricas da Core Web Vitals 29
Figura 12 – Fluxo de comunicação de uma API RESTful 31
Figura 13 – Fluxograma de análise da Cloudflare em uma aplicação WEB . . . 33
Figura 14 – Diagrama de casos de uso: Gerenciamento de cardápios 38
Figura 15 – Diagrama de classes da aplicação 39
Figura 16 – Modelo físico do banco de dados 40
Figura 17 – Estrutura *Backend* 44
Figura 18 – Arquivo configuração banco de dados 45
Figura 19 – Env - Chaves banco de dados 46
Figura 20 – Tela inicial - Painel Administrativo 47
Figura 21 – Estrutura *Frontend* 48
Figura 22 – Rotas *Frontend* 50
Figura 23 – Landing Page 51
Figura 24 – Cardapio 52
Figura 25 – Modal 52
Figura 26 – Informações Estabelecimento 53
Figura 27 – Pesquisa Cardápio 53
Figura 28 – Carrinho 53
Figura 29 – Checkout 54
Figura 30 – Interface do GitHub utilizada no versionamento 55
Figura 31 – Runner no GitHub 55
Figura 32 – GitHub Runner na EC2 56
Figura 33 – GitHub Variables 57
Figura 34 – Instância EC2 criada na AWS com plano gratuito 58
Figura 35 – *Deploy* YML 59
Figura 36 – Arquivo Dockerfile 60

Figura 37 – Arquivo docker-compose.yml 61
Figura 38 – Execução dos containers Docker na instância EC2 62
Figura 39 – Configuração ACM 63
Figura 40 – Load Balancer 63
Figura 41 – *Frontend* no GitHub 64
Figura 42 – Criação da aplicação 65
Figura 43 – Conexão GitHub 65
Figura 44 – Configuração ambiente e variáveis 66
Figura 45 – Build Cloudflare e *Deploy* 67
Figura 46 – Listagem de implementações 68
Figura 47 – Web Analytics do *deploy* 68
Figura 48 – Domínio na Cloudflare 69
Figura 49 – DNS na Cloudflare 69
Figura 50 – Pages Domínio personalizado 69
Figura 51 – Configuração do teste 70
Figura 52 – Resultado do teste no loader 71
Figura 53 – Resultado da requisições HTTP 72
Figura 54 – Desempenho PageSpeed 74
Figura 55 – Desempenho PageSpeed Images 75
Figura 56 – Desempenho Lighthouse 75
Figura 57 – Imagens não otimizadas 76
Figura 58 – Resultado: Core Web Vitals - Web Analytics 79
Figura 59 – Resultado: EC2 80
Figura 60 – Resultado: Load Balancer 80
Figura 61 – Resultado Loader - Pós Otimização 81
Figura 62 – Resultado Requisições HTTP 82
Figura 63 – Imagens hospedadas no Cloudflare 84
Figura 64 – Cloudflare Images *Variants* 84
Figura 65 – Criando Variant 85
Figura 66 – Corrigindo TTL das *variants* 86
Figura 67 – Otimização de velocidade 86
Figura 68 – Polimento de images pelo Cloudflare 87
Figura 69 – Imagens após otimização 88
Figura 70 – Site após otimização das imagens mantém qualidade 89
Figura 71 – PageSpeed pós otimização 90
Figura 72 – Rotas Worker 91
Figura 73 – Código do Worker 92
Figura 74 – Cache genérico Cloudflare 93
Figura 75 – Cache em camada 94

Figura 76 – Cache reserva 95
Figura 77 – Configurações adicionais de desempenho da Cloudflare 96
Figura 78 – Mais configurações da Cloudflare 97
Figura 79 – Web Analytics pós otimização 99
Figura 80 – Resultado: EC2 pós otimização 100
Figura 81 – ELB pós otimização 101
Figura 82 – Métricas Worker 101
Figura 83 – Dados de acesso Worker 102
Figura 84 – Worker Live Page 103
Figura 85 – Worker Cache Hit 103
Figura 86 – Cache Analytics 104

LISTA DE QUADROS

1 Métricas de software web 17
2 Comparação de funcionalidades entre AWS, Cloudflare e Akamai . . . 20
3 Comparação entre Cloudflare, Akamai e AWS 22
4 Tabela: organizations 41
5 Descrição da estrutura de pastas da aplicação *Backend* 44
6 Descrição da estrutura de pastas da aplicação *Frontend* 49
7 Resultados do teste Loader.io - 60.000 requisições (Cenário ANTES) . 73
8 Configuração do teste de desempenho com Puppeteer 77
9 Resultados do teste de desempenho com Puppeteer - 200 usuários . . 78
10 Resultados do teste Loader.io - 60.000 requisições (Cenário DEPOIS) . 83
11 Resultados do teste de desempenho com Puppeteer - 200 usuários . . 98
12 Tabela: organization_types 113
13 Tabela: products 113
14 Tabela: product_histories 114
15 Tabela: customers 114
16 Tabela: categories 115
17 Tabela: organization_users 115
18 Tabela: product_additionals 115
19 Tabela: organization_addresses 116
20 Tabela: customer_addresses 116
21 Tabela: organization_configurations 117
22 Tabela: category_product 117
23 Tabela: product_products_additional 117

SUMÁRIO

LISTA DE ILUSTRAÇÕES 6

LISTA DE QUADROS 10

1 INTRODUÇÃO 13
1.1 O problema e sua importância 14
1.2 Objetivos 14
1.2.1 Objetivo geral 14
1.2.2 Objetivos específicos 15

2 REVISÃO DE LITERATURA 16
2.1 Fundamentos da computação em nuvem 16
2.1.1 Impacto da Cloudflare na nuvem 16
2.2 Desempenho e otimização de aplicações web 16
2.3 Segurança das aplicações web 18
2.4 Plataformas semelhantes a Cloudflare 18
2.4.1 AWS 18
2.4.2 Akamai 19
2.5 Comparativo entre plataformas 19
2.6 Tecnologias que serão utilizadas 22
2.6.1 Ferramentas utilizadas no backend 22
2.6.2 Ferramentas utilizadas no frontend 23
2.6.3 Google Lighthouse 23
2.6.4 Google PageSpeed Insights 24
2.6.5 Cloudflare Web Analytics 24
2.6.6 Cloudflare Worker 25
2.6.7 Cloudflare Page 25
2.6.8 Cloudflare WAF 27
2.6.9 Cloudflare Images 28
2.6.10 Amazon EC2 28
2.6.11 Github 28
2.6.12 Github Actions 28
2.6.13 Loader 28
2.7 Métricas da WEB 29
2.8 Impacto do SEO na experiência dos usuários 30
2.9 Padrão Arquitetural: API RESTful 30

3 METODOLOGIA 32
3.1 Fluxo de desenvolvimento e ferramentas utilizadas 32
3.1.1 Configuração do *Backend* no Amazon EC2 33
3.1.2 Configuração do *Frontend* na Cloudflare 33
3.1.3 Configuração de Cache, Workers e Minificação do CSS/JS . . . 34
3.1.4 Configuração de Firewall e Segurança na Cloudflare 34
3.1.5 Simulação de Usuários com Loader.io 34
3.1.6 Análise de métricas e relatório do antes e depois 34
3.1.6.1 Loader 34
3.1.6.2 Cloudflare Web Analytics 35
3.1.6.3 Google PageSpeed e Lighthouse 35
3.2 Requisitos do sistema 35
3.2.1 Requisitos funcionais 36
3.2.2 Requisitos não funcionais 37
3.3 Diagrama de casos de uso 37
3.4 Modelagem de Domínio 38
3.4.1 Diagrama de classe 39
3.5 Modelo de banco de dados 40
3.5.1 Modelo Físico 40

4 DESENVOLVIMENTO DO ESTUDO DE CASOS 42
4.1 A aplicação 42
4.1.1 *Backend* 42
4.1.1.1 Estrutura do projeto 42
4.1.1.2 Conexão com banco de dados 45
4.1.1.3 Painel Administrativo 46
4.1.2 *Frontend* 47
4.1.2.1 Estrutura do projeto 48
4.1.2.2 Rotas de Navegação 50
4.1.2.3 *Interfaces* 51
4.2 Configuração das Aplicações 54
4.2.1 Configuração *Backend* na EC2 54
4.2.1.1 Versionamento com GitHub 54
4.2.1.2 GitHub Runner 55
4.2.1.3 GitHub Variables 56
4.2.1.4 Criação Instância EC2 na AWS 57
4.2.1.5 Integração e Entrega Contínua com GitHub Actions 58
4.2.1.6 DockerFile 60
4.2.1.7 Docker Compose 60
4.2.1.8 Execução do *Backend* na EC2 62

4.2.1.9 Configuração do AWS Certificate Manager 62
4.2.1.10 Configuração do Load Balancer 63
4.2.2 Deploy do *Frontend* na Cloudflare 63
4.2.2.1 Configuração Domínio na Cloudflare 69

5 MONITORAMENTO E COLETA DE MÉTRICAS DA CLOUDFLARE E AWS . . 70
5.1 Testes e Métricas antes da configuração Cloudflare 70
5.1.1 Resultados dos testes no Lighthouse e PageSpeed 74
5.1.2 Teste de Desempenho com *Puppeteer* 77
5.1.2.1 Resultados Teste do Puppeteer 78
5.1.2.2 Resultado Puppeteer: Análise dos Dados do Google Analytics - Core Web Vitals 79
5.1.2.3 Resultado Puppeteer: Análise EC2 79
5.1.2.4 Resultado Puppeteer: Análise Load Balancer 80
5.2 Testes e Métricas após a configuração da Cloudflare 81
5.2.1 Resultado Loader - Pós otimização 81
5.2.2 Configuração Cloudflare Images 83
5.2.3 Resultado dos testes no Lighthouse - PageSpeed 89
5.2.4 Implementação do Cloudflare Worker para cache 90
5.2.5 Cache genérico inteligente da aplicação pela Cloudflare 92
5.2.5.1 Cache em Camadas (Tiered Cache) 93
5.2.5.2 Cache Reserve 94
5.2.6 Configurações adicionais de desempenho 95
5.2.7 Resultados dos Testes com Puppeteer - Pós Configuração . . . 97
5.2.7.1 Resultado Puppeteer: Análise dos Dados do Google Analytics - Core Web Vitals 98
5.2.7.2 Resultado Puppeteer: Análise EC2 99
5.2.7.3 Resultado Puppeteer: Análise Load Balancer 100
5.2.7.4 Resultado Puppeteer: Análise Worker 101
5.2.7.4.1 Monitoramento em Tempo Real (Live Tail) 102
5.2.7.5 Resultado Puppeteer: Cache Analytics 103

6 RESULTADOS ALCANÇADOS E CONSIDERAÇÕES FINAIS 105
6.1 Análise dos Resultados Obtidos 105
6.2 Considerações finais 107
6.2.1 Trabalhos futuros 107

REFERÊNCIAS 109

APÊNDICE A - Descrição das tabelas do banco de dados no modelo físico . 112

1 Introdução

Com o avanço tecnológico e o crescimento exponencial das aplicações web, a dependência da internet intensificou-se, impulsionando a criação de *frameworks* especializados e tecnologias complementares para atender a essa demanda crescente (MOLINA-RíOS; PEDREIRA-SOUTO, 2019). Essa evolução impulsionou o uso de tecnologias complementares, como serviços web, que oferecem soluções eficientes para entrega de conteúdos e incorporam medidas avançadas de proteção contra ataques, garantindo maior segurança e resiliência no ambiente digital (GUPTA; MITTAL; MUFTI, 2024).

De forma emergente, os avanços tecnológicos, apoiados por redes digitais com capacidade robusta, revolucionaram o setor empresarial nas últimas décadas. Nesse cenário, os sistemas de informação se consolidaram como os elementos principais das atividades socioeconômicas, influenciando diretamente a gestão dos recursos e entrega de conteúdo. Além disso, ao explorar a conectividade global e a capacidade da computação em nuvem, tais sistemas superaram as fronteiras físicas e ampliaram significativamente o alcance operacional das organizações (RAFIQUE et al., 2011).

Em escala econômica global, a computação em nuvem apresenta um impacto comparável ao da eletricidade, transformando profundamente a forma como empresas operam e gerenciam seus recursos. Contudo, sua adoção não é isenta de desafios, como questões de segurança, falta de visibilidade da infraestrutura e limitações de orçamento para iniciativas de proteção (NADEEM et al., 2022).

No entanto, a escolha inadequada de um provedor ou serviço em nuvem pode resultar em uma infraestrutura altamente complexa e com custos excessivamente elevados. Isso ocorre porque a personalização ou uso de serviços desnecessários pode ampliar despesas e dificultar a gestão eficiente do ambiente de TI. Assim, é crucial que as empresas avaliem cuidadosamente suas necessidades específicas e escolham soluções que se alinhem ao seu porte, orçamento e objetivos estratégicos, priorizando fornecedores que ofereçam transparência, flexibilidade e custo-benefício. Esse alinhamento é essencial para evitar gastos desproporcionais e garantir que os investimentos em nuvem gerem retornos significativos e sustentáveis. Essa democratização da tecnologia permite que *startups* e pequenos projetos, em poucos anos, alcancem patamares que antes eram considerados inacessíveis, como o uso de serviços em nuvem, anteriormente exclusivos para grandes organizações (STEGAROIU, 2019).

A nuvem, como uma ferramenta extremamente poderosa, está acessível tanto para grandes corporações quanto para pequenas empresas em todo o mundo. Quando utilizada sob uma perspectiva orientada aos negócios, ela oferece soluções robustas

para armazenamento de dados, segurança, distribuição de conteúdos, entrega de serviços, entre outros. Essas funcionalidades não apenas impulsionam o crescimento empresarial e econômico, mas também criam um ambiente competitivo de alto nível. Além de fomentar o mercado, essa acessibilidade incentiva a inovação, possibilitando que empresas de todos os tamanhos alcancem um crescimento econômico significativo e sustentável (HASSAN et al., 2024).

1.1 O problema e sua importância

Diante das vantagens oferecidas pela nuvem, diversos serviços apareceram, tornando a infraestrutura cada vez mais complexa. Antigamente as equipes de TI se preocupavam na organização do ambiente, mas devido ao crescimento das necessidades empresariais e também com a introdução ao trabalho remoto em larga escala, as empresas passaram a ter mais responsabilidade em quesito de gerenciamento, como serviços em nuvem, serviços das aplicações, como SGBDs (Sistema Gerenciador de Banco de Dados), filas de mensageria, entre outros, além de lidar com a segurança das redes internas e externas (CLOUDFLARE, 2023a).

Neste cenário, tais serviços surgem como uma alternativa para reduzir a carga operacional das equipes de TI, principalmente no que tange à gestão centralizada de recursos. Devido a isso, a Cloudflare, com funcionalidades como CDN (rede de distribuição de conteúdo, tradução nossa), WAF (firewall de aplicações web, tradução nossa), proteção contra ataques DDoS (negação de Serviço Distribuída, tradução nossa), e nuvem de conectividade, acaba se consolidando como uma escolha estratégica para aqueles que buscam agilidade e eficiência em suas operações digitais (CLOUDFLARE, 2023b).

Segundo Ogul Tutuncu, Engenheiro de DevOps em um estudo de caso publicado pela Cloudflare, “O que normalmente nos tomaria mais de um minuto com outros fornecedores, com a Cloudflare demora segundos, além da fácil navegação. Essa eficiência é inigualável e não requer nenhuma especialização no assunto para ser executada“ (CLOUDFLARE, 2024b).

1.2 Objetivos

Esta seção apresenta as ideias principais da pesquisa por meio de seus objetivos, estabelecendo o roteiro para a obtenção dos resultados.

1.2.1 Objetivo geral

A implementação de um conjunto definido de serviços da Cloudflare em uma aplicação web, com base em um estudo de caso avaliativo. A aplicação utiliza os ser-

viços de CDN (*Content Delivery Network*), WAF (*Web Application Firewall*), proteção contra ataques DDoS, Cloudflare Workers e Cloudflare Pages. O objetivo principal é investigar como esses serviços, em conjunto, conseguem contribuir diretamente para o aumento da segurança e também da performance.

Esta pesquisa investiga como a integração dos serviços da Cloudflare, em específico a CDN, WAF, proteção contra DDoS, Cloudflare Workers e Pages. A análise se fundamenta na implementação prática dessas ferramentas em um estudo de caso avaliativo, visando mensurar os resultados objetidos por meio dessa arquitetura conjunta.

1.2.2 Objetivos específicos

- **Infraestrutura da aplicação**: Implementação das regras no WAF para bloquear ataques, injeções SQL e ataques XSS.
- **Otimização da aplicação web**: Configuração do *cache*, o qual irá reduzir a carga no *backend*, diretamente para as APIs mais acessadas, além de serviços de imagens, como os serviços de redimensionamento e compressão sem perda.
- **Melhoria na experiência do usuário**: Reduzir o tempo de carregamento com uso de páginas e CDN. Com o intuito de garantir a otimização da métrica de CWV (Principais Vitais da Web, tradução nossa), garantindo boas métricas para o LCP (velocidade de renderização do maior conteúdo, tradução nossa) e FID (primeiro atraso na interação, tradução nossa), os quais são cruciais para garantir um melhor SEO (Otimização de mecanismos de pesquisa, tradução nossa).
- **Análise de dados e relatórios**: Foi utilizado o Cloudflare Web Analytics para análise gráfica. Foi feita uma comparação do impacto dos serviços da Cloudflare no desempenho da aplicação, analisando métricas antes e depois da implementação, como SEO, segurança de rede, gargalos no *backend* que podem ser resolvidos com os Cloudflare Workers, além do detalhamento das ameaças bloqueadas pelo WAF, como XSS e injeção SQL.

2 Revisão de literatura

Esta seção tem como principal objetivo apresentar o referencial teórico do trabalho, abordando os principais conceitos e temas relacionados à pesquisa.

2.1 Fundamentos da computação em nuvem

Como modelo de implantação baseado em virtualização demonstrado pela figura 1, a computação em nuvem transformou significativamente a forma como os recursos de *hardware* são utilizados, e com isso elevando a inovação tecnológica a outros patamares. Essa inovação proporcionou uma maior flexibilidade em termos de paradigmas e modelos computacionais. Contudo, a forma de distribuição de recursos também foi afetada, de uma forma em que novos modelos podem alterar o rumo além da tecnologia, mas como também empresarial e comercial (BÖHM et al., 2011).

Essa infraestrutura em nuvem obriga que empresas tomem a inovação tecnológica como o meio principal de se destacar no mercado, com o objetivo de redução de custo e crescimento. A qual aumenta a expectativa de lucro e uma melhor forma de manipular recursos, além da possibilidade de se padronizar, com isso evitando a necessidade da duplicidade em termos de administração, configuração e suporte, levando essa mesma estrutura para outras partes (IBM, 2007).

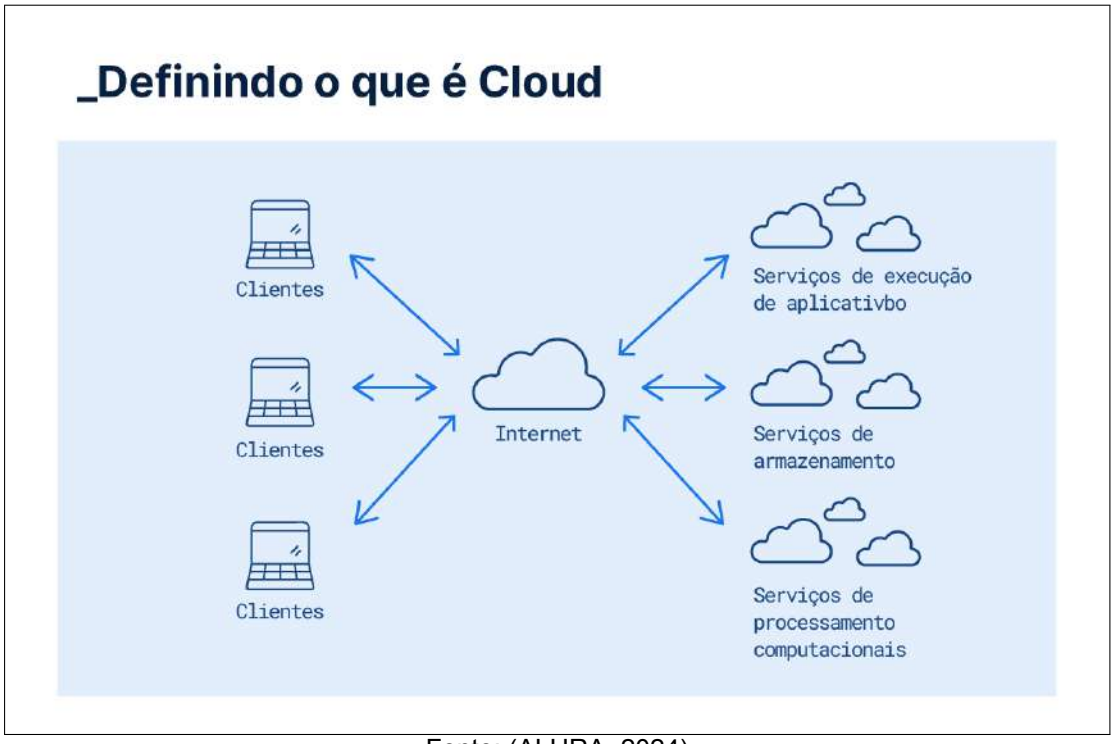
2.1.1 Impacto da Cloudflare na nuvem

A Cloudflare tem se especializado cada vez mais na segurança da nuvem e entrega de conteúdo, com foco direto em otimizar aplicações webs e garantir confiabilidade nos serviços. Entretanto, a Cloudflare não apenas provê serviços, mas também age como intermediário entre o usuário final e a aplicação na nuvem, com isso conseguindo garantir, por meio dos seus serviços, maior agilidade e segurança na entrega de informações. Diante de todas as facilidades que propõe, também entra a questão da usabilidade, o qual possui uma *interface* de usuário robusta e de fácil uso, permitindo que empresas de todo o mundo consigam configurar infraestruturas robustas para suas aplicações (BAJRAMI; IDRIZI; RUSHITI, 2024).

2.2 Desempenho e otimização de aplicações web

Com o alto número de *frameworks*, muitas aplicações são criadas e publicadas todos os dias, porém, à parte de se ter uma aplicação web surge a necessidade de preservar a performance, usabilidade, acessibilidade, entre outros (AL-DULAIMI et al.,

Figura 1 – Definição de Cloud



Fonte: (ALURA, 2024)

2024). Mediante isso, surgem novas métricas de *performance* para *softwares*, com indicadores primários e secundários, os quais podem ser vistos no quadro 1.

Quadro 1 – Métricas de software web

Primeiro nível (Métricas)	**Segundo nível (Indicadores)**	**Siglas**
Indicadores primários	Maior Pintura de Conteúdo (Largest Contentful Paint)	LCP
	Primeiro Atraso de Entrada (First Input Delay)	FID
	Mudança de Layout Cumulativa (Cumulative Layout Shift)	CLS
	Velocidade de Carregamento da Página (Page Load Speed)	PLS
Indicadores secundários	Primeira Pintura com Conteúdo (First Contentful Paint)	FCP
	Índice de Velocidade (Speed Index)	SI
	Tempo Total de Bloqueio (Total Blocking Time)	TBT
	Tempo de Carregamento (Onload Time)	OLT
	Tempo Totalmente Carregado (Fully Loaded Time)	FLT
	Maior Atraso de Entrada (Largest Input Delay)	LID

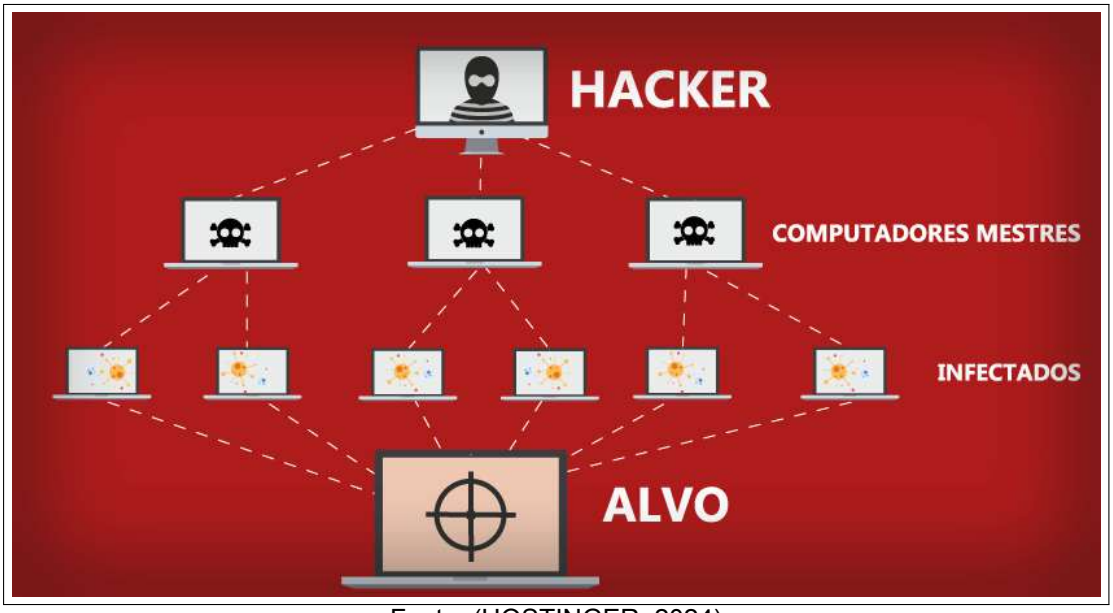
Fonte: Adaptado de (AL-DULAIMI et al., 2024).

2.3 Segurança das aplicações web

Desde o primeiro ataque DDoS, o qual teve um aumento significativo com o passar dos anos, a nuvem se encontra em ameaça, afetando não somente ela, mas como também toda a internet. Entretanto, atualmente temos uma característica peculiar nesses ataques, os quais são feitos por outros serviços online e possuem um baixo custo, comparado com o prejuízo que podem causar. O impacto crucial está no quesito corporativo e empresarial, afetando diretamente nos serviços da empresa e podendo também, afetar os dados e informações (JOHN et al., 2023).

A figura 2 exemplifica a atuação de um hacker em uma rede de computadores, e como o ataque pode se espalhar de forma escalável.

Figura 2 – Ataque DDoS



Fonte: (HOSTINGER, 2024)

2.4 Plataformas semelhantes a Cloudflare

Foi descrito nesta seção ferramentas que possuem funcionalidades semelhantes à Cloudflare. A análise de ferramentas semelhantes torna possível a identificação de como um sistema se destaca de outro.

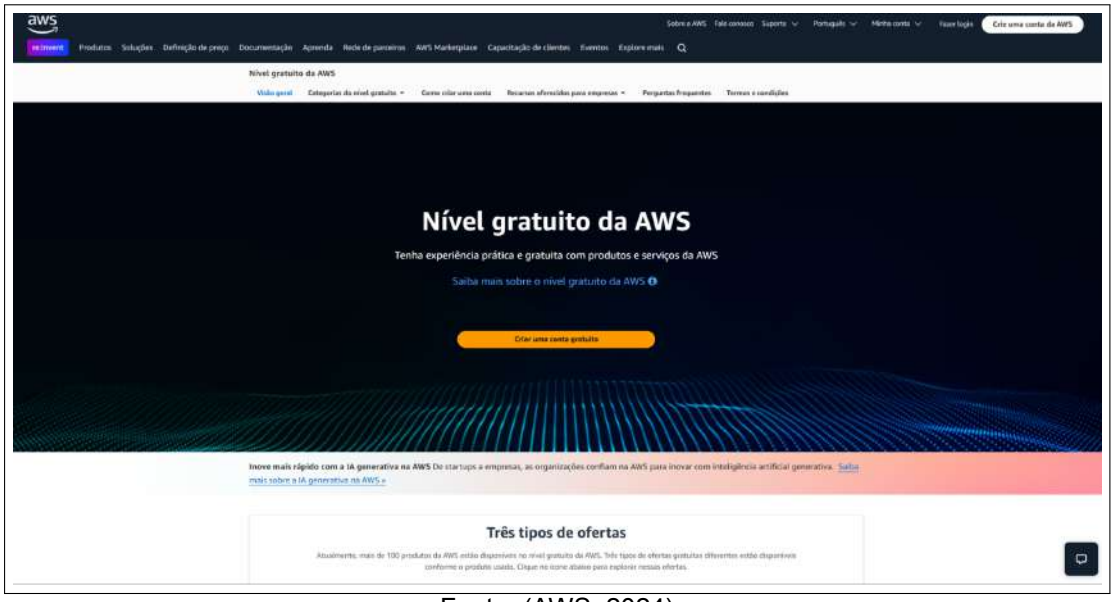
2.4.1 AWS

A Amazon Web Services (AWS, 2024) é uma plataforma mais ampla que engloba serviços como o da Cloudflare e outros. Oferece escalabilidade, elasticidade, customização para infraestrutura, e uma extensa documentação. A cloudflare possui integração a AWS com bastante facilidade, hospedar seus aplicativos de forma segura.

Provendo desempenho mais rápido, além de balanceamento de carga em implementações de S3, EC2, entre outros.

A figura 3 mostra a página inicial da Amazon Web Services.

Figura 3 – Tela inicial AWS



Fonte: (AWS, 2024)

2.4.2 Akamai

A Akamai Cloud Computing (AKAMAI, 2024) é uma empresa americana que foi fundada em 1998, contando com inúmeras soluções em segurança, desempenho e entrega de conteúdo para empresas do mundo todo. Além disso, possui um amplo conjunto de serviços de *Edge Computing* e nuvem distribuída, que inclui, máquinas virtuais, unidades de processamento gráfico, armazenamento, banco de dados em nuvem, otimização de rede e serviços de segurança, além de funções na borda da rede.

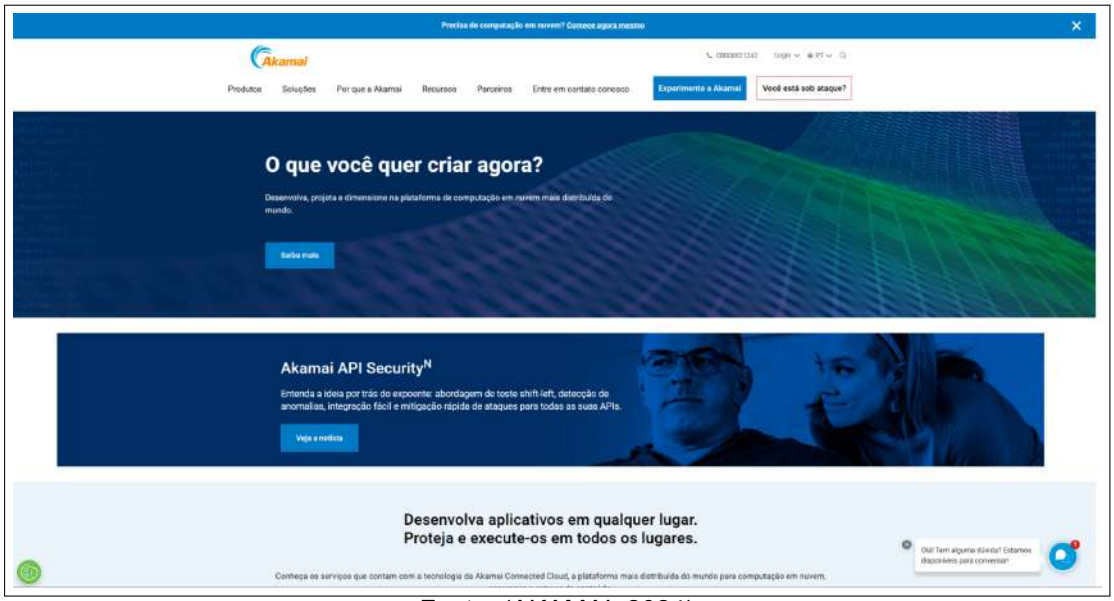
Os principais objetivos são os de implementar a redução de latência, diminuir o tempo de carregamento de conteúdo, fornecer segurança avançada, proteger aplicações contra ataques DDoS, injeção de SQL entre outros, além de possuir serviços de *streaming*.

A figura 4 mostra a página inicial da Akamai

2.5 Comparativo entre plataformas

Esta seção apresenta o comparativo de funcionalidades e preços dos serviços apresentados neste estudo, os quais podem ser vistos no quadro 2.

Figura 4 – Tela inicial Akamai



Fonte: (AKAMAI, 2024)

Quadro 2 – Comparação de funcionalidades entre AWS, Cloudflare e Akamai

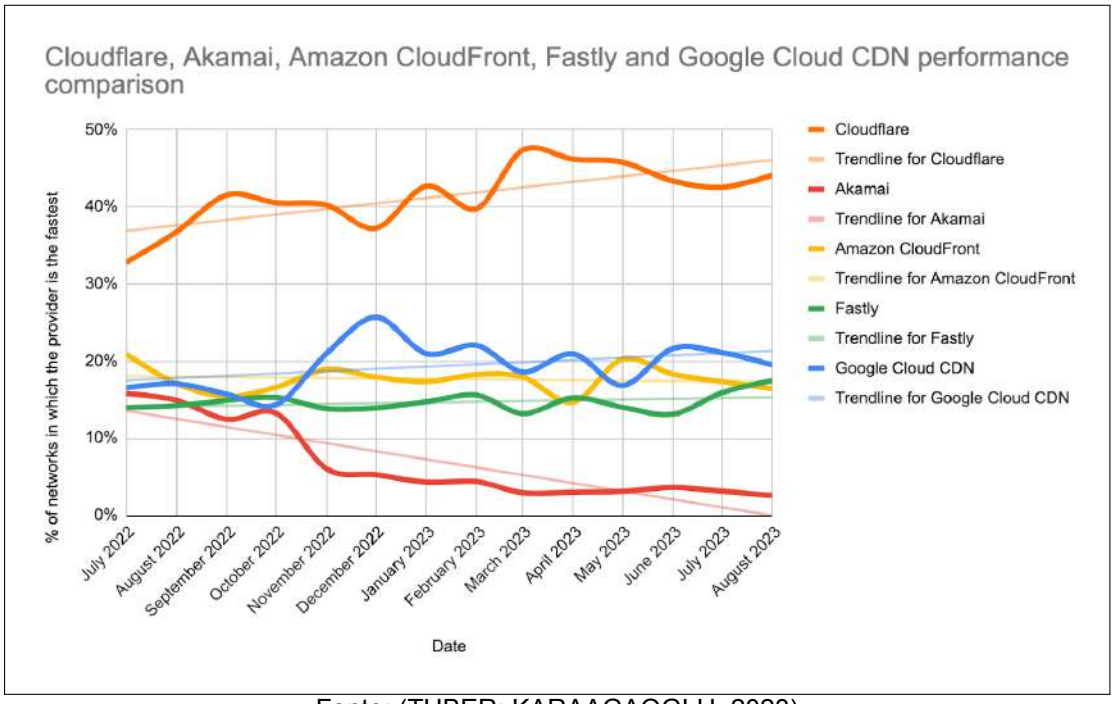
Funcionalidades	**AWS**	**Cloudflare**	**Akamai**
Rede de entrega de conteúdo (CDN)	SIM	SIM	SIM
Firewall de Aplicação Web (WAF)	SIM	SIM	SIM
Proteção contra DDoS	SIM	SIM	SIM
Funções *serverless*	SIM	SIM	SIM
Armazenamento de dados	SIM	SIM	SIM

Fonte: Adaptado de (AWS, 2024), (CLOUDFLARE, 2023b), (AKAMAI, 2024).

Como pode ser visto pelo quadro 2, todas as 3 plataformas possuem todos os serviços citados, porém, o que diferencia plataformas atualmente se dá mais por questões financeiras e performance.

Diante de serviços como CDN, a Cloudflare sai à frente. Como resumo, em 2023 a Cloudflare alcançou 44% a mais como o provedor mais rápido, como pode ser visto na figura 5 (TUBER; KARAAGAOGLU, 2023)

Figura 5 – Comparação entre CDN’s do mercado



Fonte: (TUBER; KARAAGAOGLU, 2023)

O quadro 3 demonstra comparações entre Cloudflare, Akamai e a AWS. Contando com preço, facilidade de uso, recursos de segurança e comentários de usuários.

Quadro 3 – Comparação entre Cloudflare, Akamai e AWS

Funcionalidade	**Cloudflare**	**Akamai**	**AWS**
Preço	Plano gratuito, planos pagos começam em 20 dólares mensais	Planos customizados, com preços altos na configuração inicial	Pague pelo uso (*pay-as-you-go*)
Facilidade de uso	Facilidade de implantação, *interface* amigável	Configuração complexa, feito para uso empresarial	Configuração fácil, porém possui customização em muitos serviços, o que torna confuso
Recursos de Segurança	Proteção DDoS, SSL, WAF e mais alguns recursos gratuitos de segurança	Proteção contra DDoS avançada, *firewall* de aplicação web	Proteção contra DDoS com AWS Shield, SSL/TLS com AWS Certificate Manager, WAF avançado e serviços de IAM
Comentários de Usuários	Eficiência de custo e facilidade de integração com outros serviços em nuvem	De bastante confiabilidade, mas alguns usuários acham complexo	Possui bastante flexibilidade entre os serviços, porém, pode se tornar muito custoso

Fonte: Adaptado de (GICELLE, 2024).

2.6 Tecnologias que serão utilizadas

Nesta seção foram indicadas as ferramentas que foram utilizadas no estudo de caso, em uma aplicação de Cardápio Eletrônico.

2.6.1 Ferramentas utilizadas no backend

A. **PHP** - Uma linguagem de script embutida no HTML. O qual significa *HyperText Preprocessor*, muito utilizado para desenvolvimento de aplicações web (PHP, 2024).

B. **MySQL** - É um banco de dados de código aberto, no caso um Sistema Gerenciador de Banco de Dados, o qual inclui muitos recursos de gerenciamento (MYSQL, 2024).

C. **Laravel** - Laravel é um *framework* web escrito em PHP, estruturado para facilitar o desenvolvimento de aplicações com arquitetura MVC (Model-View-Controller), oferecendo recursos como roteamento, ORM, autenticação, entre outros. (LARAVEL, 2024).

D. **Laravel Filament** - Laravel Filament é um *framework full stack UI*, o qual utiliza a *stack* TALL (Tailwind, Alpine JS, Laravel e Livewire). Possui uma incrível gama de componentes customizáveis e de fácil utilização, sendo possível criar um painel administrativo ou qualquer outro tipo de aplicação rapidamente (FILAMENT, 2024).

2.6.2 Ferramentas utilizadas no frontend

A. **CSS** - Folha de estilo em cascata, o qual é possível criar estilos para páginas web (MOZILLA, 2024).

B. **Vue.js** - *Framework* progressivo que possibilita o desenvolvimento de *interfaces* de usuário. É focado exclusivamente no desenvolvimento da camada visual (VUE.JS, 2024).

C. **Quasar** - Quasar é um *framework* vue.js, o que possui muitas ferramentas de desenvolvimento web, e também paradigmas, sendo o SPA, SSR, PWA, entre outros (QUASAR, 2024).

D. **Axios** - Axios é um cliente HTTP baseado em promessa, o qual utiliza o NODE JS. Possibilitando fazer requisições HTTP (AXIOS, 2024).

2.6.3 Google Lighthouse

Consiste em uma ferramenta com o propósito de melhorar a qualidade das páginas web, contém auditorias de desempenho, SEO entre outras. O Lighthouse é incorporado ao Chrome DevTools, o qual é possível utilizar de forma fácil e com apenas um clique (LIGHTHOUSE, 2024).

A figura 6 representa a interface de configuração para gerar o relatório no Google Lighthouse. Na mesma tela é possível selecionar diferentes categorias de análise, definir o modo que será gerado o relatório, e além disso, escolher o tipo do dispositivo a ser simulado. Com essas opções disponíveis, é possível personalizar a auditoria

conforme a necessidade do usuário, proporcionando uma análise detalhada do desempenho, acessibilidade e SEO.

Figura 6 – Demonstração Google Lighthouse

Fonte: (LIGHTHOUSE, 2024)

2.6.4 Google PageSpeed Insights

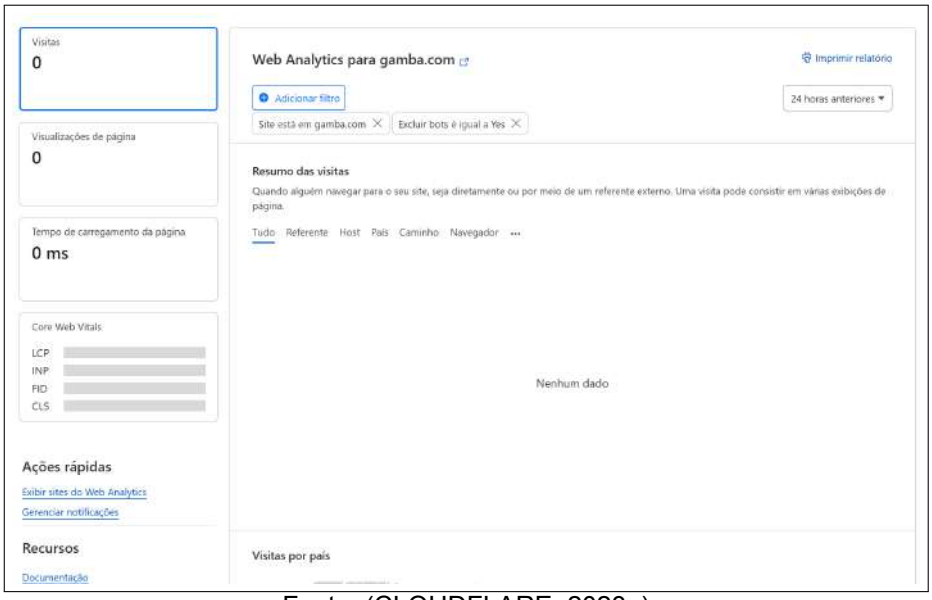
O Google PageSpeed é uma ferramenta para análise de desempenho em aplicações web, o qual gera um relatório CrUX para exibição do desempenho geral de uma página. O mesmo inclui um padrão de pontuação, sendo boa, média e ruim (Google PageSpeed, 2024).

2.6.5 Cloudflare Web Analytics

É uma ferramenta de análise de dados, com ela é possível gerenciar todos os levantamentos estatísticos da sua aplicação no Cloudflare, desde a solicitação até a resposta (CLOUDFLARE, 2023a)

A figura 7 representa a interface principal do Web Analytics. Nesta interface é possível identificar indicadores importantes, como o número de visitas, visualizações da página, tempo de carregamento inicial, tanto como o Core Web Vitals. Além disso, o painel também gera gráficos de desempenho.

Figura 7 – Página de Web Analytics da Cloudflare



Fonte: (CLOUDFLARE, 2023a)

2.6.6 Cloudflare Worker

O Cloudflare Worker tem um único objetivo, o qual é executar funções sem servidor. Em outras palavras, o próprio código que está fora do servidor, é armazenado em cache, o qual é apenas executado quando o serviço é solicitado. Esse mesmo código é hospedado em algum servidor da Cloudflare (CLOUDFLARE, 2023a).

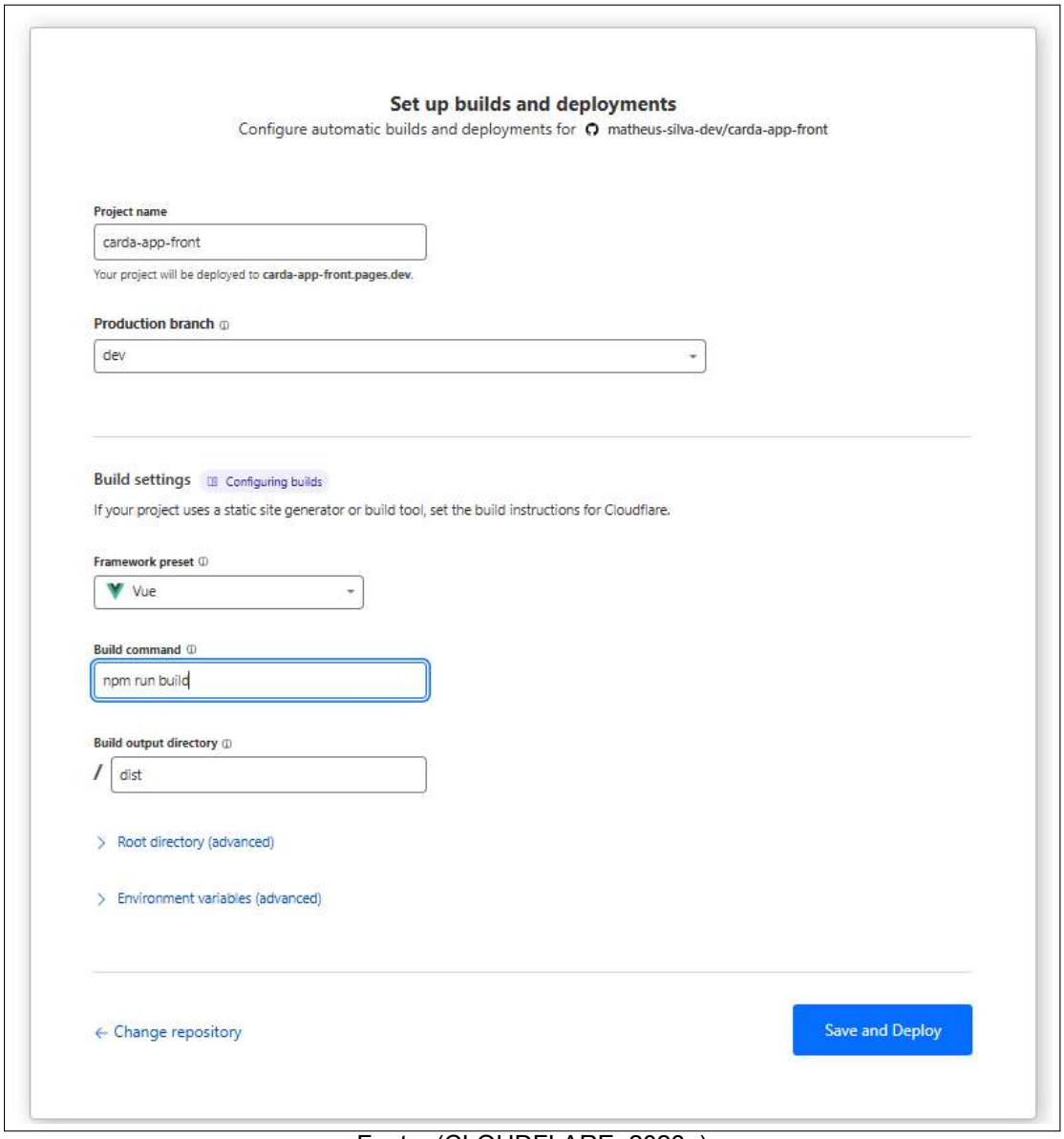
2.6.7 Cloudflare Page

De forma a solucionar o *deploy* contínuo e simplificado, a Cloudflare entra com o Pages, que consiste em manter a técnica de CI/CD para *frontends* estáticos, possuindo integração direta ao GitHub, e gerando uma publicação mais simplificada e com menor complexidade possível, sem necessidade de configuração da *pipeline* e servidores.

O Cloudflare Pages possui suporte para diversas ferramentas modernas de desenvolvimento, o que permite projetos criados com *frameworks* atuais como Next.js, React, Vue.js entre outros que possam ser implementados de forma simples. Além disso, ele fornece um ambiente de pré-visualização para cada *commit*, facilitando a colaboração entre os membros da equipe, e também é integrado diretamente com o Cloudflare Workers.

Diante disso, será utilizado o pages para prover a instância necessária da aplicação que irá passar pelo estudo de caso. A configuração do Pages, a seleção do *build* do seu projeto e os detalhes do processo de implantação podem ser vistos na figura 8, onde são apresentadas as etapas de configuração e os parâmetros utilizados para garantir o *deploy*.

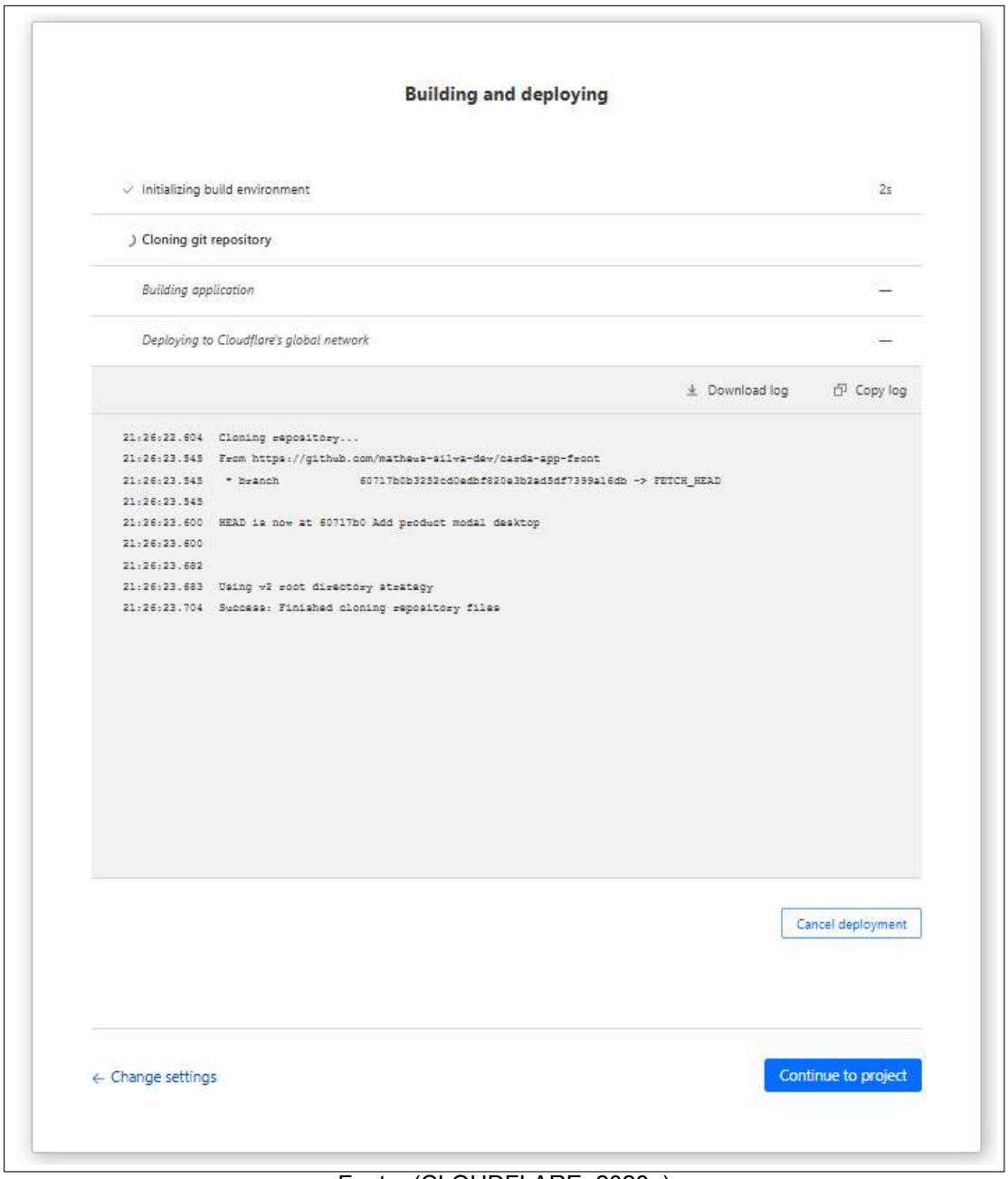
Figura 8 – Exemplo de criação de Pages no Cloudflare



Fonte: (CLOUDFLARE, 2023a)

A figura 9 mostra como é feito o *deploy* da pages da Cloudflare, com uma configuração simples e clara, com a inicialização das variáveis de ambiente, o clone do repositório do Github e com a última etapa sendo o *deploy* na rede da Cloudflare.

Figura 9 – Exemplo de *deploy* da Cloudflare pages



Fonte: (CLOUDFLARE, 2023a)

2.6.8 Cloudflare WAF

Firewall de aplicativos web, o qual protege a aplicação contra ataques maliciosos que exploram vulnerabilidades. É utilizado para fornecer cobertura para uma ampla gama de ameaças (CLOUDFLARE, 2023a).

2.6.9 Cloudflare Images

Uma solução robusta de ponta a ponta que inclui a eficiência no armazenamento e entrega de imagens, além da possibilidade de otimizar imagens fora dos servidores da Cloudflare (CLOUDFLARE, 2023a).

2.6.10 Amazon EC2

É um serviço oferecido pela Amazon, o qual consiste em um servidor na nuvem. É possível configurar a instância para deixar a sua aplicação *backend* no ar. As instâncias dependem da forma como o EC2 é configurado, mantendo o equilíbrio entre recursos como memória, armazenamento e rede (AWS, 2024).

2.6.11 Github

É um sistema de controle de versão de código fonte, com possibilidade de se criar várias ramificações de códigos, de acordo com o tempo. Também facilita a manutenção do projeto, contendo todas as possíveis alterações, e classificando cada alteração com um *hash* único (GITHUB, 2024).

2.6.12 Github Actions

Consiste em uma plataforma de integração e entrega contínuas, permitindo a utilização de compilação, testes de software e *deploy*. Com a possibilidade da criação de fluxos de trabalho de ponta a ponta (GITHUB, 2024).

2.6.13 Loader

Loader (LOADER, 2024) é uma plataforma de testes de carga, o qual é possível avaliar o desempenho e a escalabilidade das aplicações webs. Provendo uma *API RESTful* como mencionado na seção 2.9, permite que simule conexões simultâneas, com a possibilidade de monitorar os dados. A mesma foi desenvolvida pela empresa SendGrid, com infraestrutura baseada em nuvem.

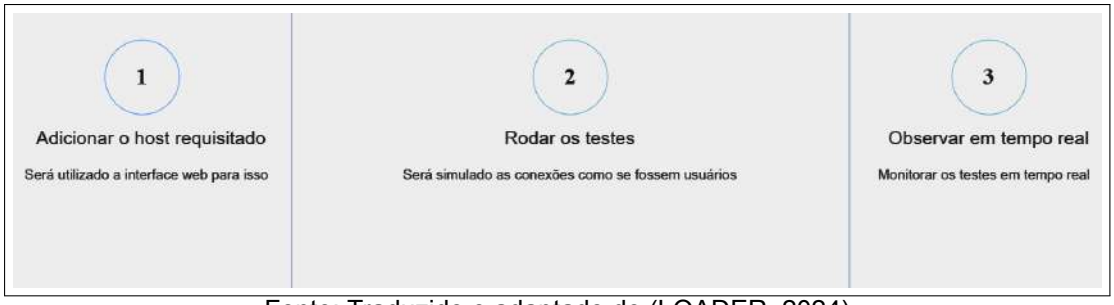
A imagem 10 representa o fluxo básico da plataforma, o qual envolve três etapas principais:

A. **Configuração de host**: É possível configurar o host em que será aplicado os testes, onde também é possível definir todos os parâmetros, como endereço do servidor, quais portas serão usadas, tempo de duração, entre outras coisas.

B. **Simulação de conexões de usuários**: O próximo passo diz respeito a parte de conexões concorrentes ao host fornecido.

C. **Métricas em tempo real** - E por fim, durante a simulação dos testes, a plataforma coleta dados e por fim exibe métricas detalhas sobre o desempenho do sistema.

Figura 10 – Fluxo básico loader



Fonte: Traduzido e adaptado de (LOADER, 2024)

2.7 Métricas da WEB

Iniciado pela Google com o intuito de se criar métricas e indicadores de qualidades comuns para as aplicações web, surge o Core Web Vitals, o qual simplifica todas as ferramentas disponíveis para esse tipo de análise de *performance*. O Core Web Vitals se consolida como um conjunto de indicadores que são aplicados para qualquer aplicação web (GOOGLE, 2024).

A figura 11 representa as métricas do Core Web Vitals, um conjunto de indicadores desenvolvido pelo Google para medir a experiência do usuário em páginas da web.

Figura 11 – Métricas da Core Web Vitals

Fonte: (GOOGLE, 2024)

Essas métricas analisam três aspectos principais:

LCP (Largest Contentful Paint) – Mede o tempo de carregamento do maior elemento da tela. Valores abaixo de 2,5 segundos são considerados bons, entre 2,5 e 4,0 segundos precisam de melhorias e acima de 4,0 segundos são classificados como ruins.

INP (Interaction to Next Paint) – Avalia a interatividade, medindo o tempo de resposta até a próxima atualização visual após a interação do usuário. Um tempo inferior a 200 ms é considerado ideal, entre 200 ms e 500 ms necessita melhorias, e acima de 500 ms é classificado como ruim.

CLS (Cumulative Layout Shift) – Representa a estabilidade visual, quantificando as mudanças inesperadas de layout que ocorrem durante o carregamento da página. Um valor menor que 0,1 é considerado bom, entre 0,1 e 0,25 necessita melhorias, e acima de 0,25 é classificado como ruim.

2.8 Impacto do SEO na experiência dos usuários

Atualmente a estratégia que o SEO provê se tornou indispensável para as aplicações web, o que garante maior visibilidade nos mecanismos de busca, com isso trazendo mais visitas ao seu website. Os mecanismos de busca utilizam os chamados *web crawlers*, que são rastreadores de conteúdo, analisando diversos fatores como o conteúdo da página, performance, o *Chrome User Experience Report* (CrUX), que é um conjunto de dados públicos que mede como os usuários reais do Google Chrome experimentam a velocidade e estabilidade de um site, e também a acessibilidade. Métricas do *Core Web Vitals* também são extremamente importantes para um melhor SEO (GOOGLE, 2024).

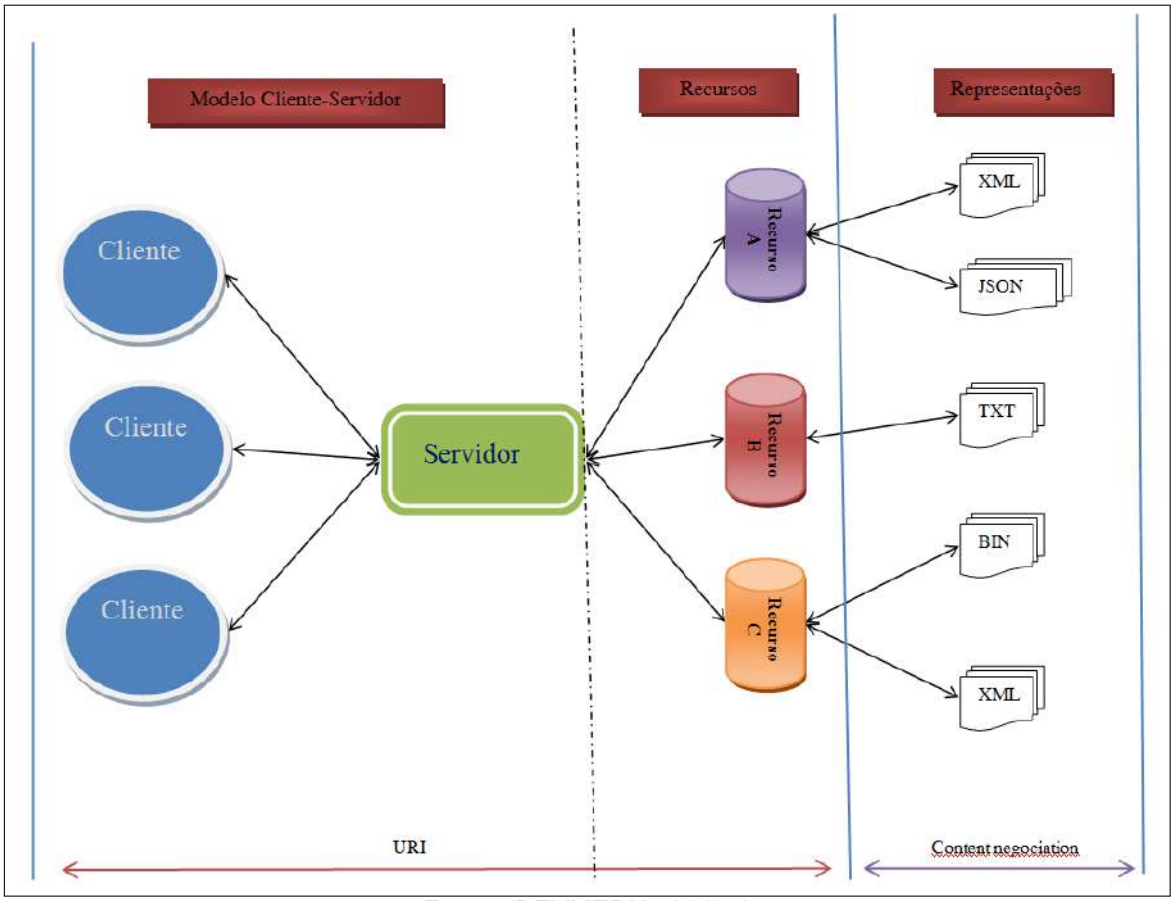
2.9 Padrão Arquitetural: API RESTful

A *API RESTful* (*Representational State Transfer*) consiste em uma *interface* o qual permite a comunicação entre dois ou mais sistemas através do protocolo HTTP. A maioria das aplicações modernas não possuem todos os recursos necessários, por isso precisam se comunicar com outros tipos de serviços. E com isso entra essa arquitetura, suportando uma troca de informações de forma estruturada (AWS, 2025).

O funcionamento dela se baseia em um modelo de cliente-servidor, o qual o cliente realiza requisições HTTP para *endpoints*, que processa a solicitação e retorna uma resposta, normalmente em formato JSON. Cada requisição é independente, ou seja, não tem estado, o servidor não armazena nenhuma informação sobre ela (AWS, 2025).

A figura 12 ilustra o fluxo de comunicação entre cliente e servidor através da *API Restful*.

Figura 12 – Fluxo de comunicação de uma API RESTful



Fonte: (DEVMEDIA, 2013a)

3 Metodologia

Esta seção tem como principal objetivo descrever o fluxo proposto.

Contando com o desenvolvimento de duas aplicações, sendo uma *frontend* e uma *backend*, foram utilizados diferentes *frameworks* e *softwares* que serão descritos abaixo. O desenvolvimento será detalhado junto do fluxo de configuração das aplicações envolvidas na pesquisa, abrangendo desde a preparação do ambiente de *backend* e *frontend* até a análise de desempenho.

Cada etapa será descrita como forma de fornecer uma compreensão clara dos procedimentos adotados.

Para o *backend* foi utilizada a linguagem PHP com o uso do *framework* Laravel para a estruturação do projeto e a configuração de uma aplicação *RESTful*. Além da biblioteca Laravel Filament, a qual foi utilizada para a construção do painel administrativo. A aplicação também conta com o PostgreSQL como gerenciador de banco de dados.

Em relação ao *frontend*, foi utilizado o *framework* Vue.js, que consiste em um *framework* JavaScript, contando com a biblioteca Axios HTTP para integração com a aplicação *backend*. Também foi utilizado o *framework* Quasar, que agiliza o desenvolvimento de aplicações Vue.

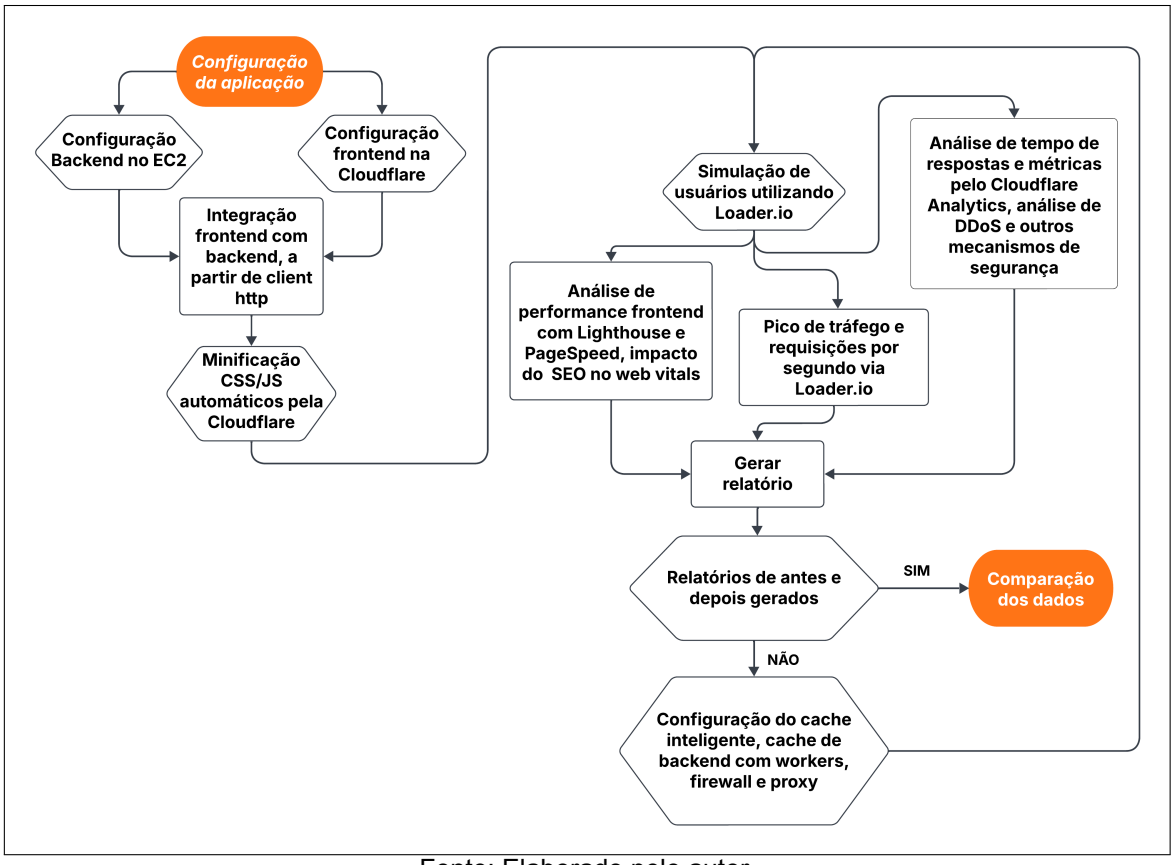
As escolhas das tecnologias mencionadas acima têm respeito à estratégia de desenvolvimento de aplicações web atuais no mercado, sendo escaláveis e de alto desempenho. Esta *stack* oferece benefícios de produtividade e organização.

3.1 Fluxo de desenvolvimento e ferramentas utilizadas

Esta seção possui o objetivo principal em detalhar o fluxo de configuração das aplicações envolvidas na pesquisa, abrangendo desde a preparação do ambiente de *backend* e *frontend* até a análise de desempenho. Cada etapa será descrita como forma de fornecer uma compreensão clara dos procedimentos.

O fluxo descrito nas próximas subseções também está ilustrado no fluxograma representado pela figura 13, em que são detalhadas as etapas do processo.

Figura 13 – Fluxograma de análise da Cloudflare em uma aplicação WEB



Fonte: Elaborado pelo autor.

3.1.1 Configuração do *Backend* no Amazon EC2

Inicialmente, foi criada uma instância no Amazon EC2, com o intuito de hospedar a aplicação *backend*. Foi utilizada inicialmente a instância t2.micro, a qual pode ser usada com a configuração gratuita da AWS. O sistema operacional utilizado foi o Ubuntu 22 LTS, devido à sua compatibilidade e facilidade de configuração.

3.1.2 Configuração do *Frontend* na Cloudflare

A aplicação *frontend* foi configurada na Cloudflare, contando com o serviço de CDN e também com funcionalidades como proteção a ataques DDoS.

A integração entre os dois sistemas foi feita por um cliente HTTP. No *frontend* foram configuradas as chamadas à API do domínio da aplicação em que está a instância EC2.

3.1.3 Configuração de Cache, Workers e Minificação do CSS/JS

Com a proposta de melhorar o desempenho e reduzir a carga da instância EC2, foi implementado o *cache* de conteúdo estático e dinâmico através da Cloudflare. Regras de *cache* foram definidas para armazenar respostas frequentes; além de diminuir a carga, e o tempo de resposta. Além do *cache*, foi utilizado o Cloudflare Worker que permite a manipulação e alteração das requisições na borda da rede.

A fim de otimizar o tempo de carregamento inicial, a Cloudflare já faz automaticamente a minificação dos arquivos CSS e JS.

3.1.4 Configuração de Firewall e Segurança na Cloudflare

Com o propósito de proteger a aplicação contra ameaças comuns na internet, como injeção de SQL, ataque XSS, o WAF da Cloudflare foi configurado. Contando com regras personalizadas para filtrar tráfego malicioso e bloquear qualquer tentativa de explorar vulnerabilidades. Ativando também a proteção automática contra DDoS.

Para proteger a aplicação contra ameaças comuns, como injeção de SQL e *Cross-Site Scripting* (XSS), foi configurado o WAF da Cloudflare com as regras básicas de segurança.

3.1.5 Simulação de Usuários com Loader.io

Como critério de avaliação de desempenho sob diferentes cargas de tráfego, foi utilizada a ferramenta Loader. Com o intuito de configurar testes de acessos simultâneos, possibilitando observar a aplicação em cenários de alto volume e estresse.

3.1.6 Análise de métricas e relatório do antes e depois

A análise foi baseada em três fontes de monitoramento, em que foram analisados os respectivos pontos:

3.1.6.1 Loader

- Simulação do acesso de múltiplos usuários.
- Monitoramento do tempo de resposta baseado na densidade de usuários.
- Comportamento da Cloudflare ao lidar com o excesso de carga.

Técnica a ser utilizada:

- Executar testes de carga, com vários usuários.

- Documentação do tempo de resposta no pico de requisições.
- Comparação dos dados antes e depois da Cloudflare.

3.1.6.2 Cloudflare Web Analytics

- Tempo médio da resposta do servidor.
- Consumo de banda antes e depois do uso do cache.
- Taxa de requisições bloqueadas pelo firewall no teste de estresse.
- Impacto do cache da Cloudflare e da otimização de imagens nas requisições.

Técnicas a serem utilizadas:

- Coleta dos dados antes e depois das otimizações.
- Aplicação do cache, otimização das imagens e segurança.

3.1.6.3 Google PageSpeed e Lighthouse

- Tempo de carregamento inicial.
- Métricas do Web Vitals (LCP, FID, CLS).
- Impacto da Cloudflare no SEO devido à otimização.
- Impacto da latência após o Cache e Images da Cloudflare.

Técnicas a serem utilizadas:

- Comparação das pontuações geradas antes e depois.
- Avaliação do ganho de performance.

3.2 Requisitos do sistema

A coleta de requisitos desempenha um papel essencial no desenvolvimento de *software*, podendo fornecer uma compreensão clara e detalhada do cliente e dos usuários finais, dependendo de como é modelada. Essa etapa é crítica a fim de reduzir ambiguidades e prevenir falhas ao longo do projeto, além de estabelecer também uma base sólida para o *design* do sistema.(PRESSMAN; MAXIM, 2021).

A respeito dos requisitos do sistema, à medida que o sistema evolui, é natural a necessidade de haver mudanças em requisitos funcionais e não funcionais. Com isso o sistema proposto pode passar por alguma alteração à medida que for sendo desenvolvido.

Então para que o sistema possa ser projetado de forma a atender todos os objetivos definidos com base nas necessidades dos usuários, foi realizado o levantamento de requisitos e a modelagem do sistema, os quais são descritos nas próximas subseções.

3.2.1 Requisitos funcionais

- **RF01 - Checkout Customer** - O sistema deve permitir que *customers* façam *checkout*.
- **RF02 - Consultar Produtos** - O sistema deve permitir que usuários consultem produtos disponíveis. O sistema deve permitir filtros e buscas específicas de produtos.
- **RF03 - Gerenciamento de Organizações** - O sistema deve permitir que administradores adicionem, editem e removam organizações.
- **RF04 - Gerenciamento de Tipos de Organizações** - O sistema deve permitir que administradores adicionem, editem e removam tipos de organizações.
- **RF05 - Gerenciamento de Endereços de Organizações** - O sistema deve permitir que administradores adicionem, editem e removam endereços de organizações.
- **RF06 - Gerenciamento de Produtos** - O sistema deve permitir que administradores adicionem, editem e removam produtos.
- **RF07 - Gerenciamento de Categorias de Produtos** - O sistema deve permitir que administradores adicionem, editem e removam categorias de produtos.
- **RF08 - Gerenciamento de Adicionais de Produtos** - O sistema deve permitir que administradores adicionem, editem e removam adicionais de produtos.
- **RF09 - Gerenciamento Histórico de produtos** - O sistema deve permitir que administradores gerenciem histórico de produtos.

3.2.2 Requisitos não funcionais

Os requisitos não funcionais descrevem a qualidade, desempenho e segurança, ou também, são caracterizados como as regras gerais do sistema. Possuem extrema importância quando se diz respeito ao desenvolvimento de *software* (PRESSMAN; MAXIM, 2021). Com isso, os requisitos não funcionais do sistema proposto serão descritos abaixo.

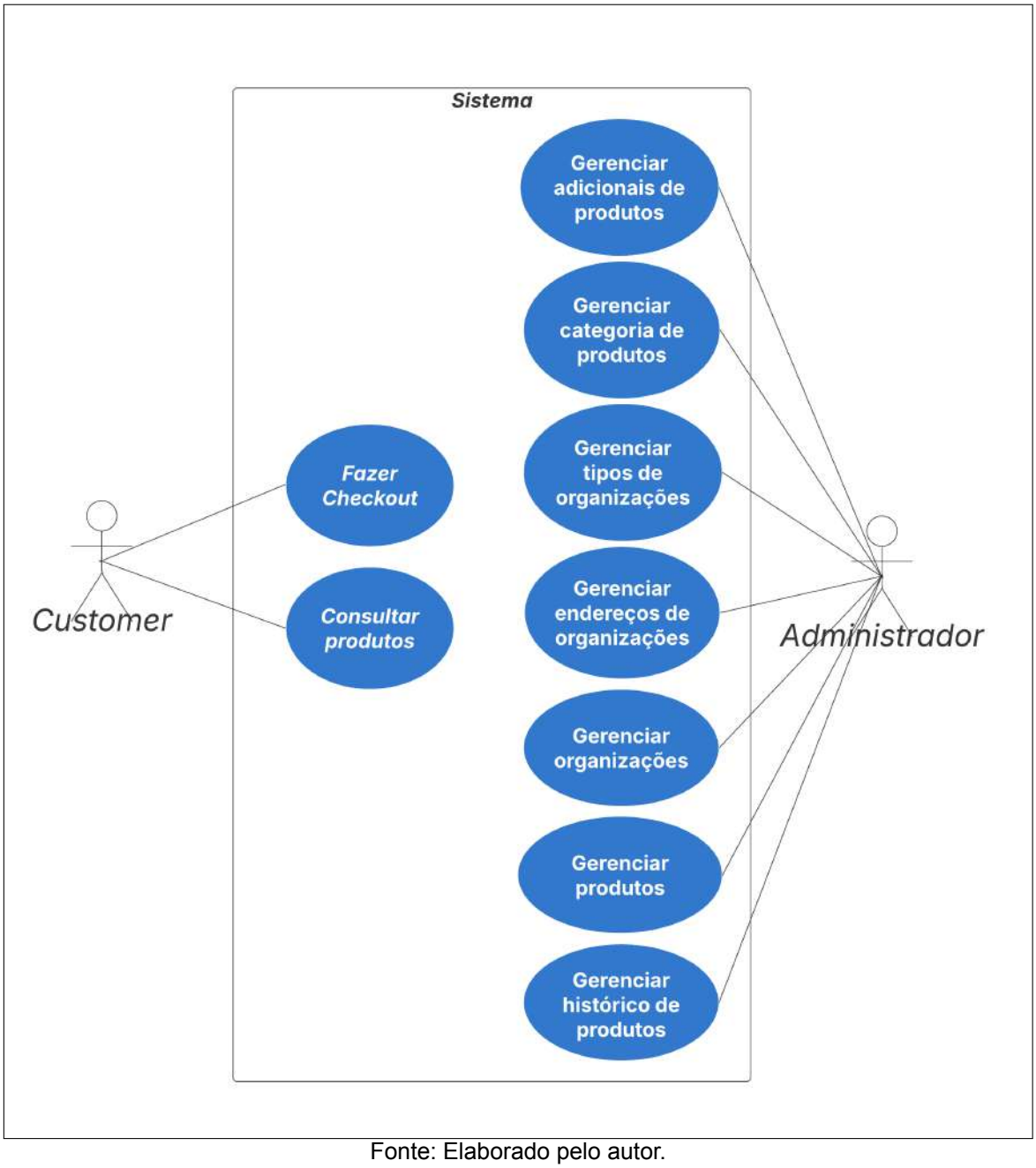
- **RNF01 - Segurança** - O sistema deve proteger contra ataques comuns, como SQL Injection e XSS. O sistema deve implementar autenticação e autorização adequadas para proteger dados sensíveis.
- **RNF02 - Desempenho** - O sistema deve suportar múltiplos usuários simultâneos.
- **RNF03 - Usabilidade** - O sistema deve ter uma *interface* de usuário intuitiva e fácil de usar. O sistema deve fornecer *feedback* claro e útil para as ações do usuário.
- **RNF04 - Manutenibilidade** - O sistema deve ser desenvolvido com boas práticas de codificação para facilitar a manutenção. O sistema deve ter uma documentação clara e completa para desenvolvedores.

3.3 Diagrama de casos de uso

A modelagem de casos de uso descreve como usuários desempenham diferentes papéis quando interagem com o sistema, permitindo que sejam definidas as circunstâncias em que essas interações ocorrem.

Para o sistema em questão, optou-se pelo Diagrama de Casos de Uso, ilustrado pela figura 14, que consiste na modelagem visual do comportamento da interação do usuário com o sistema (IBM, 2007).

Figura 14 – Diagrama de casos de uso: Gerenciamento de cardápios



Fonte: Elaborado pelo autor.

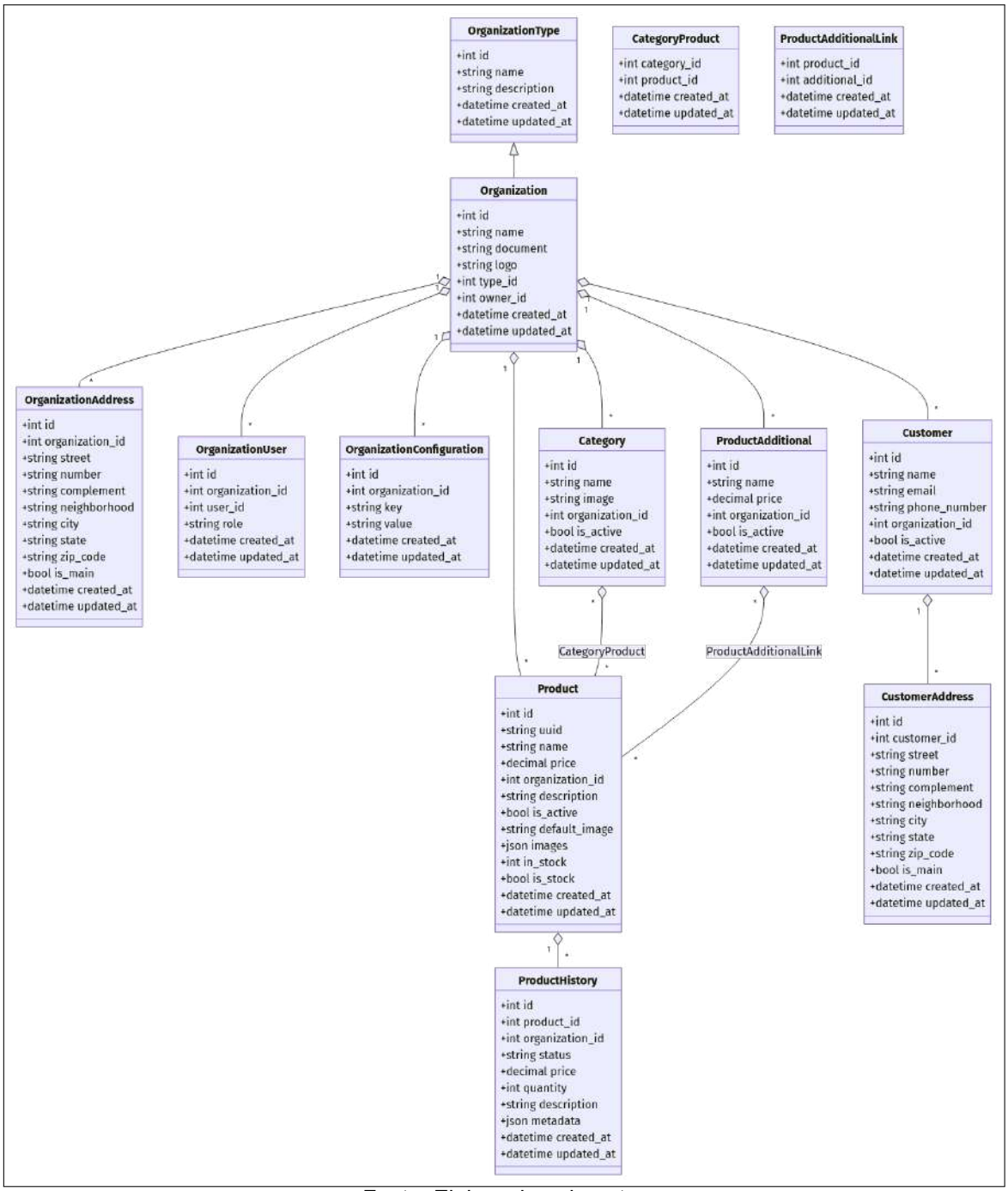
3.4 Modelagem de Domínio

A Modelagem de Domínio por meio do Diagrama de Classes tem como principal objetivo representar um sistema orientado a objetos, definindo suas classes e suas associações. Em termos gerais, uma classe pode ser definida como um tipo de objeto do sistema, enquanto a associação é a relação entre as classes (SOMMERVILLE, 2011).

3.4.1 Diagrama de classe

No contexto do desenvolvimento da aplicação proposta, que segue o paradigma de orientação a objetos, é essencial identificar os objetos que, ao serem implementados, vão garantir a funcionalidade do sistema. Nesse sentido, o modelo adotado, ilustrado pela figura 15, representa o Diagrama de Classes da aplicação, fornecendo uma visão detalhada de sua estrutura.

Figura 15 – Diagrama de classes da aplicação



Fonte: Elaborado pelo autor

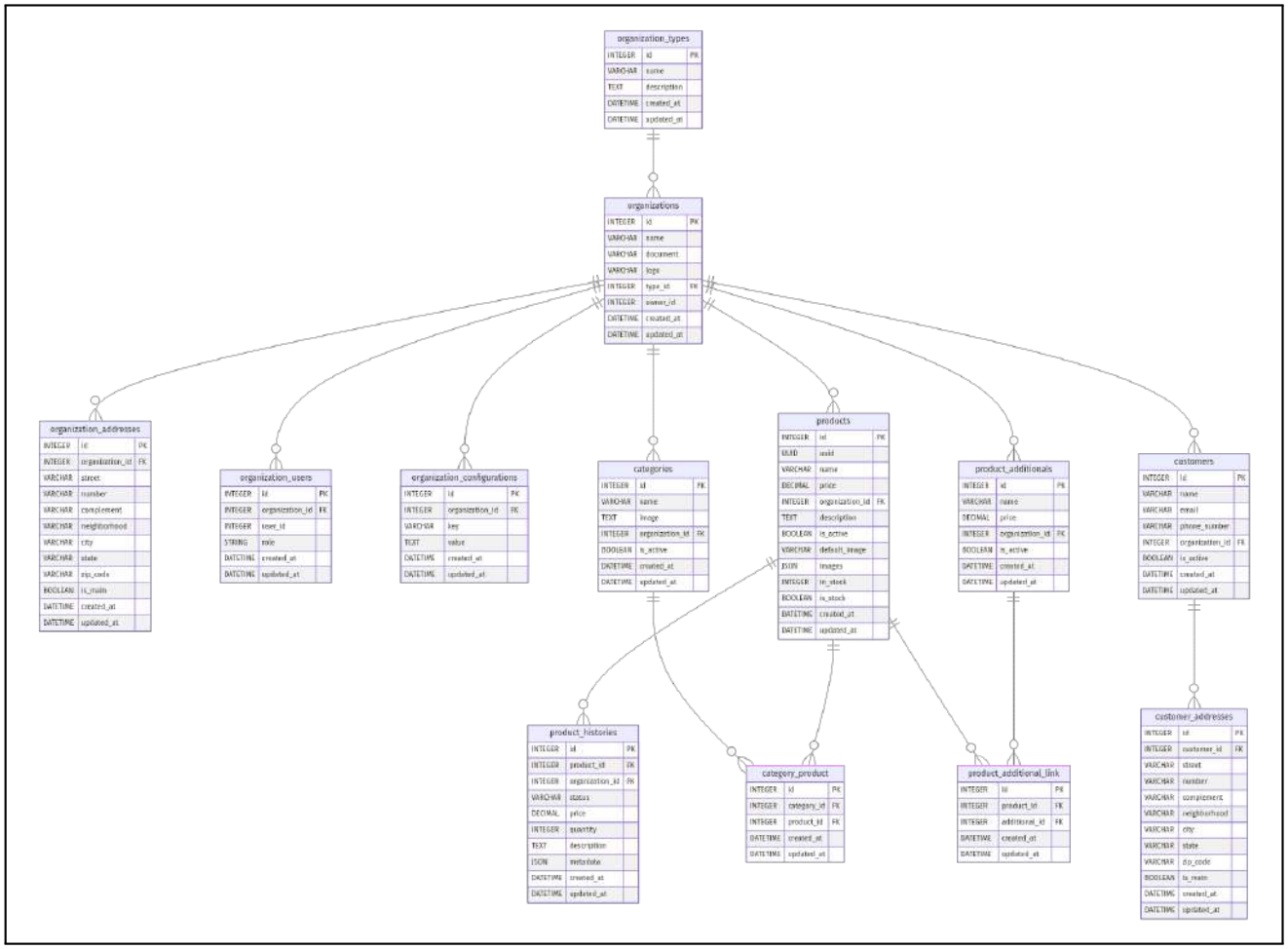
3.5 Modelo de banco de dados

Um modelo de banco de dados representa a estrutura das informações armazenadas, permitindo a organização e manipulação eficiente dos dados. Dependendo especificamente do objetivo da modelagem, o banco pode ser estruturado em diferentes níveis de abstração, variando conforme o nível adotado (HEUSER, 1998).

3.5.1 Modelo Físico

Este modelo possui uma estrutura semelhante ao modelo lógico, porém, inclui alguns detalhes específicos dos tipos internos dos dados. Tendo como principal finalidade garantir compatibilidade direta com o banco (HEUSER, 1998). A figura 16 representa o modelo físico da aplicação.

Figura 16 – Modelo físico do banco de dados



Fonte: Fonte: Elaborado pelo autor

O quadro 4 representa a descrição da tabela Organizações no modelo físico, as demais tabelas serão descritas no Apêndice A.

Quadro 4 – Tabela: organizations

Campo	**Tipo**	**Descrição**	**Restrições**
id	INTEGER	Identificador único da organização	PK
name	VARCHAR(255)	Nome da organização	
document	VARCHAR(20)	Documento da organização (CNPJ/CPF)	Único
logo	VARCHAR(255)	Caminho ou URL do logotipo	Opcional
type_id	INTEGER	Tipo da organização	FK → organization_types.id
owner_id	INTEGER	Usuário proprietário da organização	FK → users.id
created_at	DATETIME	Data de criação do registro	
updated_at	DATETIME	Data de atualização do registro	

Fonte: Elaborado pelo autor.

4 Desenvolvimento do estudo de casos

Nesta unidade será detalhada a ordem e a utilização das ferramentas para o estudo de caso. Para fins de clareza e progressão lógica, esta seção é subdividida em duas partes, sendo elas: a descrição de como as ferramentas foram utilizadas, aplicação inicial em um ambiente sem as configurações da Cloudflare e por fim, para fins de comparação, a aplicação em um ambiente totalmente configurado.

4.1 A aplicação

O sistema desenvolvido para este estudo de caso consiste em um Cardápio Eletrônico simples e intuitivo. Possui funcionalidades essenciais como uma página principal com listagem de produtos, categorização, opções de adicionais, carrinho e checkout. Essa aplicação representa um cenário típico de tráfego variado e por demandar alta disponibilidade, sendo um ambiente ideal para testar a otimização de desempenho e proteção de segurança.

O sistema utilizado no estudo de caso possui as seguintes divisões: o *frontend* e o *backend*. Para o *frontend* foram utilizadas as tecnologias: HTML5, CSS, VUE, Javascript, Typescript, CSS e Quasar. Já para o *Backend* foram utilizadas: PHP, Laravel, Laravel Filament e o Laravel Blade.

4.1.1 *Backend*

O *backend* é responsável pela transferência e gerenciamento de dados da aplicação, tal como a parte de registro de novas empresas no sistema e seus cardápios. O mesmo irá prover dados ao *frontend* de forma que o mesmo não consiga manipular dados, apenas acessá-los.

4.1.1.1 Estrutura do projeto

A aplicação foi desenvolvida utilizando o *framework* Laravel, seguindo o padrão arquitetural MVC (Model-View-Controller). Esse padrão consiste na separação das responsabilidades, sendo a modelagens de dados como Model, a lógica de negócios como Controller e a camada de apresentação das telas do painel administrativo como View (DEVMEDIA, 2013b). Essa estrutura agiliza e simplifica a organização, manutenção e a evolução do sistema.

Além do padrão MVC abordado e utilizado, a aplicação segue os princípios da arquitetura REST, para a construção de suas APIs. Cada parte de transferência de dados da aplicação é representada por uma rota bem definida, que podem ser acessadas por métodos HTTP.

Complementando essa arquitetura, a aplicação está containerizada com Docker, permitindo o isolamento dos ambientes. O uso do Docker garante que a aplicação tenha o mesmo comportamento independente da máquina ou sistema operacional.

Para a criação da interface administrativa do sistema, foi utilizada o Filament, uma biblioteca moderna baseada em Laravel, oferecendo toda uma camada de abstração para a criação dos painéis e gerenciamento de cargos.

O processo de *deploy* da aplicação é automatizado por meio do GitHub Actions, um serviço de integração contínua CI/CD. Através do arquivo ”deploy.yml”, que está localizado no diretório ”.github/workflows”, em que é definido uma *pipeline* que executa etapas como atualização de dependências, *caching* de dependências, envio da imagem do repositório para a instância EC2. Esse fluxo garante o processo de entrega da nova versão, reduzindo a possibilidade de erros manuais.

O Quadro 5 apresenta de forma detalhada a estrutura dos diretórios do *backend* da aplicação, com descrição das responsabilidades de cada pasta dentro da arquitetura do projeto. Essa organização segue as boas práticas recomendadas também pelo proprio *framework* Laravel.

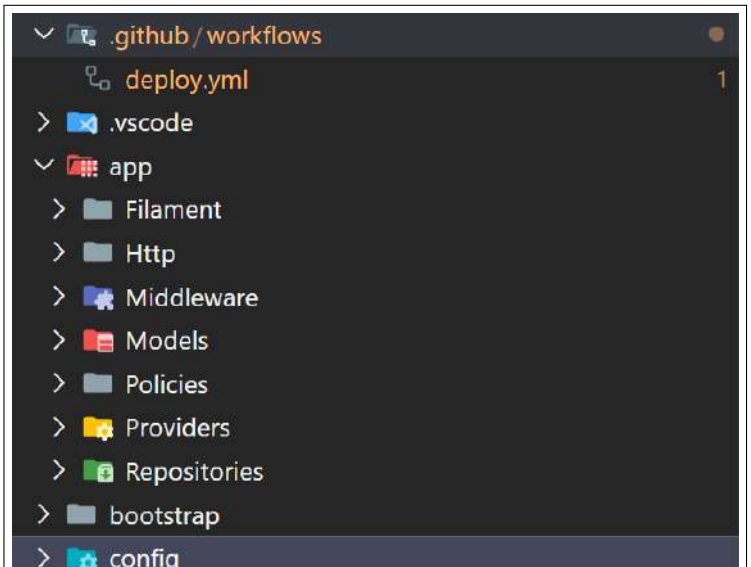
A Figura 17 ilustra a estrutura de arquivos e diretórios.

Quadro 5 – Descrição da estrutura de pastas da aplicação *Backend*

Diretório	**Descrição**
`app/`	Contém a lógica principal da aplicação, incluindo Models, Controllers, Providers, Policies e Repositories.
`app/Filament`	Armazena recursos usados pelo painel administrativo construído com o Filament, como Resources, Pages e Widgets.
`bootstrap/`	Inicializa a aplicação e carrega o autoloader e arquivos de cache.
`config/`	Arquivos de configuração da aplicação, incluindo banco de dados, cache, mail, etc.
`database/`	Contém as migrations, seeders e factories responsáveis pela estrutura e popularização do banco de dados.
`docker/`	Contém arquivos de configuração de *containers* Docker utilizados na infraestrutura do projeto.
`lang/`	Arquivos de tradução para suportar internacionalização da aplicação.
`public/`	Diretório público acessível pela web, onde ficam o arquivo `index.php` e ativos como imagens e scripts.
`resources/`	Contém views (Blade), arquivos de *frontend* e traduções personalizadas.
`routes/`	Define as rotas da aplicação (web, api, console), mapeando endpoints para seus respectivos controllers.
`storage/`	Armazena *logs*, arquivos temporários e uploads de usuários.
`tests/`	Contém os testes automatizados da aplicação, garantindo a qualidade e integridade do sistema.
`.github/workflows/`	Diretório com arquivos de configuração para integração contínua via GitHub Actions.

Fonte: Elaborado pelo autor.

Figura 17 – Estrutura *Backend*



4.1.1.2 Conexão com banco de dados

Para a conexão com o banco de dados, o Laravel possui uma estrutura orientada a convenções, que facilita a configuração, representado pela figura 18, para o banco de dados. Logo, a conexão é gerenciada através do arquivo de configuração ”database.php”, o qual define os parâmetros de acesso para diferentes tipos de banco de dados suportados, sendo eles o MySQL, PostgreSQL, SQLite e SQL Server (LARAVEL, 2024).

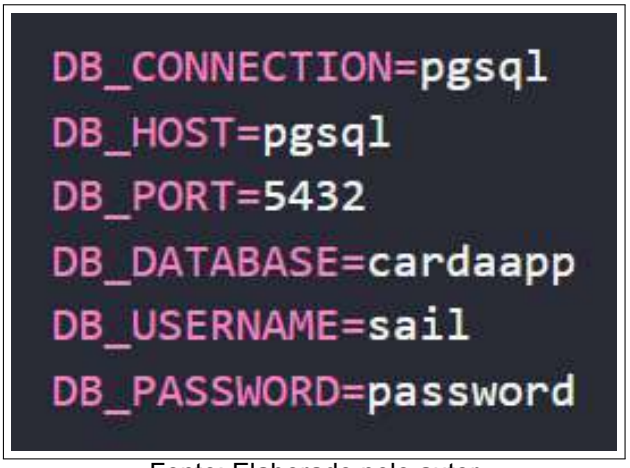
Figura 18 – Arquivo configuração banco de dados

```php
<?php

use Illuminate\Support\Str;

return [

    /*
    |--------------------------------------------------------------------------
    | Default Database Connection Name
    |--------------------------------------------------------------------------
    |
    | Here you may specify which of the database connections below you wish
    | to use as your default connection for database operations. This is
    | the connection which will be utilized unless another connection
    | is explicitly specified when you execute a query / statement.
    |
    */

    'default' => env('DB_CONNECTION', 'sqlite'),

    /*
    |--------------------------------------------------------------------------
    | Database Connections
    |--------------------------------------------------------------------------
    |
    | Below are all of the database connections defined for your application.
    | An example configuration is provided for each database system which
    | is supported by Laravel. You're free to add / remove connections.
    |
    */

    'connections' => [
Ctrl+L to chat, Ctrl+K to generate
        'sqlite' => [
            'driver' => 'sqlite',
            'url' => env('DB_URL'),
            'database' => env('DB_DATABASE', database_path('database.sqlite')),
            'prefix' => '',
            'foreign_key_constraints' => env('DB_FOREIGN_KEYS', true),
        ],

        'mysql' => [
            'driver' => 'mysql',
            'url' => env('DB_URL'),
            'host' => env('DB_HOST', '127.0.0.1'),
            'port' => env('DB_PORT', '3306'),
            'database' => env('DB_DATABASE', 'laravel'),
```

Fonte: Elaborado pelo autor

Esses parâmetros, no entanto, estão definidos com base nas variáveis de ambiente, no arquivo .env, representado pela figura 19, que se encontra na raiz do projeto. Contando com as chaves DB_CONNECTION, DB_HOST, DB_PORT, DB_DATABASE, DB_USERNAME e DB_PASSWORD.

Figura 19 – Env - Chaves banco de dados



Fonte: Elaborado pelo autor

Ao iniciar a aplicação, o Laravel já utiliza os arquivos de configuração e Env para carregar a conexão com o banco dentro do ambiente Docker. Utilizando o Eloquent ORM para mapear os dados para as models, que permite trabalhar com os dados orientados a objetos dentro da aplicação.

4.1.1.3 Painel Administrativo

O painel administrativo foi desenvolvido utilizando o pacote Filament, que proporciona uma estrutura para criação de formulários, tabelas e permissões de forma customizável.

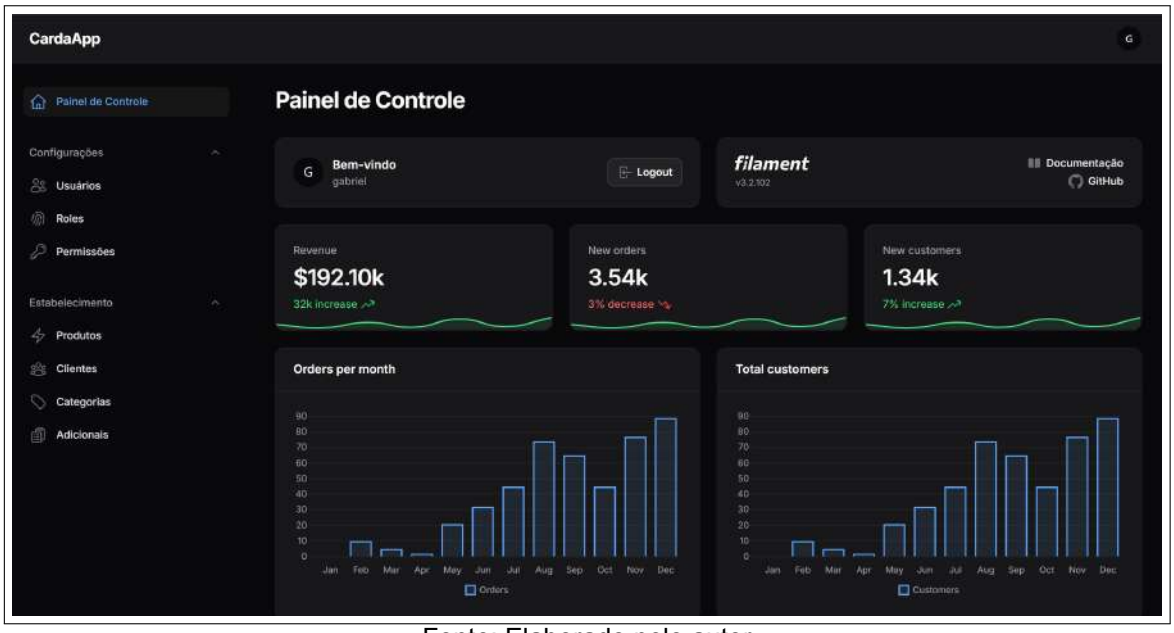
A autenticação única que o sistema apresenta, é baseado no modelo OrganizationUser, que estende a classe Authenticable do Laravel, a qual implementa também a interface FilamentUser. Esse modelo representa os usuários vinculados a organização específica, sendo responsáveis por gerenciar os dados de cada cardápio, sendo produtos, funcionários, customers e outras informações operacionais.

Como mencionado, a autenticação é realizada a partir de sessões, utilizando o próprio middleware guard web do Laravel. Após o login, um ID de sessão é gerado e armazenado no banco de dados, sendo vinculado ao navegador do usuário por meio de um cookie chamado laravel_session. Esse mecanismo permite que o Laravel identifique o usuário autenticado a cada requisição. Também contando com a opção de

”lembrar-me”, que consiste salvar o remember_token, permitindo restaurar a sessão sem precisar logar novamente (LARAVEL, 2024).

O dashboard inicial, representado pela figura 20, demonstra a interface principal do painel administrativo, o qual pode ser acessado pelos usuários autenticados. Essa tela fornece uma visão geral das funcionalidades mais importantes do sistema, exibindo, no menu lateral esquerdo, atalhos para gerenciamento de usuários, roles, permissões, produtos, clientes, categorias e os adicionais dos produtos.

Figura 20 – Tela inicial - Painel Administrativo



Fonte: Elaborado pelo autor

4.1.2 *Frontend*

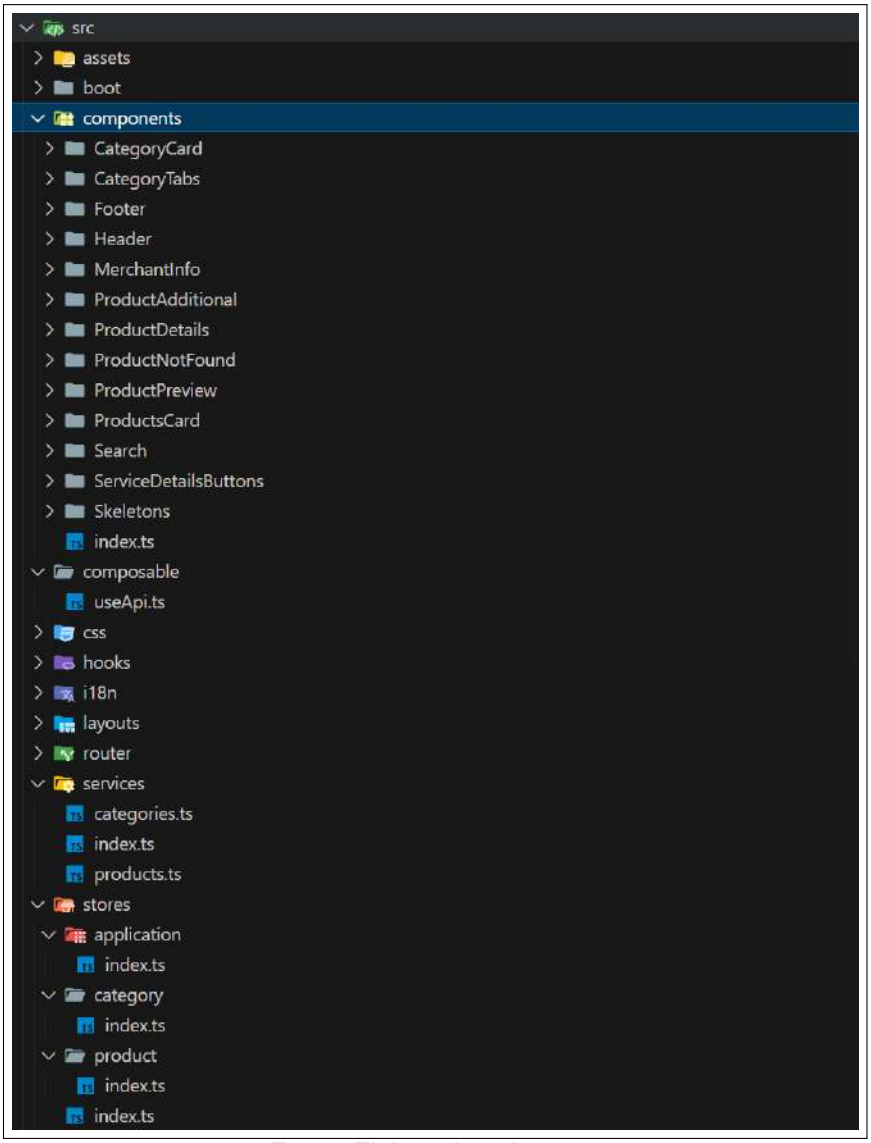
O *frontend* consiste na parte de interface gráfica do projeto utilizado como estudo de caso, em que foram aplicadas as configurações de otimizações da Cloudflare e também do Loader.

Para o desenvolvimento, como citado na seção 4.1, foi utilizado o Vue.js e o *framework* Quasar para componentização.

4.1.2.1 Estrutura do projeto

Na Figura 21 é exibida a estrutura de pastas e arquivos do *frontend*, com o quadro 6 descrevendo os pacotes, arquivos e suas descrições.

Figura 21 – Estrutura *Frontend*



Fonte: Elaborado pelo autor

Quadro 6 – Descrição da estrutura de pastas da aplicação *Frontend*

Diretório	**Descrição**
`src/`	Diretório principal que contém todo o código fonte da aplicação.
`src/assets/`	Armazena recursos estáticos como imagens, fontes e outros arquivos de mídia.
`src/boot/`	Contém arquivos de inicialização que são carregados antes da aplicação, como configurações de plugins.
`src/components/`	Componentes Vue reutilizáveis que podem ser compartilhados em toda a aplicação.
`src/composable/`	Funções composables do Vue 3 que encapsulam lógica reutilizável.
`src/css/`	Arquivos de estilo globais e variáveis CSS/SCSS.
`src/hooks/`	*Hooks* personalizados para reutilização de lógica entre componentes.
`src/i18n/`	Arquivos de internacionalização para suporte a múltiplos idiomas.
`src/layouts/`	Layouts base da aplicação que definem a estrutura comum das páginas.
`src/router/`	Configuração do Vue Router e definição das rotas da aplicação.
`src/services/`	Serviços para comunicação com APIs e lógica de negócios.
`src/stores/`	Gerenciamento de estado global usando Pinia ou Vuex.
`src/utils/`	Funções utilitárias e helpers reutilizáveis.
`src/views/`	Componentes de página/view principais da aplicação.
`public/`	Arquivos estáticos que serão servidos diretamente sem processamento.
`.quasar/`	Arquivos gerados automaticamente pelo *framework* Quasar.
`dist/`	Diretório de build contendo a aplicação compilada para produção.
`.github/workflows/`	Configurações de CI/CD para GitHub Actions.

Fonte: Elaborado pelo autor.

4.1.2.2 Rotas de Navegação

Contando com o gerenciamento e controle das rotas de navegação, o pacote utilizado foi o *Vue-Router*, responsável pelo gerenciamento de *layouts* baseado na estrutura SPA (*Single Page Application*).

Por exemplo, a primeira rota definida na Figura 22 possui *path: "/"*, o qual indica que será acessada na raiz da aplicação. Seu *component* irá carregar o *LandingLayout*, que funciona como o *layout* principal da aplicação.

Figura 22 – Rotas *Frontend*

```
import { useOrganizationStore } from '@stores';
import type { RouteRecordRaw } from 'vue-router';

const routes: RouteRecordRaw[] = [
  {
    path: '/',
    component: () => import('@layouts/LandingLayout.vue'),
    children: [
      {
        path: '',
        component: () => import('@views/LandingPage/index.vue'),
      },
    ],
  },
  {
    path: '/:organization',
    component: () => import('@layouts/MainLayout.vue'),
    beforeEnter: (to, from, next) => {
      const organizationStore = useOrganizationStore();

      if (!organizationStore.selectedOrganization?.name) {
        next('/');
      } else {
        next();
      }
    },
    children: [
      {
        path: '',
        name: 'Home',
        component: () => import('@views/Main/index.vue'),
      },
    ],
  },
  {
    path: '/:catchAll(.*)*',
    name: 'not-found',
    component: () => import('@views/NotFound/index.vue'),
  },
  {
    path: '/checkout',
    name: 'checkout',
    component: () => import('@views/Checkout/Checkout.vue'),
  },
];

export default routes;
```

Fonte: Elaborado pelo autor

O ponto principal é a implementação do *hook* de navegação *beforeEnter* na rota dinâmica */:organization*. Isso garante que uma função seja executada antes de

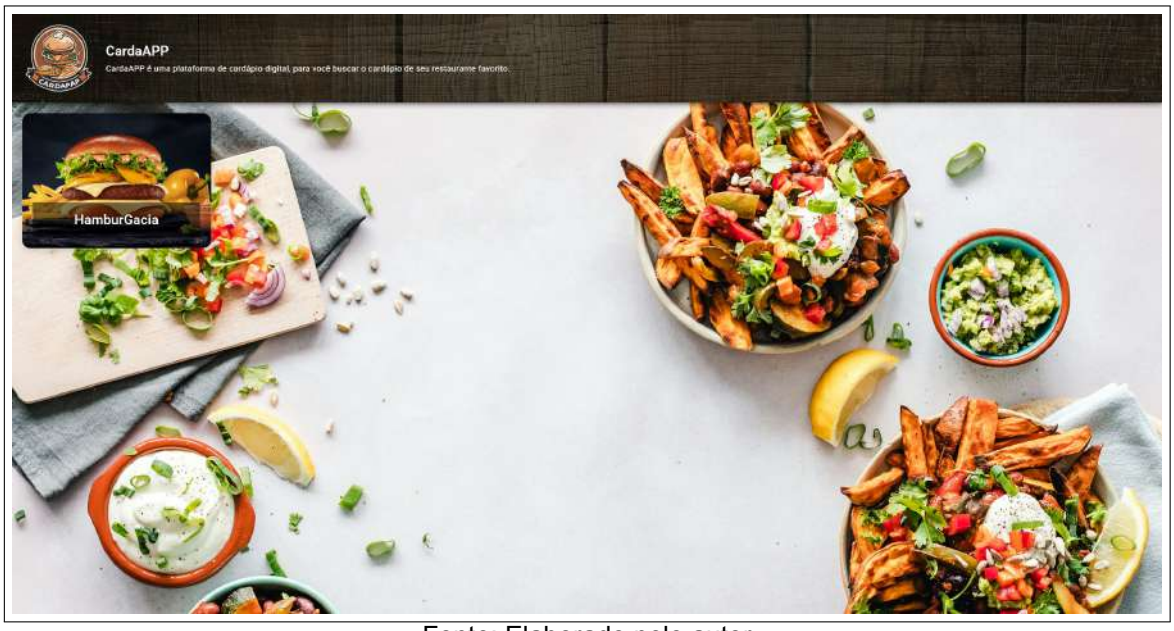
permitir a entrada na rota.

No *vue-router*, cada rota é definida como um objeto no Javascript contendo as propriedades essenciais. A propriedade *path* que define o caminho da URL que será acessado pelo usuário; *component* especifica qual componente o Vue será renderizado quando essa URL for acessada, e a propriedade *children* permite a definição de rotas aninhadas.

4.1.2.3 *Interfaces*

Ao acessar a aplicação, é possível verificar toda a listagem de empresas que possuem cardápios registrados, ilustrado pela figura 23.

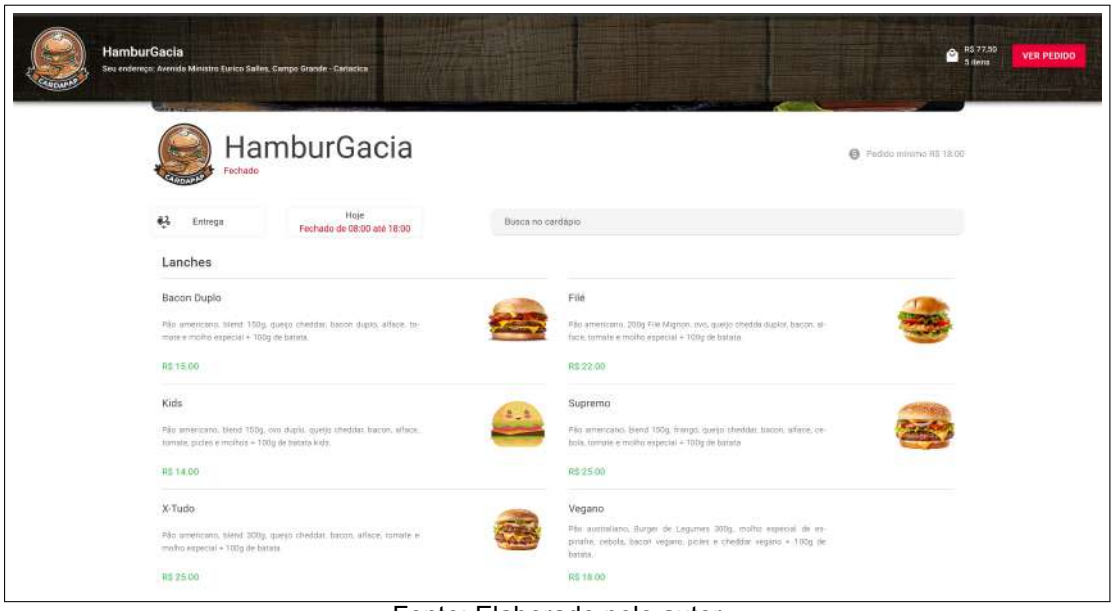
Figura 23 – Landing Page



Fonte: Elaborado pelo autor

Ao selecionar um cardápio, o usuário é redirecionado para a tela, ilustrada pela figura 24, que consiste na listagem de categorias, produtos e adicionais dos produtos presentes no cardápio.

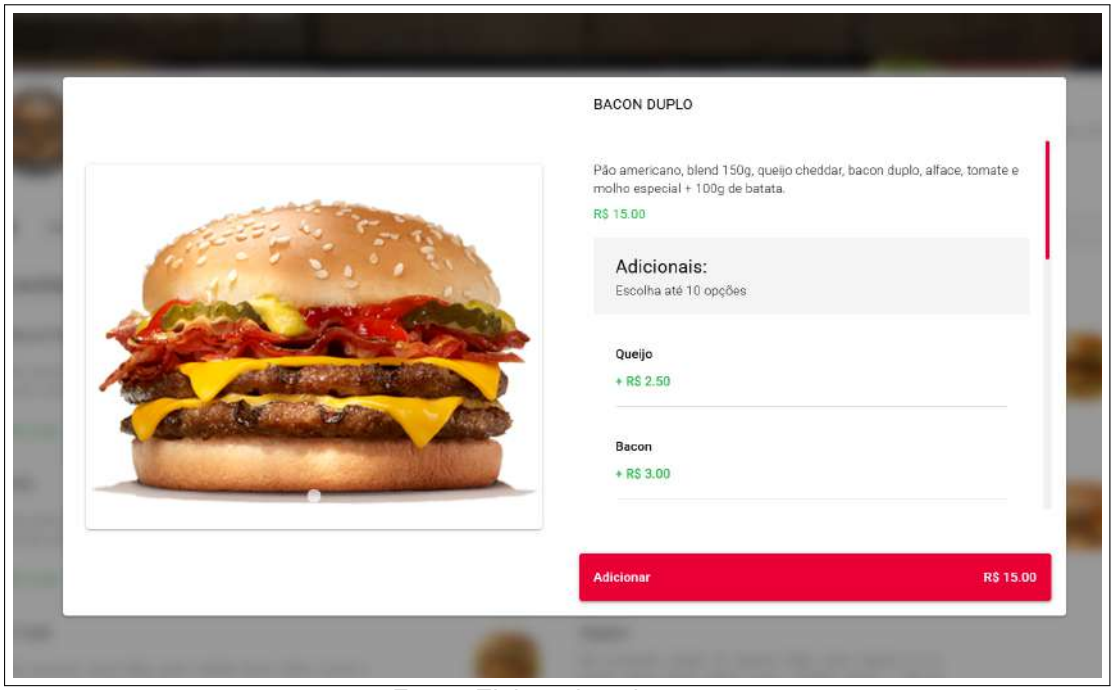
Figura 24 – Cardapio



Fonte: Elaborado pelo autor

Ao clicar em qualquer produto, é aberto um modal, representado pela figura 25, que consiste na escolha de todos adicionais do produto, dentro de sua categoria. Contando com o botão de adicionar ao carrinho.

Figura 25 – Modal



Fonte: Elaborado pelo autor

A página também conta com as informações do estabelecimento, como tipo de entrega, horário de funcionamento, e também o pedido mínimo, representado pela

figura 26.

Figura 26 – Informações Estabelecimento

Fonte: Elaborado pelo autor

Ao tentar buscar um produto no cardápio irá abrir um modal, representado pela figura 27, o qual pode filtrar todas as opções presentes.

Figura 27 – Pesquisa Cardápio

Fonte: Elaborado pelo autor

Todos produtos selecionados, contando também com seus adicionais, são adicionados ao estado do carrinho no Pinia, que é o gerenciador de estado utilizado pela aplicação. O carrinho foi representado pela figura 28.

Figura 28 – Carrinho

Fonte: Elaborado pelo autor

Contando com a soma dos adicionais e do preço dos produtos, a tela de Checkout, ilustrada pela figura 29, representa o preço total do pedido, podendo ser enviado ao estabelecimento em questão.

Figura 29 – Checkout

Fonte: Elaborado pelo autor

4.2 Configuração das Aplicações

Nessa seção foi apresentado o passo a passo de configuração e *deploy* das aplicações, tanto *backend* como *frontend*.

4.2.1 Configuração *Backend* na EC2

O projeto *backend* está versionado no GitHub, contando com uma *pipeline* CI/CD para *deploy* automático em uma instância EC2, através de um ambiente dockerizado. Todas etapas serão descritas nas próximas subseções.

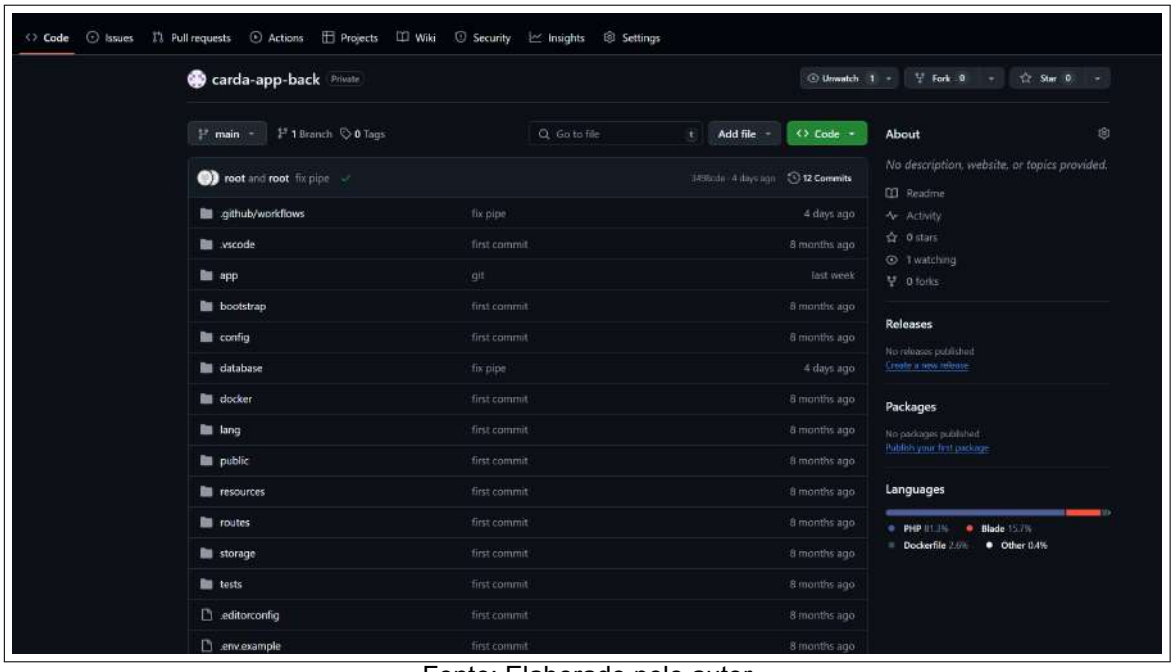
4.2.1.1 Versionamento com GitHub

O versionamento do código da aplicação foi feita através da plataforma GitHub, que oferece um sistema distribuído de controle de versões. Essa ferramenta permitiu gerenciar, de forma eficiente o histórico de alterações do projeto.

A utilização do mesmo proporcionou controle sobre *branches*, organização das funcionalidades em diferentes *pull requests*, e também integração com a ferramenta de automação, como o GitHub Actions.

A Figura 30 ilustra a interface principal do repositório utilizado no projeto.

Figura 30 – Interface do GitHub utilizada no versionamento



Fonte: Elaborado pelo autor

4.2.1.2 GitHub Runner

O GitHub Runner é o agente responsável por executar os *workflows*, definidos no Github Actions (GITHUB, 2024).

No contexto do projeto, foi configurado um *Runner Self-Hosted*, ou seja, rodando na própria instância da Amazon EC2.

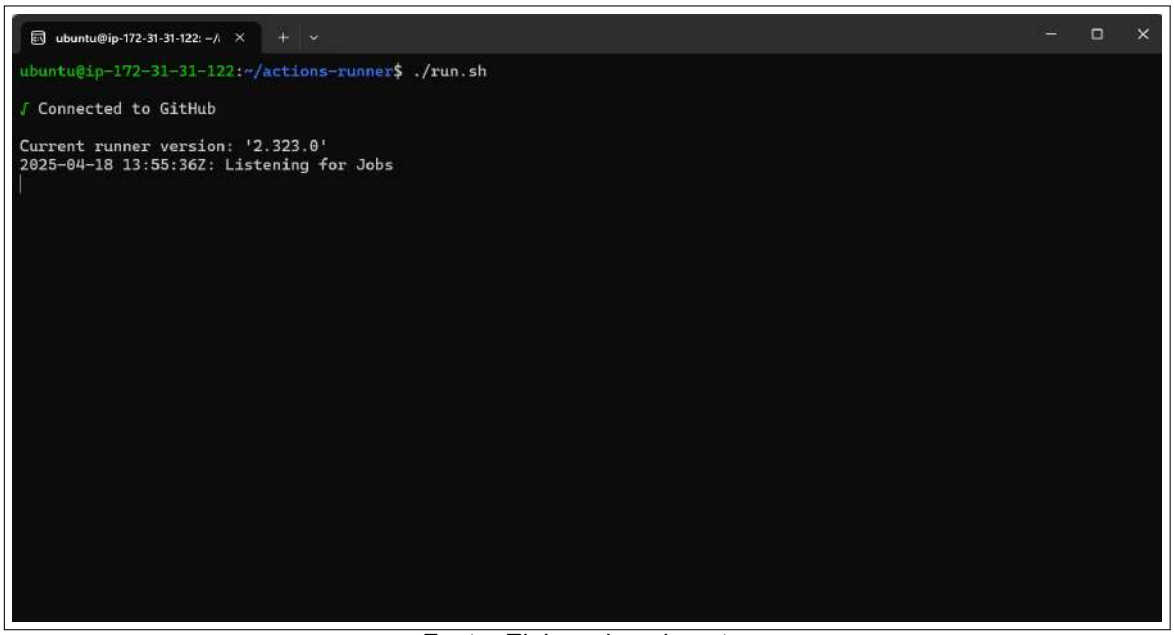
Configurado o Runner no Github, o qual pode ser visto na Figura 31, permite que todas alterações feitas ao repositório via GitHub PUSH dispare automaticamente a execução da *pipeline* CI/CD definidos.

Figura 31 – Runner no GitHub

Fonte: Elaborado pelo autor

Diante disso, para realizar a execução do Runner, primeiramente é necessário executar o *script* do mesmo dentro da Amazon EC2, o qual é representado pela Figura 32, após essa etapa, todos os *workflows* serão executados nessa instância.

Figura 32 – GitHub Runner na EC2



Fonte: Elaborado pelo autor

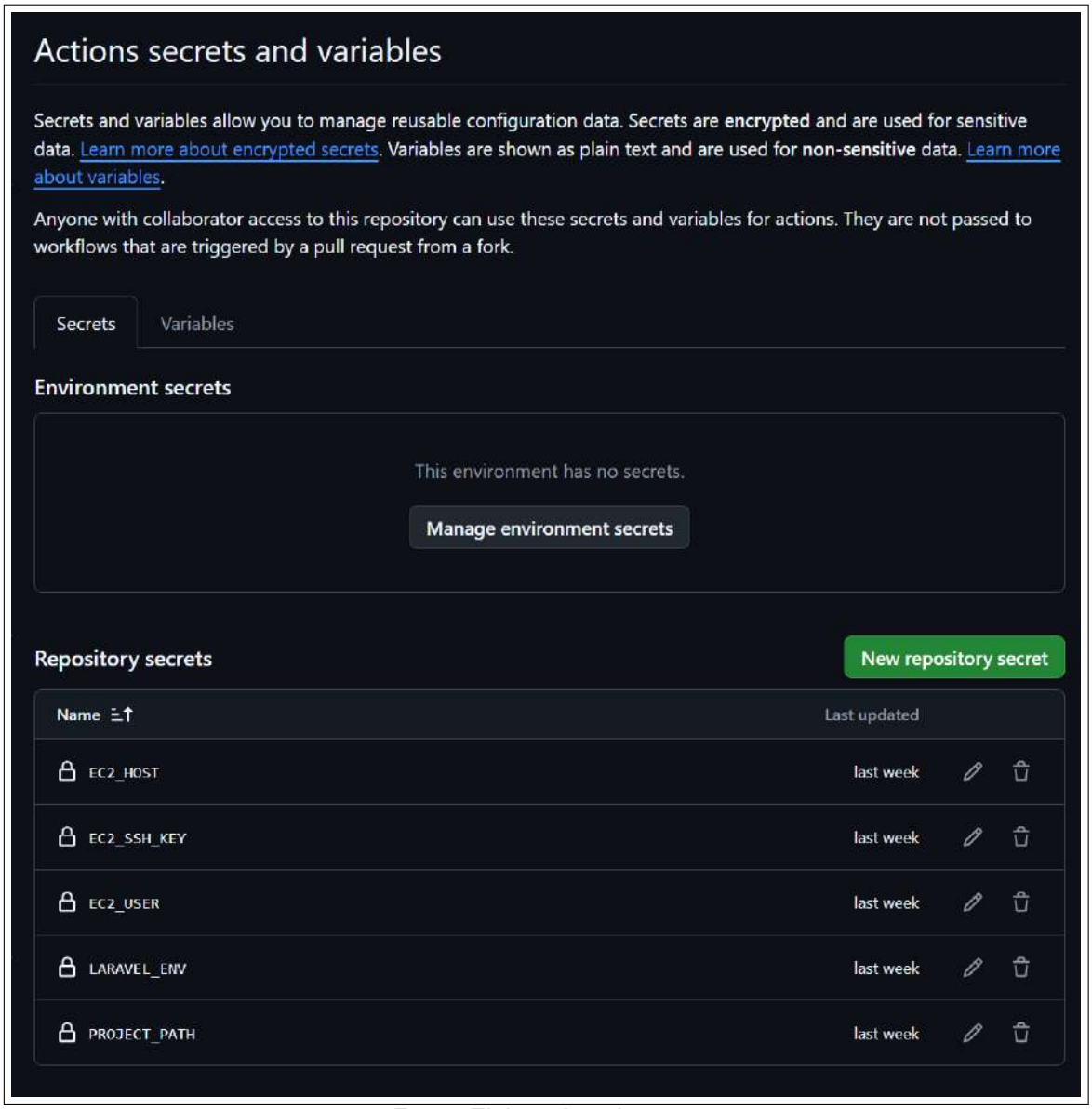
4.2.1.3 GitHub Variables

As Github Variables são variáveis definidas no próprio repositório ou na organização, os quais permitem armazenar informações que serão utilizadas em arquivos de *pipeline* (GITHUB, 2024).

Para configuração da *pipeline* proposta, foram utilizadas variáveis presentes no repositório para configuração do arquivo CI/CD. As mesmas são representadas pela figura 33. Sendo de principal uso o LARAVEL_ENV, para configuração das variáveis e *secrets* do projeto, como senha do banco de dados, *host*, porta da aplicação, entre outros.

Além de proporcionar uma maior segurança e organização, as GitHub Variables evitam que chaves secretas sejam expostas ao público. Ao utilizar essas variáveis definidas no repositório, é possível centralizar os dados como *tokens* de autenticação, credenciais de banco de dados, entre outras chaves.

Figura 33 – GitHub Variables



Fonte: Elaborado pelo autor

4.2.1.4 Criação Instância EC2 na AWS

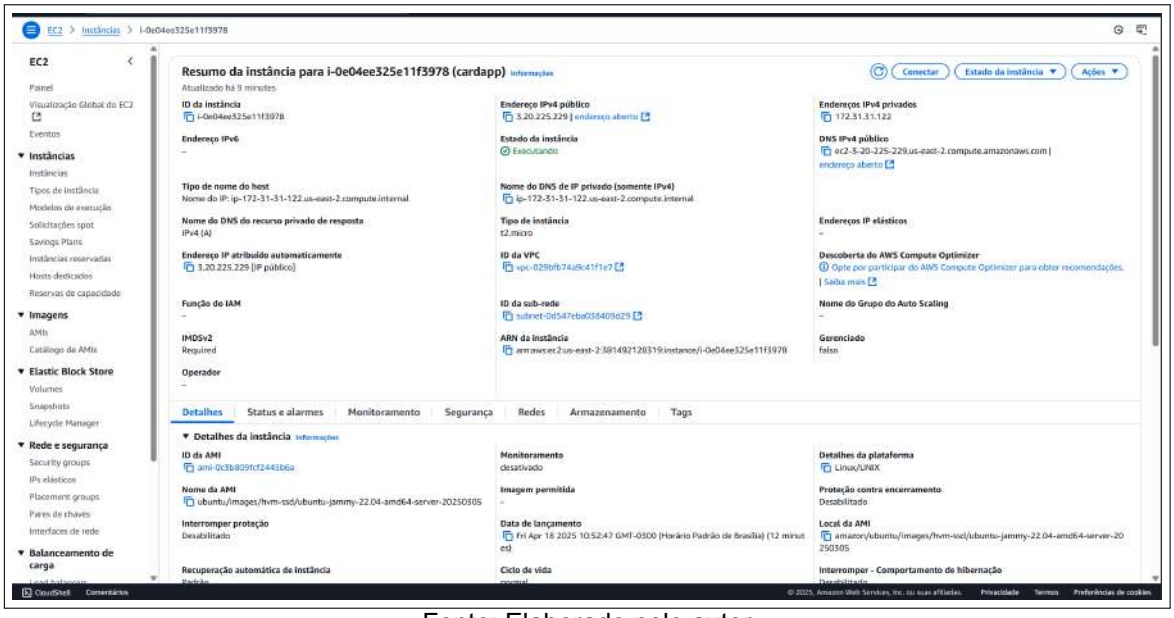
Para hospedar o ambiente de produção do *backend*, foi criado uma instância na Amazon EC2, utilizando o plano gratuito. A configuração escolhida foi o tipo t2.micro, o que consiste em 1 vCPU e 1GB de memória RAM, suficiente para aplicações pequenas, o que é o caso dessa.

A máquina virtual utiliza a imagem do sistema operacional Ubuntu 22.04 LTS, proporcionando o ambiente estável e seguro, o mesmo possui apenas a dependência do docker, sendo também acessado por SSH.

A Figura 34 ilustra a instância criada no console de gerenciamento da AWS. O

qual confirma a escolha do tipo t2.micro e a utilização do Ubuntu. Além disso, durante a configuração da instância, foram também definidos aspectos essenciais que garantem a segurança e a eficiência do ambiente, como a criação do par de chaves SSH e bloqueio para uso apenas pelo IP do autor.

Figura 34 – Instância EC2 criada na AWS com plano gratuito



Fonte: Elaborado pelo autor

4.2.1.5 Integração e Entrega Contínua com GitHub Actions

Para realizar a automação do processo de integração e entrega continua, foi utilizado a ferramenta mencionada, o GitHub Actions, o qual permite a definição de fluxos de trabalho, contando com a instalação de dependências, compilação e *deploy*.

O arquivo ”deploy.yml”, localizado no diretório ”.github/workflows”, define os estágios dessa automação. O mesmo pode ser visto na figura 35. Possuindo apenas um *JOB* com *steps*, que consistem em: transferir o projeto para o Amazon EC2, configurar e instalar o node, gerar o arquivo ”.env”, através das variáveis do GitHub, atualizar as dependências do projeto, e por fim rodar os *containers* no docker dentro da EC2.

Figura 35 – *Deploy* YML

```
.github > workflows > deploy.yml
root, 4 days ago | 1 author (root)
name: Deploy

on:
  push:
    branches:
      - main

jobs:
  deploy:
    name: Deploy
    runs-on: self-hosted

    steps:
      - name: Checkout Code
        uses: actions/checkout@v3

      - name: Setup Node
        uses: actions/setup-node@v3
        with:
          node-version: 18

      - name: Setup PHP
        uses: shivammathur/setup-php@v2
        with:
          php-version: 8.3
          tools: composer:v2
          coverage: none

      - name: Generate .env file
        run: echo '${{ secrets.LARAVEL_ENV }}' > .env

      - name: Get Composer Cache Directory
        id: composer-cache
        run: |
          echo "dir=$(composer config cache-files-dir)" >> $GITHUB_OUTPUT

      - uses: actions/cache@v4
        with:
          path: ${{ steps.composer-cache.outputs.dir }}
          key: ${{ runner.os }}-composer-${{ hashFiles('**/composer.lock') }}

      - name: Install PHP dependencies
        run: composer install --optimize-autoloader      root, 4 days ago • fix pipe …

      - name: Restart containers with Docker Compose
        run: |
          ./vendor/bin/sail down
          ./vendor/bin/sail up -d --build
```

Fonte: Elaborado pelo autor

4.2.1.6 DockerFile

O ”Dockerfile”é um arquivo de instruções utilizado pelo Docker para construir imagens a partir de um sistema operacional. Ele define todos os comandos necessários para preparar o ambiente da aplicação, como instalar as dependências, cópia dos arquivos do projeto, entre outros (Docker Inc., 2024).

No contexto do projeto, o DockerFile é o responsável por montar a imagem da aplicação em Laravel com as dependências necessárias do PHP, representado pela figura 36, permite que a instância da Amazon EC2 não precise de nenhuma dependência a não ser a do Docker.

Figura 36 – Arquivo Dockerfile

```
docker > 8.3 > Dockerfile
    matheus-junglobet, 8 months ago | 1 author (matheus-junglobet)
1   FROM ubuntu:22.04
2
3   LABEL maintainer="Taylor Otwell"
4
5   ARG WWWGROUP
6   ARG NODE_VERSION=18
7   ARG POSTGRES_VERSION=15
8
9   WORKDIR /var/www/html
10
11  ENV DEBIAN_FRONTEND noninteractive
12  ENV TZ=UTC
13
14  RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
15
16  RUN apt-get update \
17      && mkdir -p /etc/apt/keyrings \
18      && apt-get install -y gnupg gosu curl ca-certificates zip unzip git supervisor sqlite3 libcap2-bin libpng-dev python2 dnsutils librsvg2-bin fswatch \
19      && curl -sS "https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x14aa40ec0831756756d7f66c4f4ea0aae5267a6c" | gpg --dearmor | tee /etc/apt/keyrings/ppa_ondrej_php.gpg > /dev/null \
20      && echo "deb [signed-by=/etc/apt/keyrings/ppa_ondrej_php.gpg] https://ppa.launchpadcontent.net/ondrej/php/ubuntu jammy main" > /etc/apt/sources.list.d/ppa_ondrej_php.list \
21      && apt-get update \
22      && apt-get install -y php8.3-cli php8.3-dev \
23      php8.3-pgsql php8.3-sqlite3 php8.3-gd php8.3-imagick \
24      php8.3-curl \
25      php8.3-imap php8.3-mysql php8.3-mbstring \
26      php8.3-xml php8.3-zip php8.3-bcmath php8.3-soap \
27      php8.3-intl php8.3-readline \
28      php8.3-ldap php8.3-gmp \
29      php8.3-msgpack php8.3-igbinary php8.3-redis php8.3-swoole \
30      php8.3-memcached php8.3-pcov php8.3-xdebug \
31      && curl -sLS https://getcomposer.org/installer | php -- --install-dir=/usr/bin/ --filename=composer \
32      && curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg \
33      && echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_$NODE_VERSION.x nodistro main" > /etc/apt/sources.list.d/nodesource.list \
34      && apt-get update \
35      && apt-get install -y nodejs \
36      && npm install -g npm \
37      && npm install -g pnpm \
38      && npm install -g bun \
39      && curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | gpg --dearmor | tee /etc/apt/keyrings/yarn.gpg >/dev/null \
40      && echo "deb [signed-by=/etc/apt/keyrings/yarn.gpg] https://dl.yarnpkg.com/debian/ stable main" > /etc/apt/sources.list.d/yarn.list \
41      && curl -sS https://www.postgresql.org/media/keys/ACCC4CF8.asc | gpg --dearmor | tee /etc/apt/keyrings/pgdg.gpg >/dev/null \
42      && echo "deb [signed-by=/etc/apt/keyrings/pgdg.gpg] http://apt.postgresql.org/pub/repos/apt jammy-pgdg main" > /etc/apt/sources.list.d/pgdg.list \
43      && apt-get update \
44      && apt-get install -y yarn \
45      && apt-get install -y mysql-client \
46      && apt-get install -y postgresql-client-$POSTGRES_VERSION \
47      && apt-get -y autoremove \
48      && apt-get clean \
49      && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
50
51  RUN setcap "cap_net_bind_service=+ep" /usr/bin/php8.3
52
53  RUN groupadd --force -g $WWWGROUP sail
54  RUN useradd -ms /bin/bash --no-user-group -g $WWWGROUP -u 1337 sail
55
56  COPY start-container /usr/local/bin/start-container
57  COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
58  COPY php.ini /etc/php/8.3/cli/conf.d/99-sail.ini
59  RUN chmod +x /usr/local/bin/start-container
60
61  EXPOSE 8000
62
63  ENTRYPOINT ["start-container"]
```

Fonte: Elaborado pelo autor

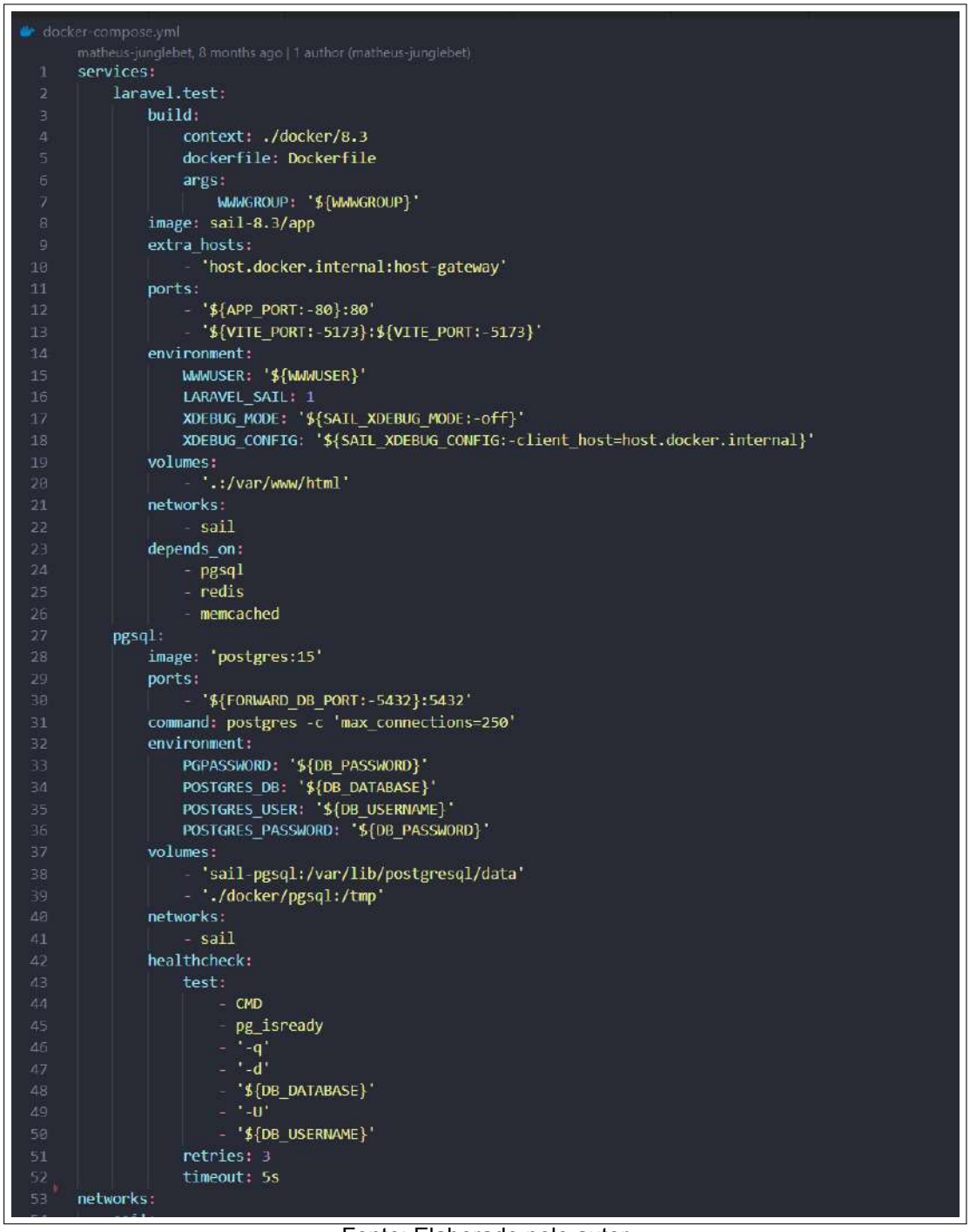
4.2.1.7 Docker Compose

O Docker Compose é uma ferramenta que permite definir e gerenciar vários containers Dockers por meio de um único arquivo de configuração yml, no caso do

projeto, nomeado como ”docker-compose.yml”. O mesmo descreve os serviços que serão utilizados, como também a rede docker.

No contexto do projeto, foi utilizado para orquestrar o *backend* com seu Dockerfile e outras imagens, como o Postgresql, podendo ser visto na figura 37. Facilitando a inicialização e manutenção do ambiente, e permitindo que a instância EC2 execute toda a aplicação sem precisar configurar dependências da mesma.

Figura 37 – Arquivo docker-compose.yml



docker-compose.yml

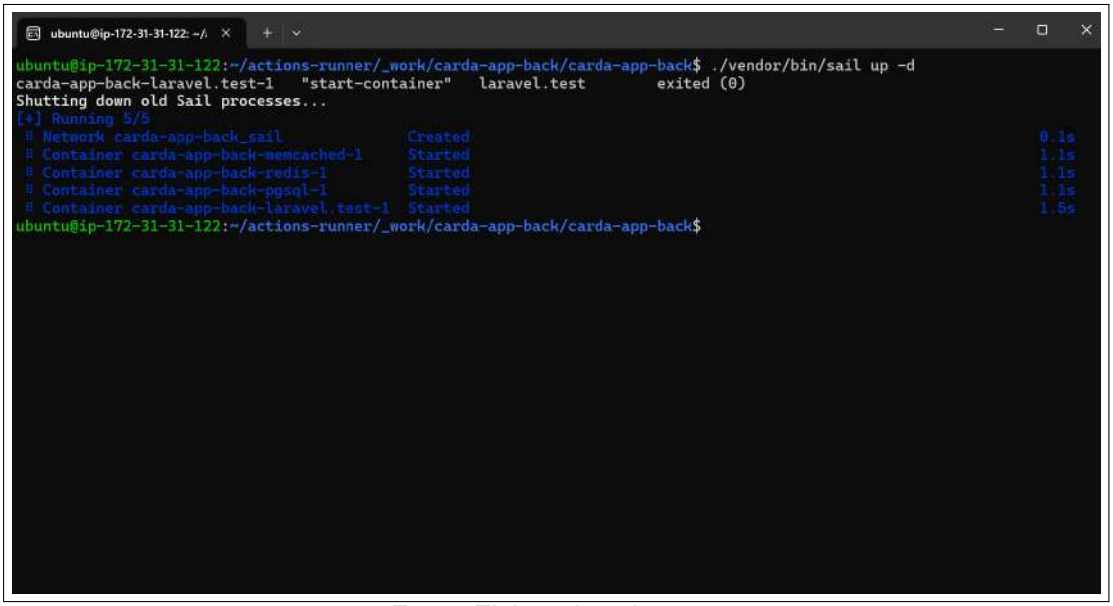
```
matheus-junglebet, 8 months ago | 1 author (matheus-junglebet)
services:
    laravel.test:
        build:
            context: ./docker/8.3
            dockerfile: Dockerfile
            args:
                WWWGROUP: '${WWWGROUP}'
        image: sail-8.3/app
        extra_hosts:
            - 'host.docker.internal:host-gateway'
        ports:
            - '${APP_PORT:-80}:80'
            - '${VITE_PORT:-5173}:${VITE_PORT:-5173}'
        environment:
            WWWUSER: '${WWWUSER}'
            LARAVEL_SAIL: 1
            XDEBUG_MODE: '${SAIL_XDEBUG_MODE:-off}'
            XDEBUG_CONFIG: '${SAIL_XDEBUG_CONFIG:-client_host=host.docker.internal}'
        volumes:
            - '.:/var/www/html'
        networks:
            - sail
        depends_on:
            - pgsql
            - redis
            - memcached
    pgsql:
        image: 'postgres:15'
        ports:
            - '${FORWARD_DB_PORT:-5432}:5432'
        command: postgres -c 'max_connections=250'
        environment:
            PGPASSWORD: '${DB_PASSWORD}'
            POSTGRES_DB: '${DB_DATABASE}'
            POSTGRES_USER: '${DB_USERNAME}'
            POSTGRES_PASSWORD: '${DB_PASSWORD}'
        volumes:
            - 'sail-pgsql:/var/lib/postgresql/data'
            - './docker/pgsql:/tmp'
        networks:
            - sail
        healthcheck:
            test:
                - CMD
                - pg_isready
                - '-q'
                - '-d'
                - '${DB_DATABASE}'
                - '-U'
                - '${DB_USERNAME}'
            retries: 3
            timeout: 5s
networks:
```

Fonte: Elaborado pelo autor

4.2.1.8 Execução do *Backend* na EC2

Executando o comando ”sail up -d”, a aplicação encontra-se no ar e executando na porta 8000. A qual também foi configurada no grupo de segurança da instância para manter essa porta aberta e pública, representado pela figura 38.

Figura 38 – Execução dos containers Docker na instância EC2



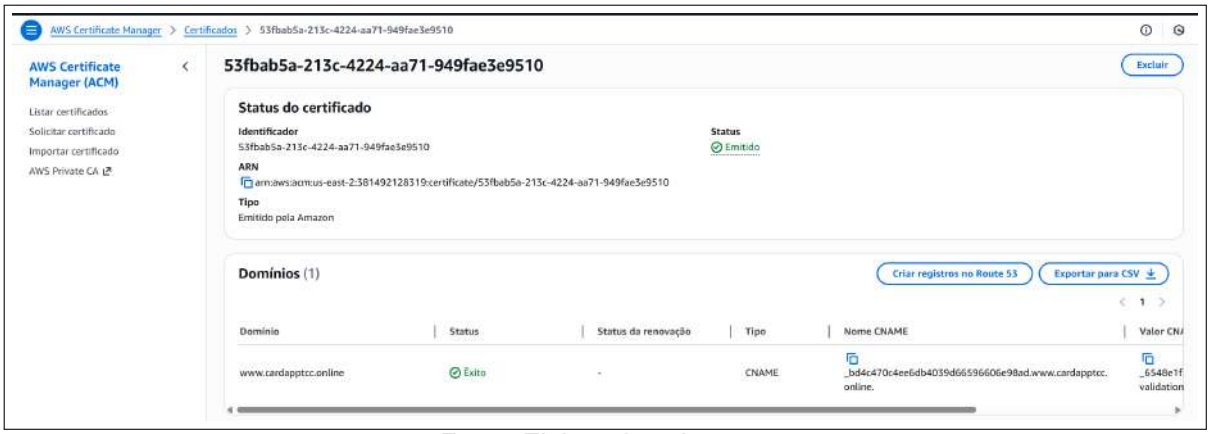
Fonte: Elaborado pelo autor

4.2.1.9 Configuração do AWS Certificate Manager

Como a Cloudflare possui por padrão os certificados SSL/TLS com o https habilitado, o *backend* na EC2 tem necessidade também de possuir esse certificado, para que os dois se comuniquem sem qualquer problema.

Com intuito de utilizar esse SSL/TLS, foi adquirido um domínio cardapptcc.online, possibilitando utilizar o mesmo no ACM (AWS Certificate Manager), o que permite a emissão do certificado para o *backend*. A figura 39 representa a tela de configuração desse certificado, onde é necessário adicionar o CNAME especificado a fim de validar o seu domínio.

Figura 39 – Configuração ACM

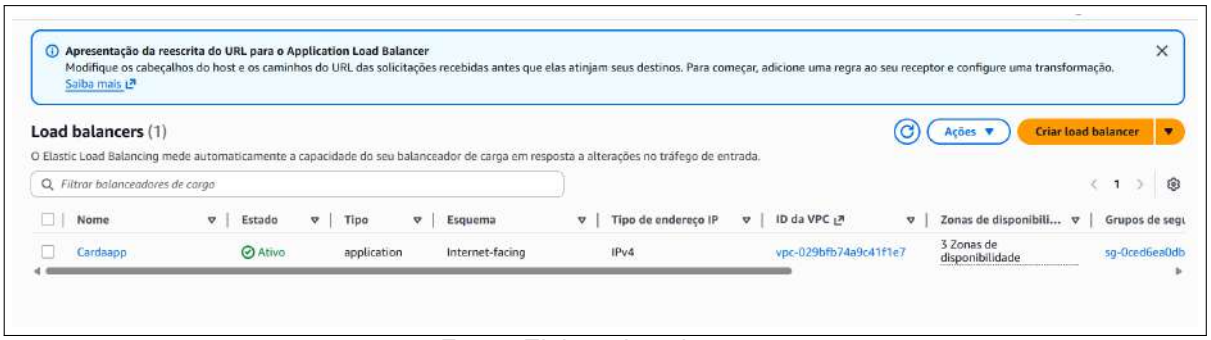


Fonte: Elaborado pelo autor

4.2.1.10 Configuração do Load Balancer

Com a configuração do ACM, a forma de se aplicar foi por meio do Load Balancer da AWS. O mesmo consiste em balancear a carga entre as requisições dos clientes, porém, para o intuito da pesquisa, foi utilizado apenas como forma de aplicar o certificado. A configuração do mesmo pode ser vista na figura 40, que consiste em selecionar sua instância EC2 e o seu certificado.

Figura 40 – Load Balancer



Fonte: Elaborado pelo autor

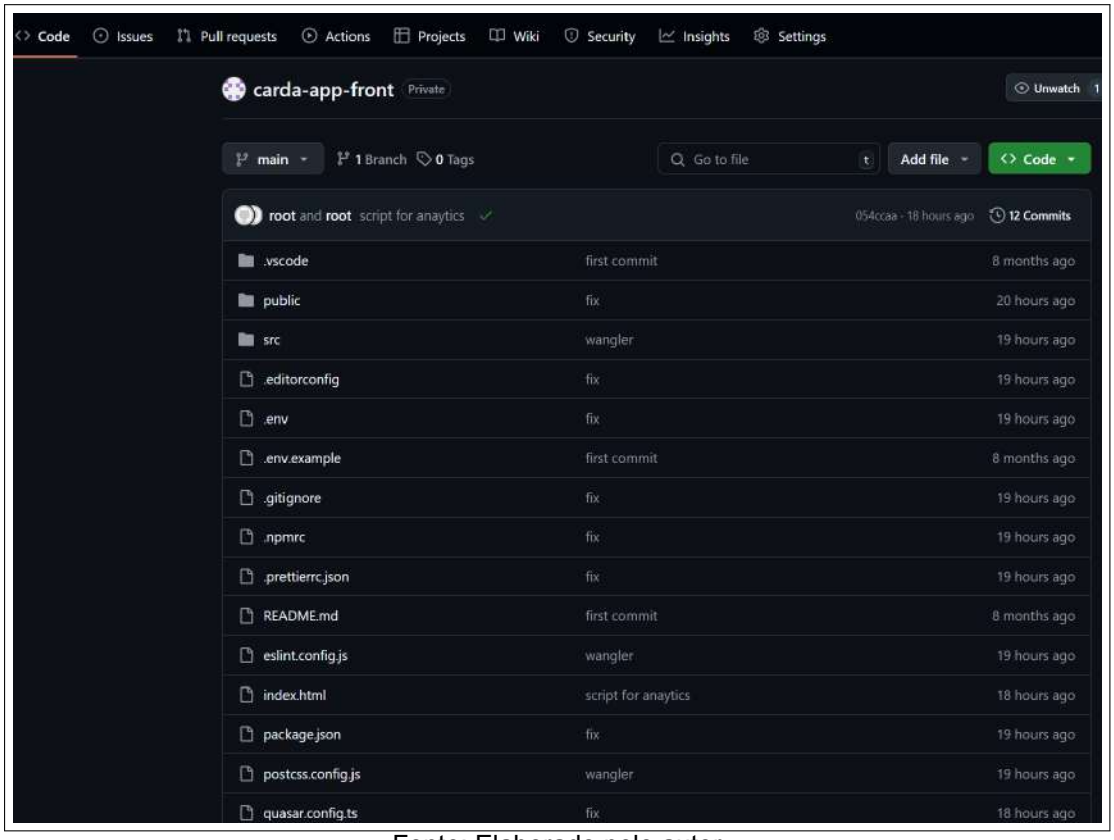
4.2.2 Deploy do *Frontend* na Cloudflare

Para hospedar o ambiente de produção do *frontend*, foi criado uma conta na Cloudflare, utilizando o próprio plano gratuito, o qual já oferece o suporte necessário para o estudo de caso, que é o *deploy* de aplicações estáticas e dinâmicas pelo Pages. A Cloudflare permite integrações automáticas com os repositórios do GitHub (CLOUDFLARE, 2023a).

O projeto foi conectado diretamente ao repositório do GitHub, conforme ilustrado na figura 41. Permitindo que cada novo push ou merge na branch principal utili-

zada, o Pages realize o *build* do projeto e publique automaticamente esse novo build.

Figura 41 – *Frontend* no GitHub



Fonte: Elaborado pelo autor

Existem duas abordagens possíveis para configurar o *deploy* no Cloudflare Pages, o contexto utilizado foi o do próprio GitHub, mas são elas:

- **Conexão direta com o Git**: forma mais comum utilizada, onde o próprio Cloudflare monitora o repositório e realiza o *build* sempre que alguma alteração sobe na branch principal. É possível configurar variáveis de ambiente, comandos personalizados de *build*, os caminhos que serão monitorados do projeto, entre outras coisas.
- **Uso de Wrappers e CLI**: como outra alternativa, é possível também realizar o *deploy* manual, ou configurar uma *pipeline* com o GitHub Actions utilizando o Wrangler, a CLI oficial da Cloudflare. Essa abordagem define mais flexibilidade no processo de gerenciar os ambientes.

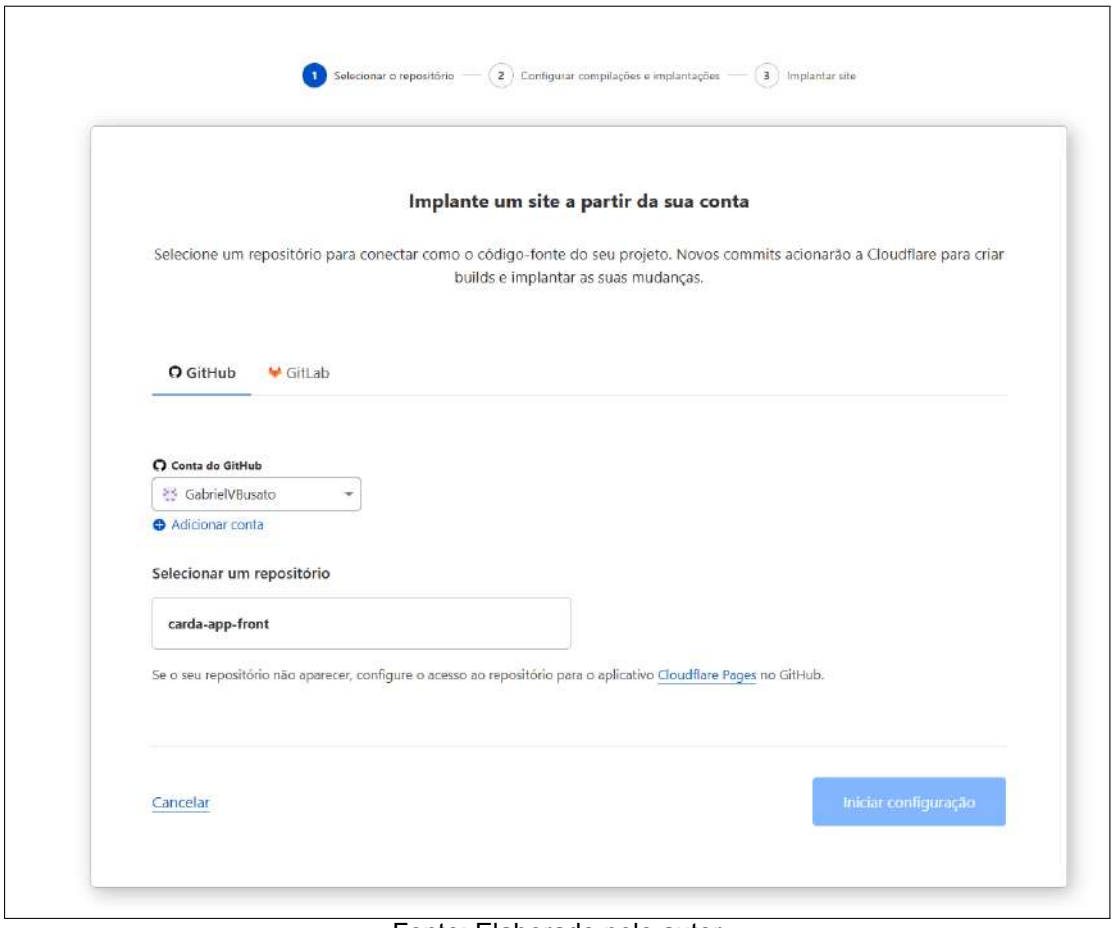
De início é necessário criar uma aplicação no Pages, representado pela figura 42.

Figura 42 – Criação da aplicação

Fonte: Elaborado pelo autor

Diante disso, é obrigatório seguir três passos de configuração para o *deploy*, sendo o primeiro a escolha do repositório, representado pela figura 43, tanto quanto a conta do GitHub que será utilizada.

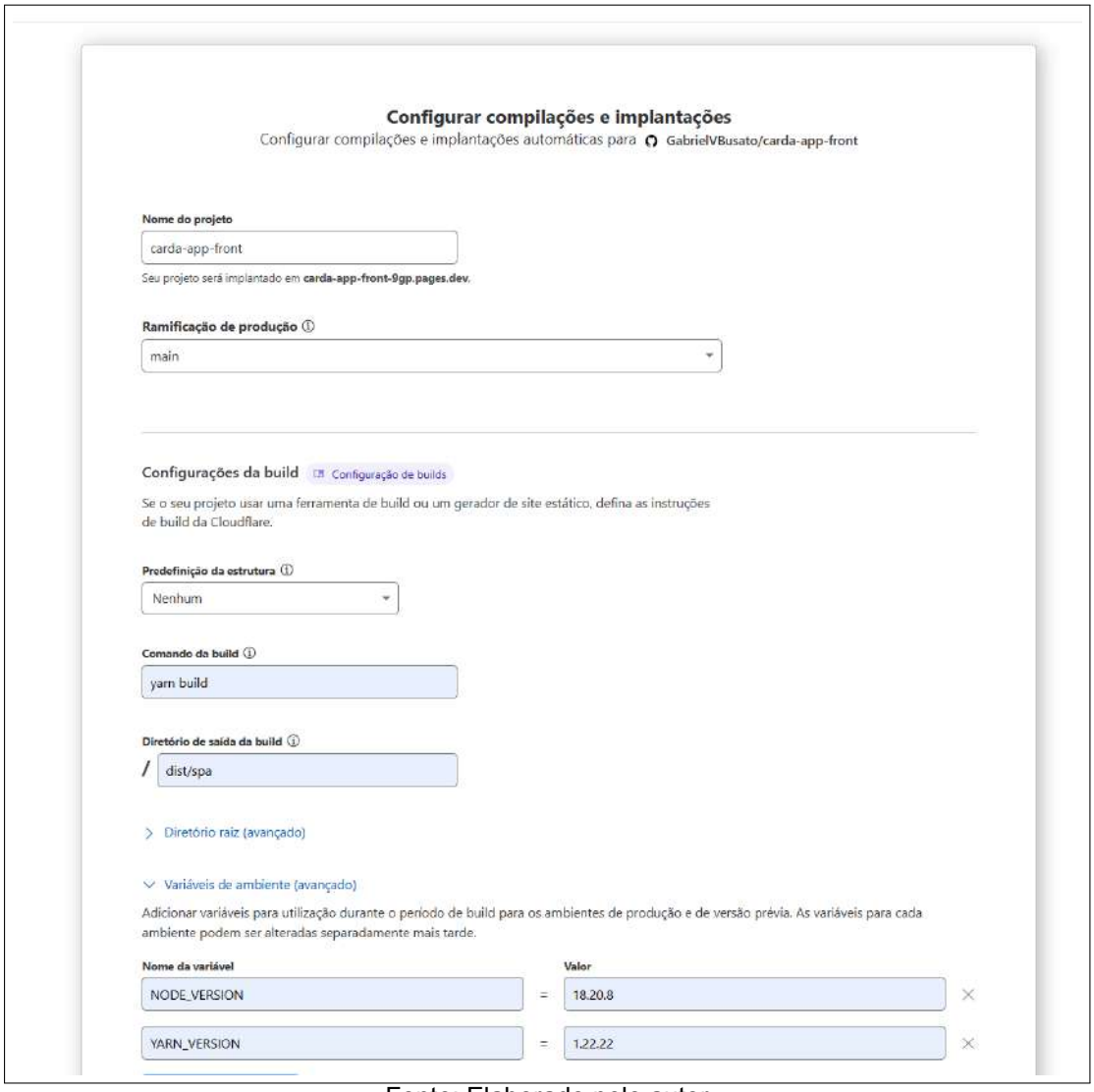
Figura 43 – Conexão GitHub



Fonte: Elaborado pelo autor

De segunda etapa temos a parte de configuração do nome do projeto, a *branch* que será utilizada como produção, as configurações do *build*, contando com o comando de gerar o *build*, o diretório de saida do *build* e as variáveis de ambiente necessárias para o projeto, essas configurações são ilustradas pela figura 44.

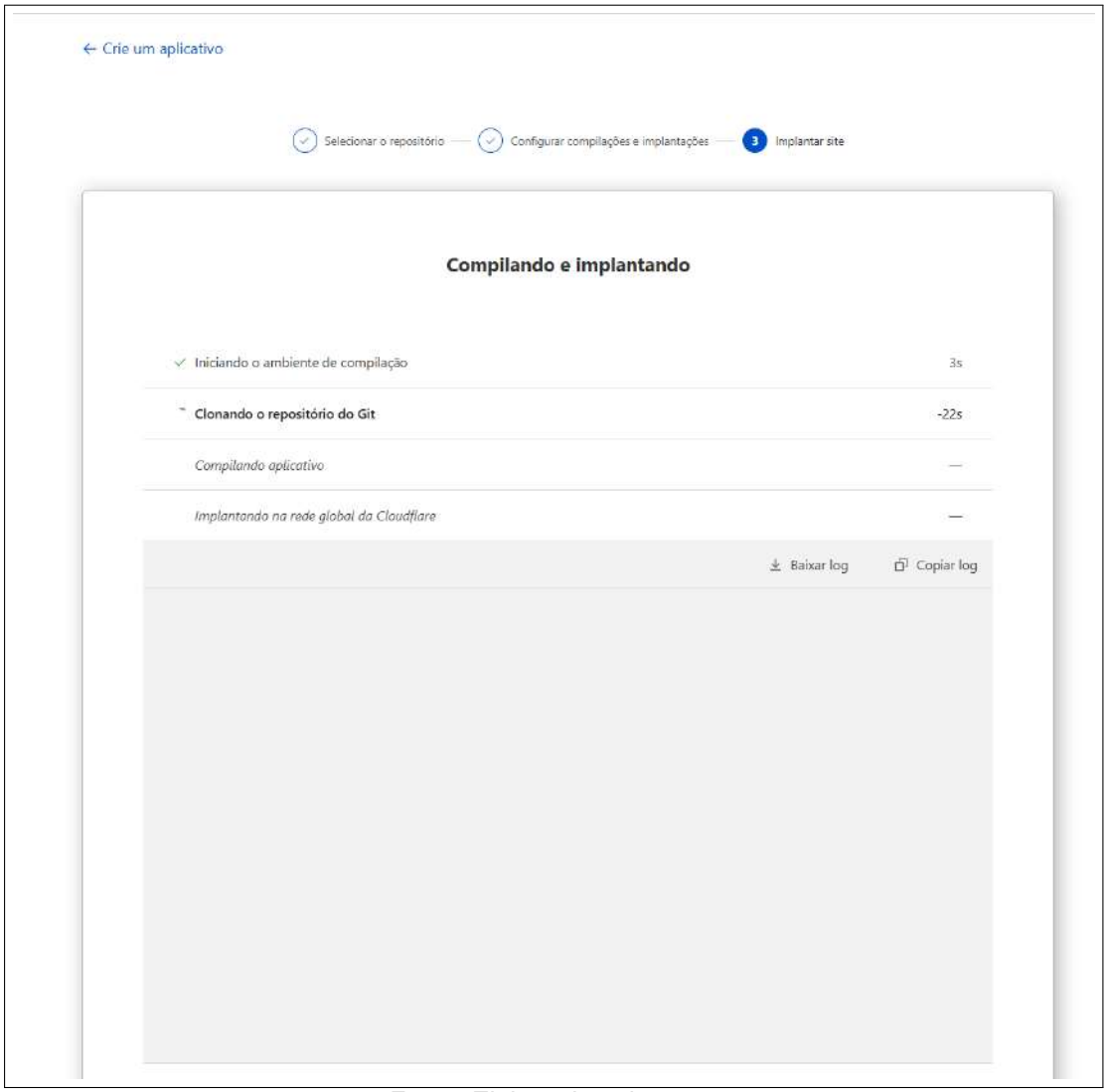
Figura 44 – Configuração ambiente e variáveis



Fonte: Elaborado pelo autor

Como terceira e última etapa, a própria Cloudflare tem sua *pipeline* de *deploy*, contando com a parte de iniciar o ambiente de compilação, clonar o repositório do git, compilar os arquivos e gerar o build, e por fim implantar no CDN da Cloudflare, essas etapas são representadas pela figura 45.

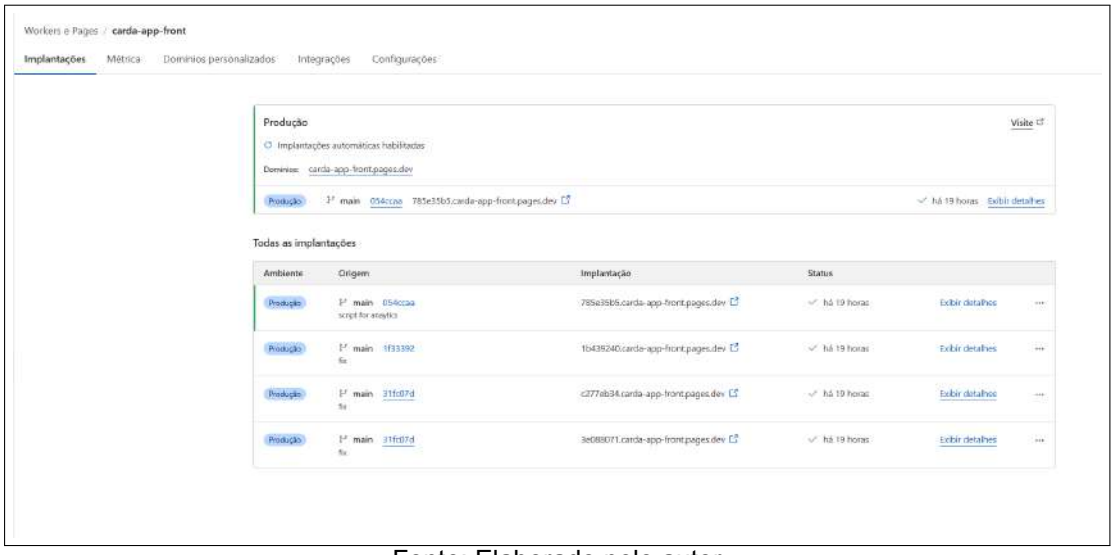
Figura 45 – Build Cloudflare e *Deploy*



Fonte: Elaborado pelo autor

Após feita a publicação, é obtida a listagem de todas as implementações feitas baseadas em cada push para o GitHub, representado pela figura 46, onde também é possivel configurar métricas, domínios personalizados, integrações e outras configurações.

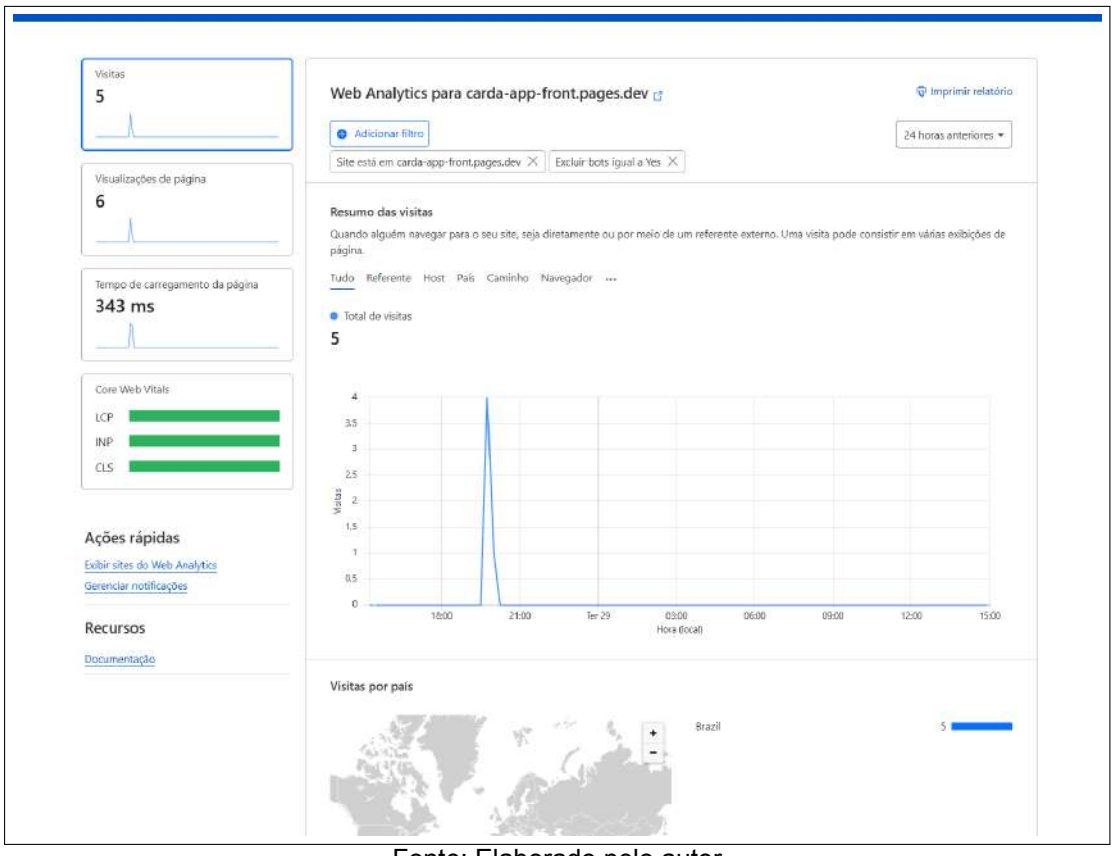
Figura 46 – Listagem de implementações



Fonte: Elaborado pelo autor

Para parte do estudo de caso, a parte de analytics foi configurada utilizando o domínio próprio gerado pela Cloudflare, representado pela figura 47, em que pode ser vista toda a parte de monitoramento relacionada ao Core Web Vitals, visitas e tempo de carregamento do projeto em questão.

Figura 47 – Web Analytics do *deploy*

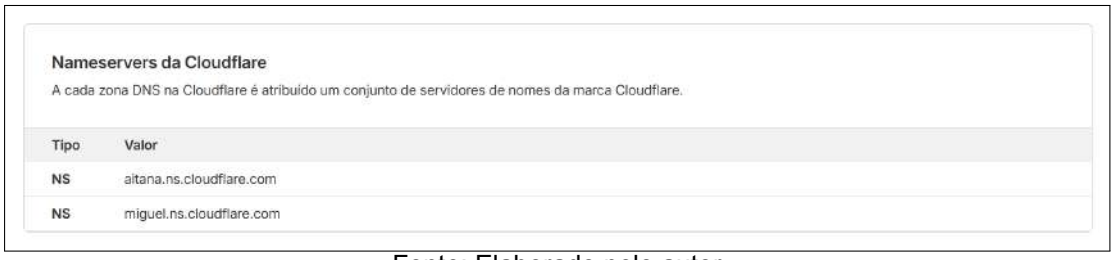


Fonte: Elaborado pelo autor

4.2.2.1 Configuração Domínio na Cloudflare

A fim de utilizar o domínio e os registros de servername pelo Cloudflare Pages, foi necessário importar o mesmo para o servername da Cloudflare, representado pela figura 48.

Figura 48 – Domínio na Cloudflare



Fonte: Elaborado pelo autor

Após feita essa configuração, foi necessário registrar também o DNS para a url do Load Balancer, e também para a url do Cloudflare Pages, a qual pode ser vista na imagem 49.

Figura 49 – DNS na Cloudflare

Fonte: Elaborado pelo autor

Feito isso, foi possível selecionar um domínio personalizado para o Cloudflare Pages ter o SSL/TLS. O qual pode ser visto na figura 50.

Figura 50 – Pages Domínio personalizado

Fonte: Elaborado pelo autor

5 Monitoramento e coleta de métricas da Cloudflare e AWS

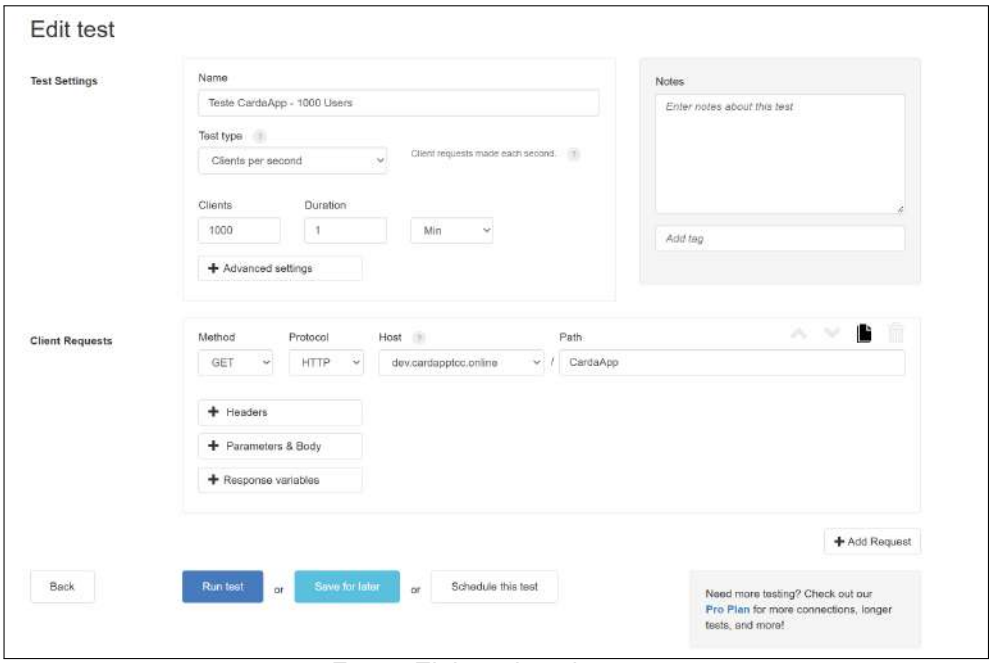
Essa seção tem como principal objetivo a coleta de dados feita tanto na EC2 quanto na Cloudflare. Contando com duas possíveis subseções: a primeira relatando os testes e resultados das métricas, antes de configurar a Cloudflare, e a segunda com a coleta de dados realizada após.

Como principal objetivo, foi utilizada a plataforma Loader.io para a requisição e o monitoramento dos tempos de respostas da Cloudflare Pages em relação ao cardápio representado na figura 24. Consiste em uma página com várias imagens pesadas sem o menor tratamento pelo Cloudflare Images, e também sem possuir nenhum tipo de *cacheamento*, WAF ou proxy[1].

5.1 Testes e Métricas antes da configuração Cloudflare

Com a configuração do teste na plataforma Loader representado pela figura 51, foi gerado o levantamento de dados em relação a como se comporta a Cloudflare em questão de múltiplos acessos.

Figura 51 – Configuração do teste

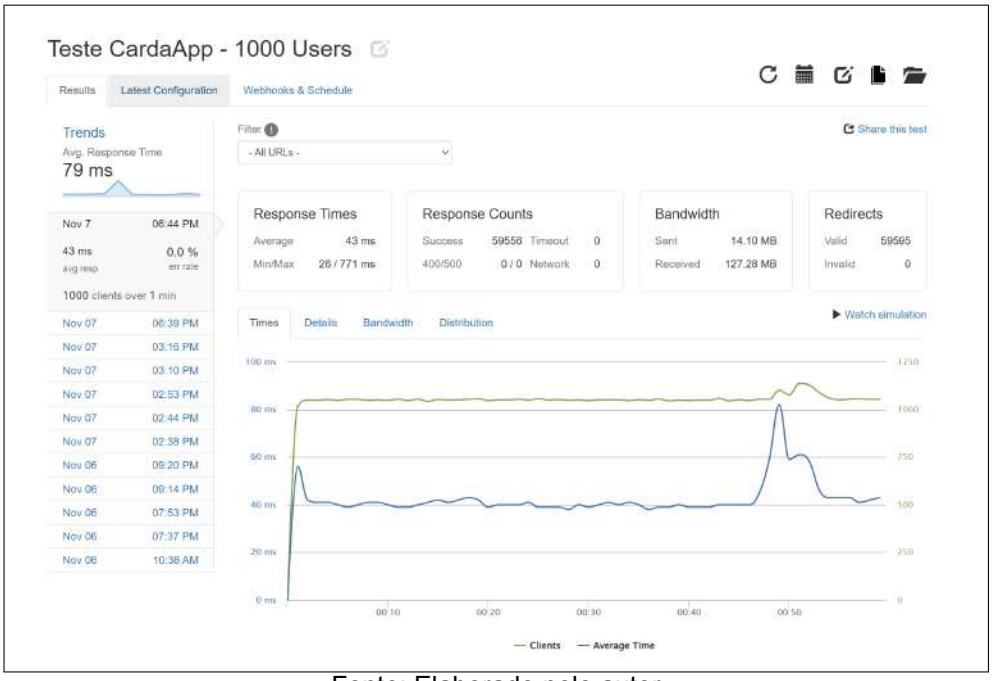


Fonte: Elaborado pelo autor

[1] Cacheamento: armazenamento temporário de dados para acelerar acessos; WAF (*Web Application Firewall*): firewall para proteção de aplicações web; Proxy: servidor intermediário entre cliente e servidor de origem.

Sendo a configuração do teste para 1000 usuários por segundo em 1 minuto, contando então com mais de 60 mil usuários, sendo 60 mil visualizações de html e 120 mil requisições, sendo arquivos js e outros. O tempo de resposta da plataforma Cloudflare pode ser visualizado na figura 52.

Figura 52 – Resultado do teste no loader

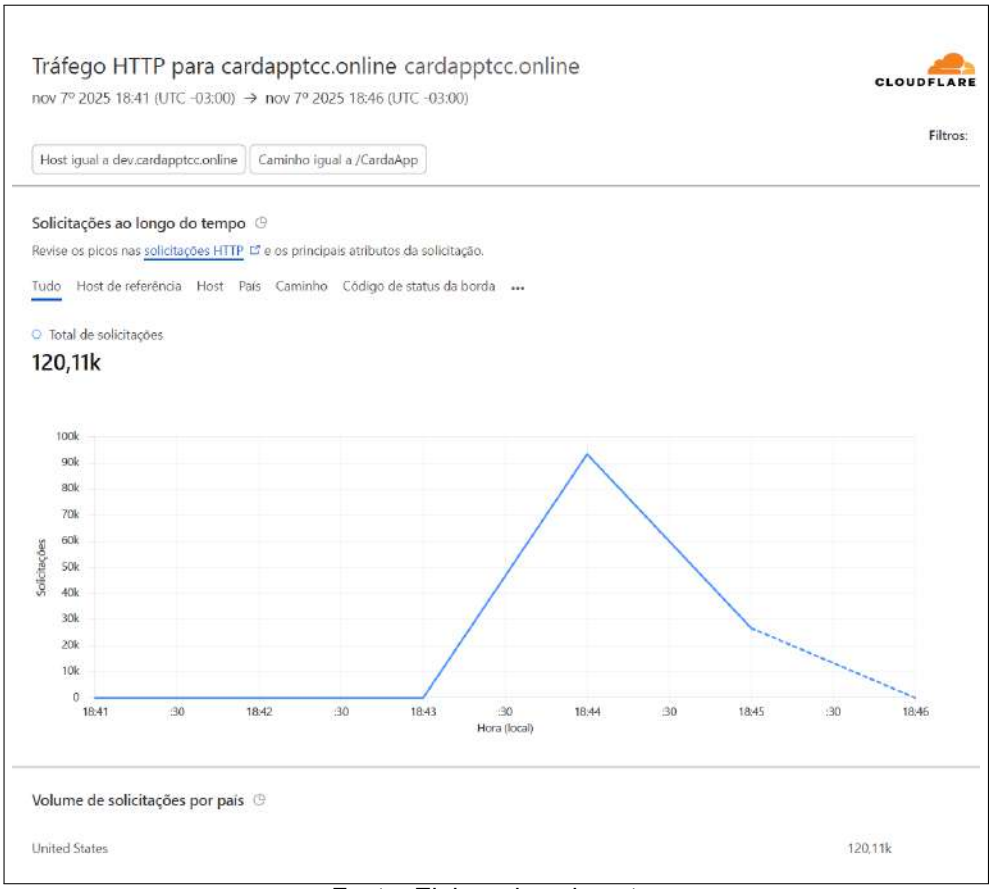


Fonte: Elaborado pelo autor

A plataforma processou as 60 mil requisições em uma performance média de 43 ms, com uma taxa de sucesso de 99.26%. Porém, teve um pico de latência alta, onde o tempo de resposta dobrou, resultando em 444 falhas. Esse comportamento mostra que o sistema atingiu um limite de capacidade temporário. A Cloudflare permitiu todas as requisições sem barrar em regras de proxy e *Anti-DDoS*, por não estarem configuradas.

Com a possibilidade de se monitorar essas informações pela Cloudflare Analytics, temos a figura 53, que representa todas as requisições HTTP feitas pelo Loader.

Figura 53 – Resultado da requisições HTTP



Fonte: Elaborado pelo autor

A Cloudflare (Pelo tráfego HTTP) processou 120.110 requisições em um intervalo de 5 minutos. Esse volume é maior que as métricas do loader por simplesmente contabilizar os *redirects* http para https, *retry*, entre outras coisas.

O gráfico apresenta uma medida de *ramp-up* e *ramp-down* que normalmente é visto em testes de carga, a taxa no máximo alcançou aproximadamente 1.8 mil requisições por segundo. O Quadro 7 representa as informações do teste no loader.

Quadro 7 – Resultados do teste Loader.io - 60.000 requisições (Cenário ANTES)

Métrica	**Valor**	**Observação**
Response Time		
Tempo médio	43ms	*Average*
Tempo mínimo	26ms	*Min*
Tempo máximo	771ms	*Max*
Desvio (máx - mín)	745ms	Variação
Performance e Confiabilidade		
Requisições bem-sucedidas	59.556	99.26%
Requisições falhadas	444	0.74%
Timeouts	0	Nenhum *timeout*
Erros de rede	0	Nenhum erro de rede
Taxa de erro	0.0%	*Error rate*
Bandwidth		
Dados enviados	14.10 MB	Requisições
Dados recebidos	127.28 MB	Respostas
Bandwidth total	141.38 MB	Soma de envio/recebimento
Tamanho médio da resposta	~2,13 KB	Por requisição bem-sucedida
Razão envio:recebimento	1:9	-
Redirects		
Redirects válidos	59.595	-
Redirects inválidos	0	-

Fonte: Fonte: Elaborado pelo autor

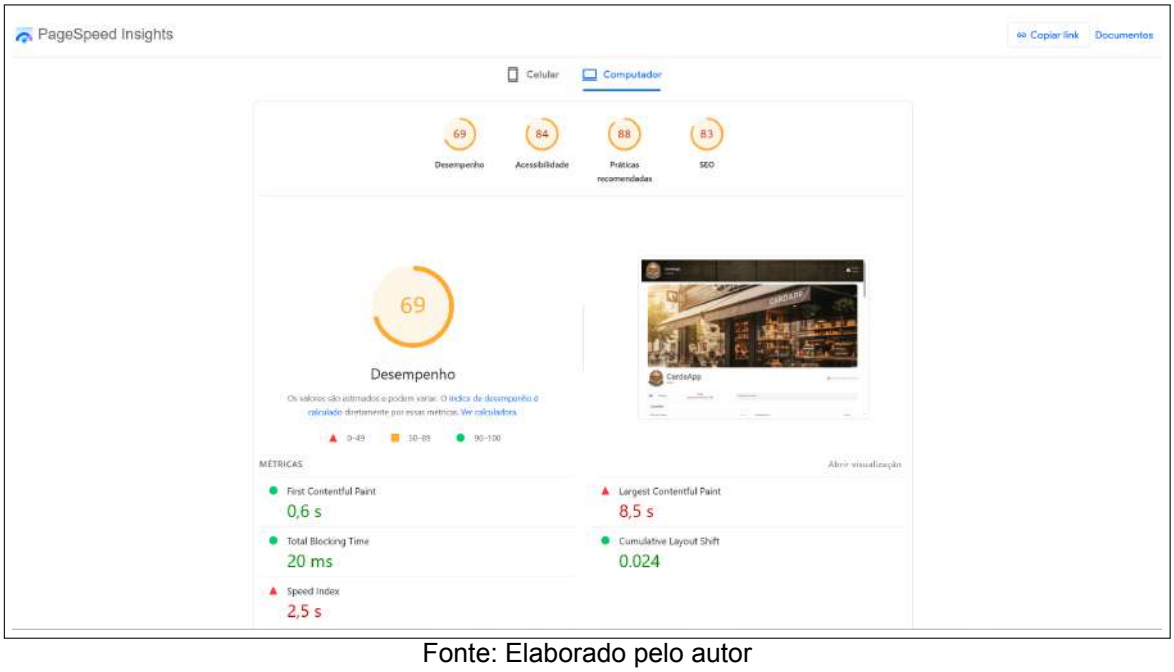
5.1.1 Resultados dos testes no Lighthouse e PageSpeed

A fim de avaliar o desempenho da aplicação antes de se aplicar as otimizações da Cloudflare, foram realizados testes utilizando o PageSpeed e o Lighthouse. Ambos realizados na versão desktop.

A Figura 54 ilustra os resultados do PageSpeed. A pontuação geral de desempenho ficou em 69 pontos em uma escala de 0 a 100. Ficando classificada como faixa amarela, a qual indica necessidade de melhorias urgentes. As demais categorias, que dependem mais do código da aplicação em si como acessibilidade e práticas recomendadas ficaram em 84 e 83 respectivamente. A análise detalha as métricas do Core Web Vitals, observando que o FCP registrou 0.6 segundos com *status* verde, indicando que o primeiro elemento renderizou rapidamente.

Outras métricas como o *Total Blocking Time* apresentou 20 milissegundos, o cumulative layout também se manteve bem superior com 0.024, bem abaixo do limite de 0.1 indicando alta estabilidade visual. Porém, duas métricas criticas se apresentam com *status* em vermelho: o LCP com 8.5 segundos, apresentando 340% acima do limite recomendado de 2.5 segundos e o Speed Index com 2.5 segundos, o qual indica lentidão na renderização.

Figura 54 – Desempenho PageSpeed



Fonte: Elaborado pelo autor

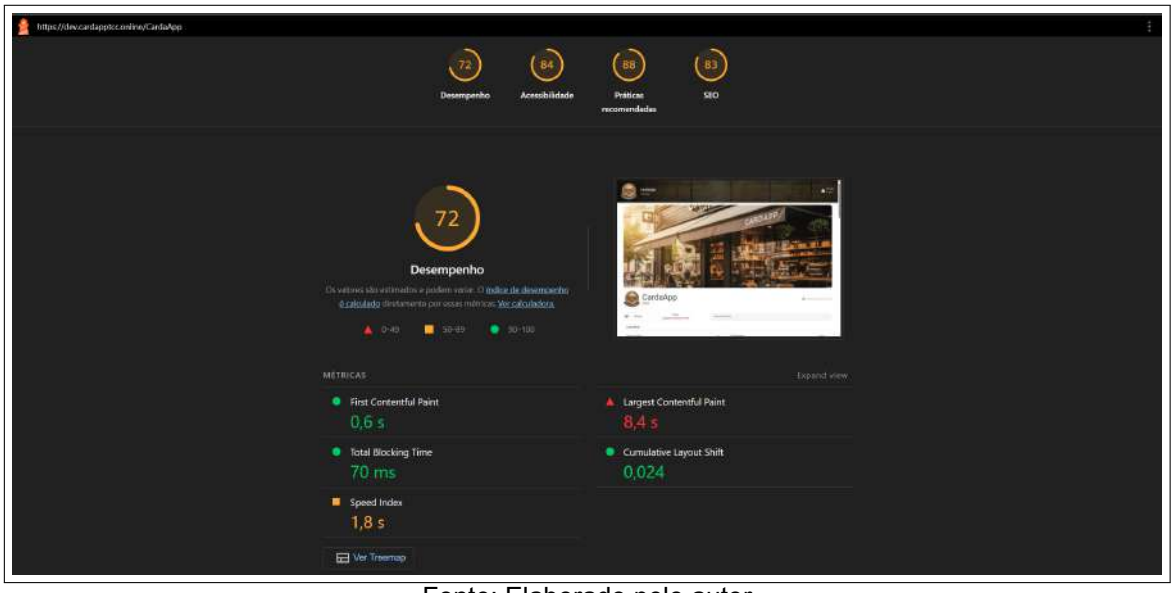
Em relação as imagens da aplicação, a figura 55 do PageSpeed Images apresenta um relatório o qual identifica que deve-se melhorar e muito a entrega de imagens, destacando alerta e apontando uma economia estimada de até 32.040 KiB, o que equivale a 3.3 MB. Este valor é um comparativo entre o tamanho das imagens atualmente e o tamanho que poderiam ter, com compressão, otimização e redimensionamento.

Figura 55 – Desempenho PageSpeed Images

Fonte: Elaborado pelo autor

Em relação aos testes do lighthouse, representaram basicamente métricas idênticas ao PageSpeed, as quais podem ser vistas na figura 56.

Figura 56 – Desempenho Lighthouse



Fonte: Elaborado pelo autor

Atualmente, as imagens armazenadas na instância EC2 não passam por um processo de redimensionamento, compressão ou otimização. A omissão dessa configuração tem impacto direto no desempenho da aplicação, o que pode ser observado na figura 57. É visível também o tráfego das imagens com tamanho excessivo, chegando a 1.396KB, o que resulta em um tempo de carregamento elevado, atingindo picos de até 1,16. O fato de não estar em um CDN da Cloudflare também impacta ainda mais no tempo de resposta.

Figura 57 – Imagens não otimizadas

Name	Status	Type	Initiator	Size	Time
pepperoni.png	200	png	Other	1,396 kB	1.16 s
chicken-burger.jpeg	200	jpeg	Other	1,402 kB	953 ms
napolitana.jpeg	200	jpeg	Other	1,316 kB	819 ms
portuguesa.jpeg	200	jpeg	Other	1,305 kB	724 ms
pizza-marguerita.jpeg	200	jpeg	Other	1,320 kB	711 ms
fanta-laranja-350ml.jpeg	200	jpeg	Other	388 kB	703 ms
classic-burger.jpeg	200	jpeg	Other	1,527 kB	668 ms
monster-burguer.jpeg	200	jpeg	Other	1,551 kB	639 ms
quatro-queijos.jpeg	200	jpeg	Other	117 kB	639 ms
calabresa.jpeg	200	jpeg	Other	171 kB	638 ms
frango-catupiry.jpeg	200	jpeg	Other	220 kB	637 ms
file.png	200	png	Other	4,989 kB	628 ms
suco-maracuja.jpeg	200	jpeg	Other	1,306 kB	611 ms
x-egg-bagon-burger.jpeg	200	jpeg	Other	1,608 kB	522 ms
hamburgacia.jpg	200	jpeg	Other	759 kB	431 ms
cardapp.png	200	png	Other	1,878 kB	429 ms
xtudo.png	200	png	Other	720 kB	410 ms
supremo.png	200	png	Other	859 kB	396 ms
kids.png	200	png	Other	37.0 kB	313 ms
risoto-funghi.jpeg	200	jpeg	Other	1,501 kB	312 ms
bacon_duplo.png	200	png	Other	287 kB	309 ms
lasanha-forno.jpeg	200	jpeg	Other	1,447 kB	280 ms
background-header.jpg	200	jpeg	index-Dqfd9ZTm.css	1,526 kB	237 ms
strogonoff-frango.jpeg	200	jpeg	Other	1,363 kB	227 ms
petit-gateau.jpeg	200	jpeg	Other	1,412 kB	222 ms
brownie.jpeg	200	jpeg	Other	1,467 kB	214 ms
carda-app-logo.png	200	png	Other	687 kB	211 ms
feijoada.jpeg	200	jpeg	Other	527 kB	197 ms
iscas-de-frango.jpeg	200	jpeg	Other	130 kB	185 ms
guarana-antarctica-350ml.jpeg	200	jpeg	Other	23.6 kB	182 ms

Fonte: Elaborado pelo autor

5.1.2 Teste de Desempenho com *Puppeteer*

Com o objetivo de avaliar o desempenho do Load Balancer e da instância EC2, bem como também validar a coleta de métricas pelo Web Analytic, foi utilizado um cenário de teste focado no impacto ao *backend* na ausência de *cacheamento* correto. As requisições são feitas direcionadas diretamente ao *backend*. A utilização do *Puppeteer* se dá pelo fato do Loader.io apenas fazer requisições via protocolo HTTP/HTTPS e não renderizar a página em si. Mais detalhes da configuração podem ser vistos no quadro 8.

Quadro 8 – Configuração do teste de desempenho com Puppeteer

Parâmetro	**Valor**	**Descrição**
Total de usuários simulados	200	Número total de sessões de navegador
Navegadores concorrentes	2	Instâncias simultâneas do Chrome
Delay entre ações	2000ms	Tempo de espera entre navegações
Timeout de navegação	60000ms	Limite de tempo para carregamento
Modo de execução	*Headless*	Navegador sem interface gráfica
URL testada	/CardaApp	*Endpoint* do *frontend*
Estratégia de espera	*networkidle2*	Aguarda rede quase ociosa
Resoluções do *viewport*	1920x1080	Simulação de tela desktop
	1366x768	Simulação de notebook
	414x896	Simulação de smartphone
	768x1024	Simulação de tablet
Delay entre lançamentos	1000ms	Intervalo para iniciar novo navegador

Fonte: Elaborado pelo autor.

5.1.2.1 Resultados Teste do Puppeteer

Como resultado alcançado no teste feito no site, foram alcançadas as seguintes métricas, representadas pelo quadro 9.

Quadro 9 – Resultados do teste de desempenho com Puppeteer - 200 usuários

Métrica	**Valor**	**Avaliação**
Resumo Geral		
Total de testes	200	—
Taxa de sucesso	100%	Todos os testes concluídos
Falhas	0	Nenhuma falha registrada
Tempo total	108s (1,8 min)	Aproximadamente 0,54s por usuário
Core Web Vitals (Médias)		
LCP (Largest Contentful Paint)	3256ms	RUIM - Acima do ideal (2500ms)
FCP (First Contentful Paint)	3284ms	RUIM - Acima do ideal (1800ms)
CLS (Cumulative Layout Shift)	0,002	BOM - Dentro do ideal (<0,1)
Tempo de navegação médio	1217ms	Tempo de carregamento inicial
Análise de Performance		
Estabilidade visual	Excelente	CLS muito baixo indica página estável
Velocidade de carregamento	Necessita otimização	LCP e FCP acima dos valores recomendados
Confiabilidade	Alta	100% de taxa de sucesso

Fonte: Elaborado pelo autor.

A análise feita nos resultados no teste com Puppeteer mostra detalhes importantes sobre o desempenho da aplicação, mesmo com carga pequena de 200 usuários.

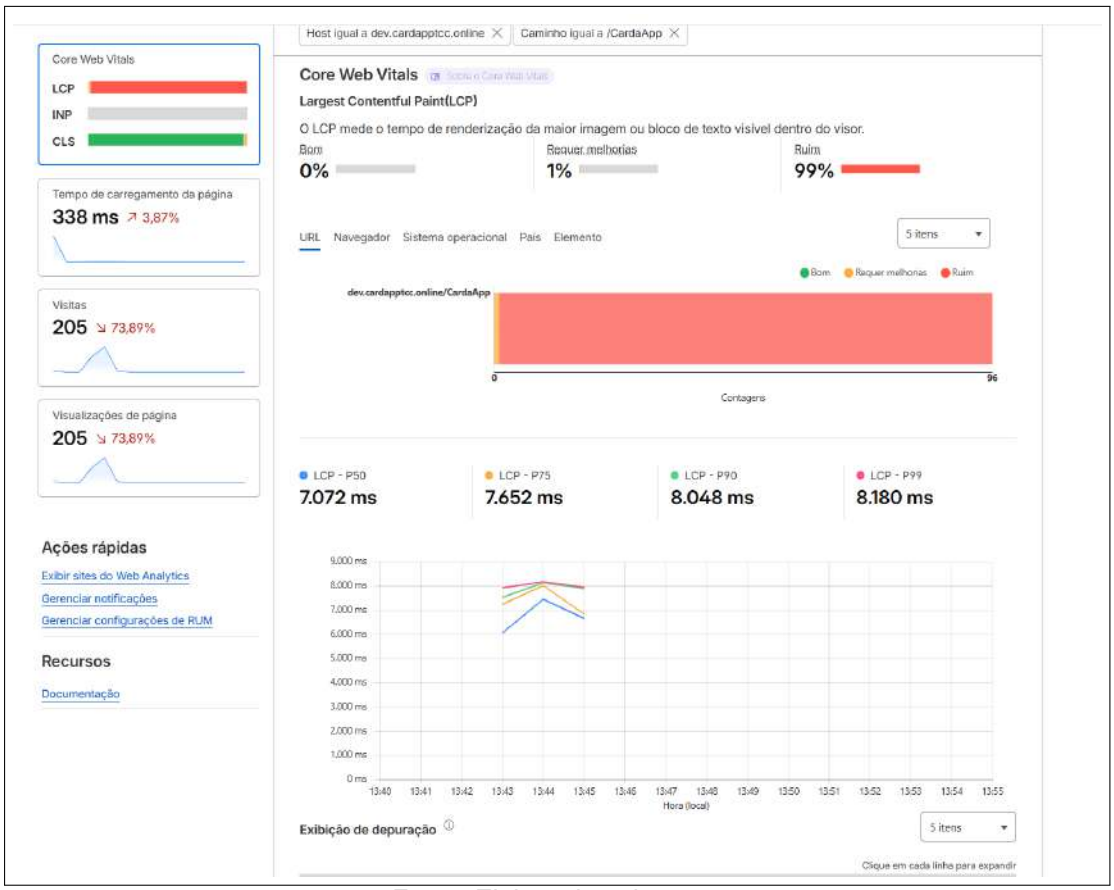
Quanto às Core Web Vitals, os resultados apresentam um cenário interessante. O CLS ficou como 0.002, o qual é bem superior. Porém, tanto o LCP quanto o FCP, apresentaram valores preocupantes, com 3256ms e 3282ms respectivamente. O ideal para o LCP seria abaixo de 2500ms, já para o FCP seria 1800ms.

5.1.2.2 Resultado Puppeteer: Análise dos Dados do Google Analytics - Core Web Vitals

A coleta feita pelo Google Analytics durante o período de testes revela informações críticas sobre a experiência dos usuários ao acessar a aplicação. Conforme pode ser observado na figura 58, a análise das Core Web Vitals apresenta um cenário que demanda atenção, especialmente no LCP.

Durante o período de testes, foram registradas 205 visitas e 205 visualizações de página. O gráfico representa a evolução do LCP ao longo do período, revelando picos que variam entre 6000ms e 8000ms, com certa instabilidade nos valores.

Figura 58 – Resultado: Core Web Vitals - Web Analytics



Fonte: Elaborado pelo autor

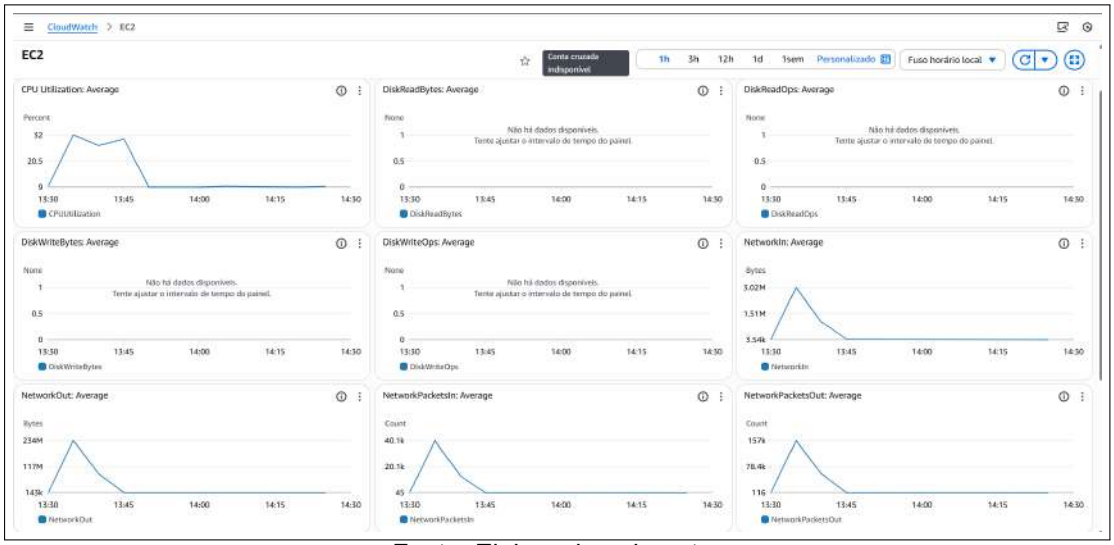
5.1.2.3 Resultado Puppeteer: Análise EC2

O monitoramento da instância EC2 foi feito através do AWS CloudWatch, durante a execução do teste com Puppeteer. Conforme observado na figura 59, a utilização de CPU apresentou um aumento significativo durante o período de teste, iniciando próximo a zero e atingindo picos altos durante o teste de 200 usuários.

O tráfego de rede de entrada demonstrou também o momento exato de início do teste, com um crescimento alto de 3.54KB para 3.02MB, refletindo todas as requisições simultaneas.

É de extrema importância destacar que, sem o *cache* da Cloudflare ativo, toda requisição do *frontend* resulta em múltiplas chamadas à EC2 para buscar dados, imagens, arquivos, entre outros.

Figura 59 – Resultado: EC2

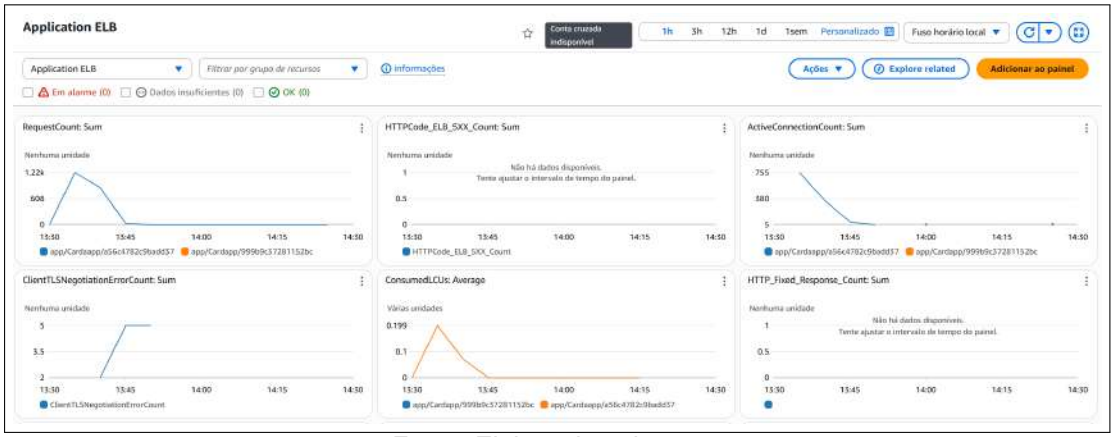


Fonte: Elaborado pelo autor

5.1.2.4 Resultado Puppeteer: Análise Load Balancer

O Application Load Balancer da AWS também foi monitorado pelo Cloudwatch, o qual é representado na figura 60. O gráfico de *RequestCount* mostra claramente o aumento de requisições, chegando a um topo maior que 1200 requisições, evidenciando o tráfego gerado pelos usuários.

Figura 60 – Resultado: Load Balancer



Fonte: Elaborado pelo autor

O número de conexões ativas do *ActiveConnectionCount* também apresentou esse comportamento similar, iniciando em níveis baixos e atingindo picos de 755 conexões.

5.2 Testes e Métricas após a configuração da Cloudflare

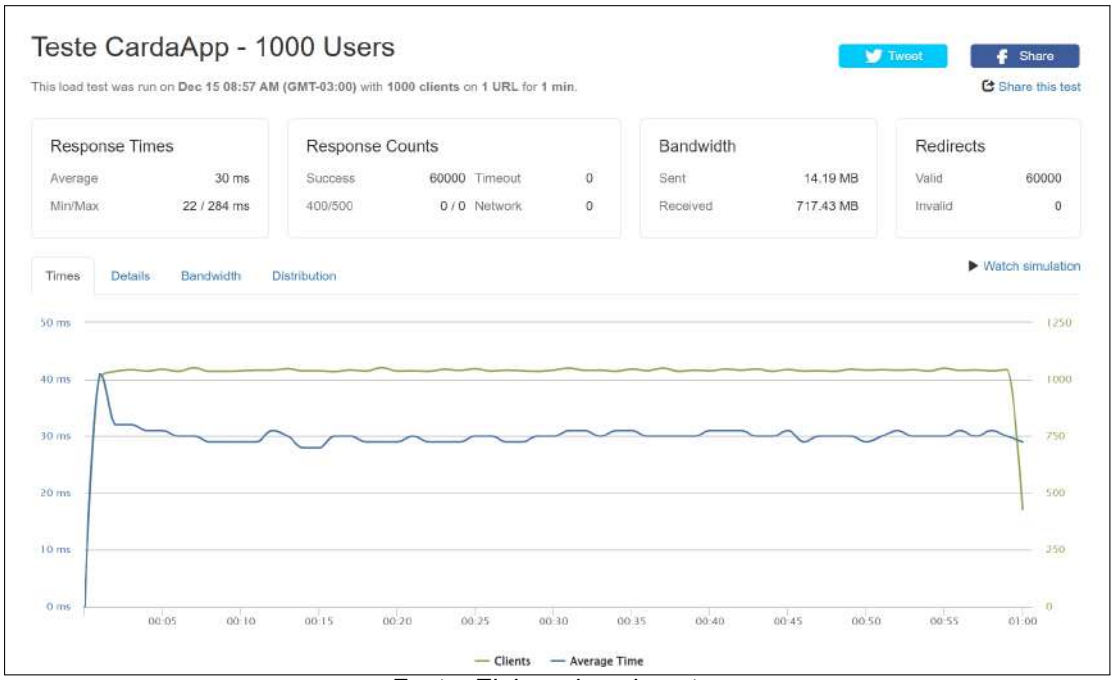
Essa seção tem como principal objetivo descrever todas as configuraçoes feitas na Cloudflare, contando também com testes e métricas de desempenho após as configurações.

5.2.1 Resultado Loader - Pós otimização

Foi utilizado o mesmo teste representado pela seção 5.1. A partir do mesmo, foi gerado o levantamento de dados do Loader.

O teste foi executado novamente mantendo a carga de 1000 usuários por segundo durante 1 minuto, totalizando novamente as 60 mil requisições. O novo comportamento da infraestrutura e o tempo de resposta podem ser visualizados na figura 61.

Figura 61 – Resultado Loader - Pós Otimização



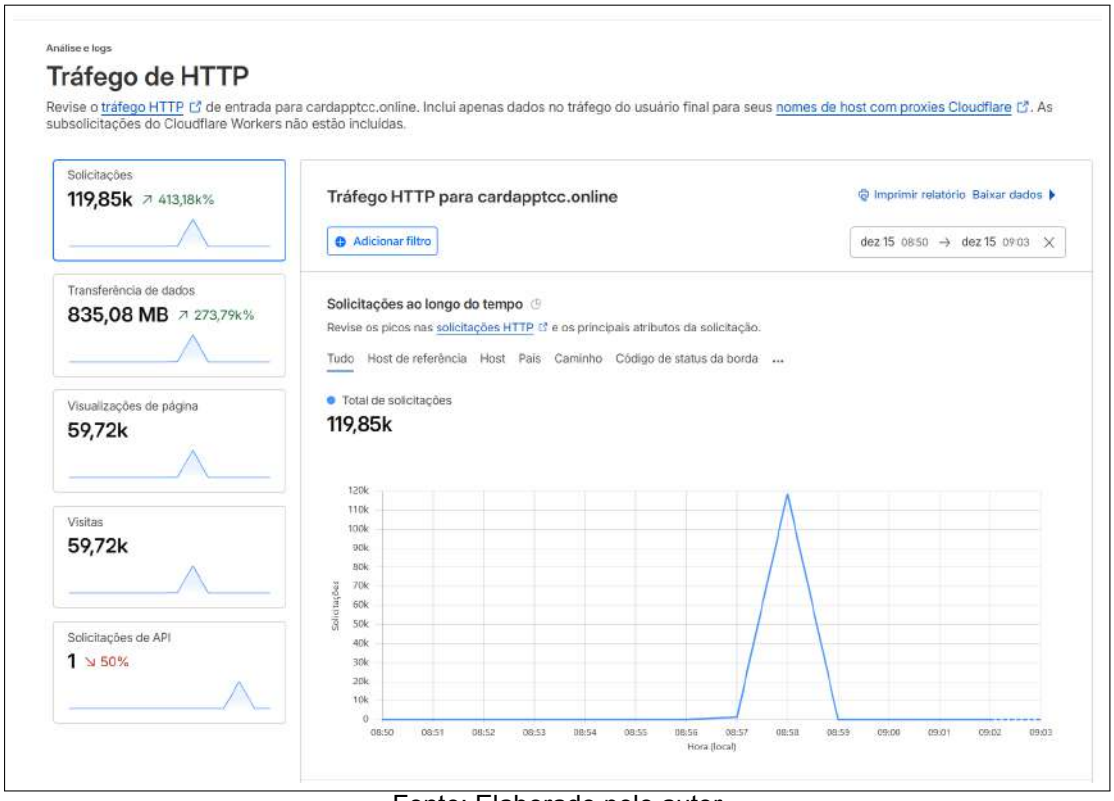
Fonte: Elaborado pelo autor

Neste cenário, a Cloudflare processou as requisições em uma performance média muito melhor que anteriormente, sendo de 30ms, representando um ganho de eficiência.

A taxa de sucesso foi de 100%, sem *timeouts*, erros de rede ou falhas HTTP. A latência não teve picos igual como anteriormente, variando entre um ponto de 22 ms até um máximo isolado de 284 ms.

Utilizando o monitoramento pelo *analytics*, temos a representação pela figura 62. A imagem em questão confirma o recebimento massivo de tráfego, registrando um pico de até 119,85 mil requisições.

Figura 62 – Resultado Requisições HTTP



Fonte: Elaborado pelo autor

O quadro 10 apresenta os resultados do teste de carga realizado em um cenário contendo 60 mil requisições, o mesmo pós otimizado, evidenciando a eficiência da estrutura implementada. Os dados demonstram que houve uma latência média de apenas 30ms, com uma estabilidade total de 100% de taxa de sucesso e ausência de falhas ou *timeouts*.

Quadro 10 – Resultados do teste Loader.io - 60.000 requisições (Cenário DEPOIS)

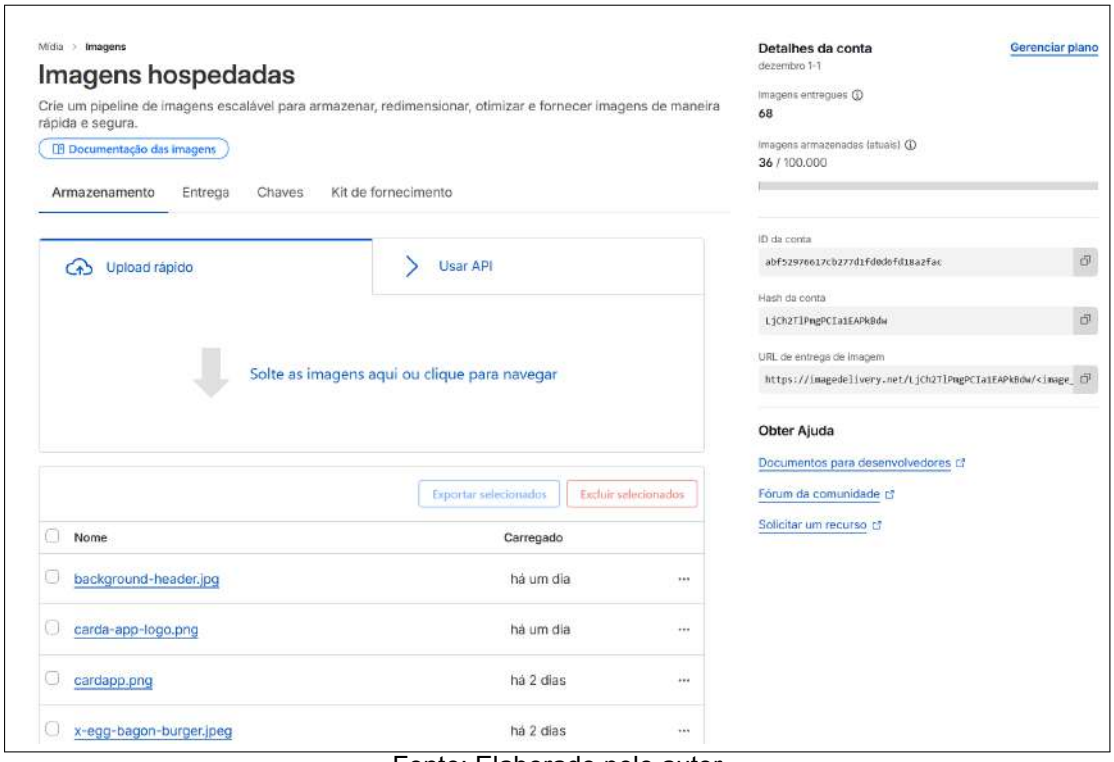
Métrica	**Valor**	**Observação**
Response Time		
Tempo médio	30ms	Melhora de 30%
Tempo mínimo	22ms	Estável
Tempo máximo	284ms	Sem picos extremos
Desvio (máx - mín)	262ms	Baixa variação
Performance e Confiabilidade		
Requisições bem-sucedidas	60.000	100% (Perfeito)
Requisições falhadas	0	0%
Timeouts	0	Nenhum *timeout*
Erros de rede	0	Nenhum erro de rede
Taxa de erro	0.0%	Estabilidade total
Bandwidth		
Dados enviados	14.19 MB	Requisições
Dados recebidos	717.43 MB	Respostas (*Cache Hit*)
Bandwidth total	731.62 MB	Alto *throughput*
Tamanho médio da resposta	~11,9 KB	Conteúdo completo entregue
Razão envio:recebimento	1:50	Alta eficiência
Redirects		
Redirects válidos	60.000	Fluxo correto
Redirects inválidos	0	-

Fonte: Fonte: Elaborado pelo autor

5.2.2 Configuração Cloudflare Images

Como primeiro passo para a otimização das imagens que estavam com um tempo de carregamento alto e com um tamanho em KB alto, como ilustrado pela figura 55, foi feito a migração das mesmas da instância EC2 para o Cloudflare Images, como pode ser visto na imagem 63.

Figura 63 – Imagens hospedadas no Cloudflare

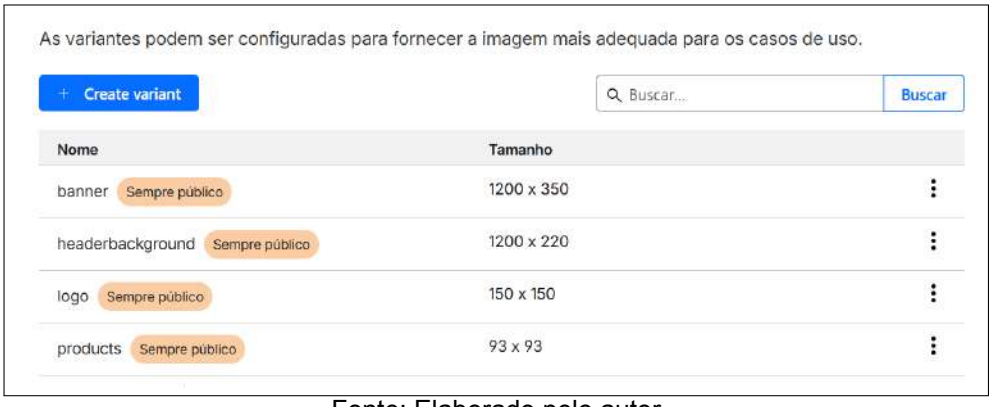


Fonte: Elaborado pelo autor

Para garantir essa otimização adequada, foi feito também o uso de *variants*, que consistem em redimensionamentos automáticos e compressão. Como por exemplo, caso um determinado sistema precise de uma imagem 100x100, porém caso só tenha a mesma em 1300x400, o Cloudflare Images consegue comprimir e redimensionar essas imagens através das *variants*, como pode ser visto na imagem 64.

Foram criadas quatro *variants*, sendo banner, headerbackground, logo e products.

Figura 64 – Cloudflare Images *Variants*

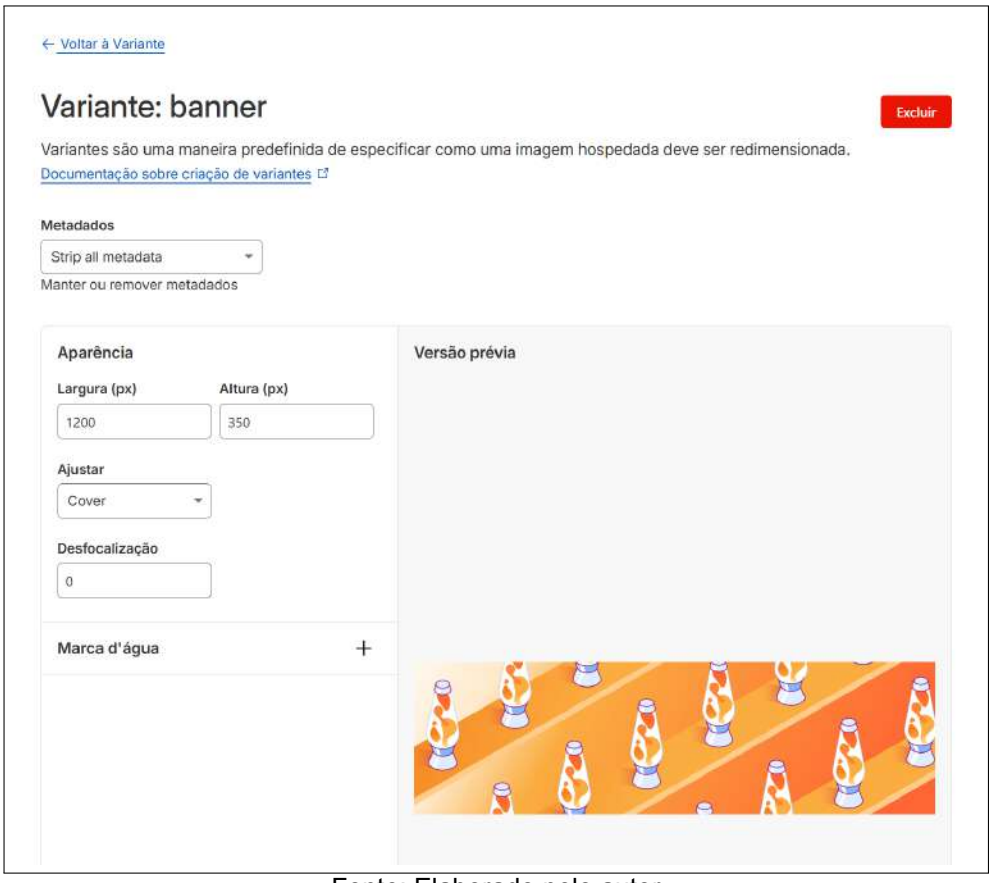


Fonte: Elaborado pelo autor

Na criação da mesma, é possivel escolher o tipo de ajuste, largura e altura, como

pode ser visto na figura 65. Com isso, independente do tamanho que foi adicionado a imagem na Cloudflare, esta será redimensionada e comprimida, além de ser entregue pela própria CDN da Cloudflare.

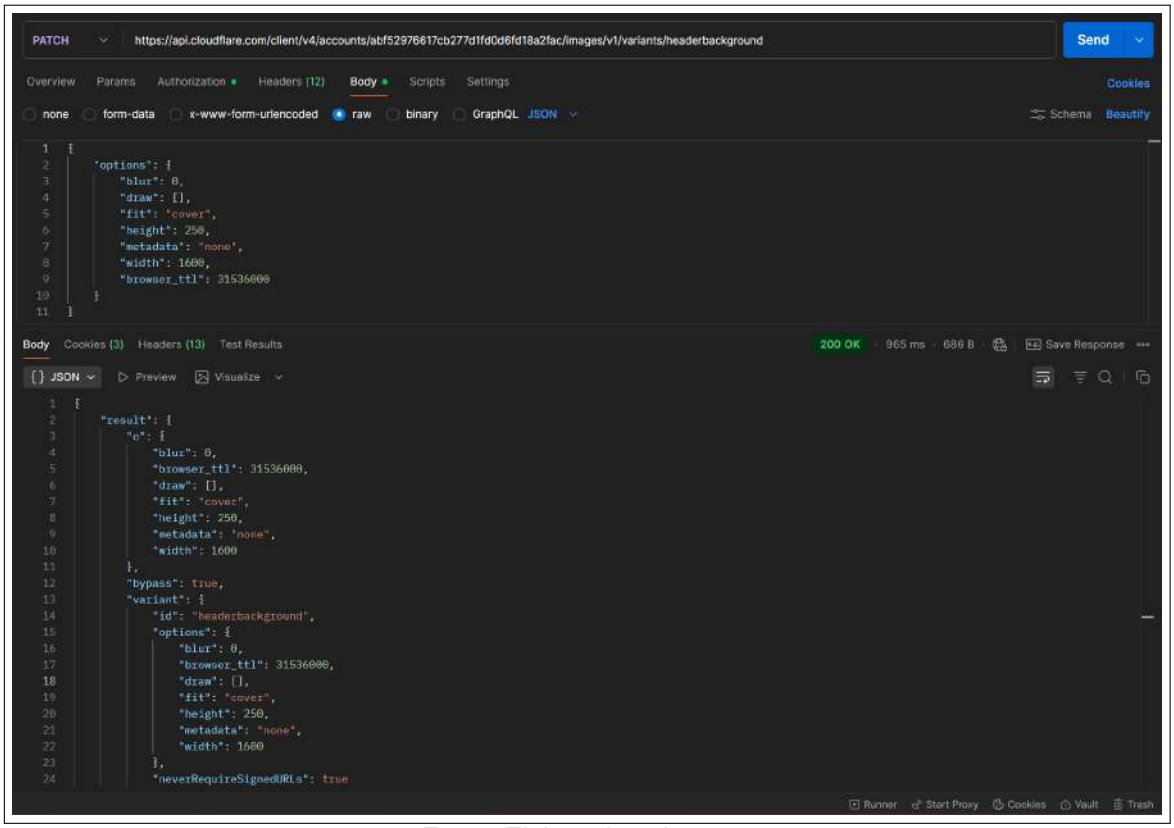
Figura 65 – Criando Variant



Fonte: Elaborado pelo autor

Um grande fator de preocupação também é o *TTL* - Time to Live (Tempo de vida, tradução nossa), o qual consta como dois dias como padrão da Cloudflare Pages. Porém, a alteração foi feita para manter o *cache* em navegador das imagens e arquivos estáticos para um valor padronizado, sendo o mesmo por mais de um ano, o qual pode ser visto na figura 66. Como não é possível a alteração via painel, foi feita uma requisição post utilizando as chaves da Cloudflare.

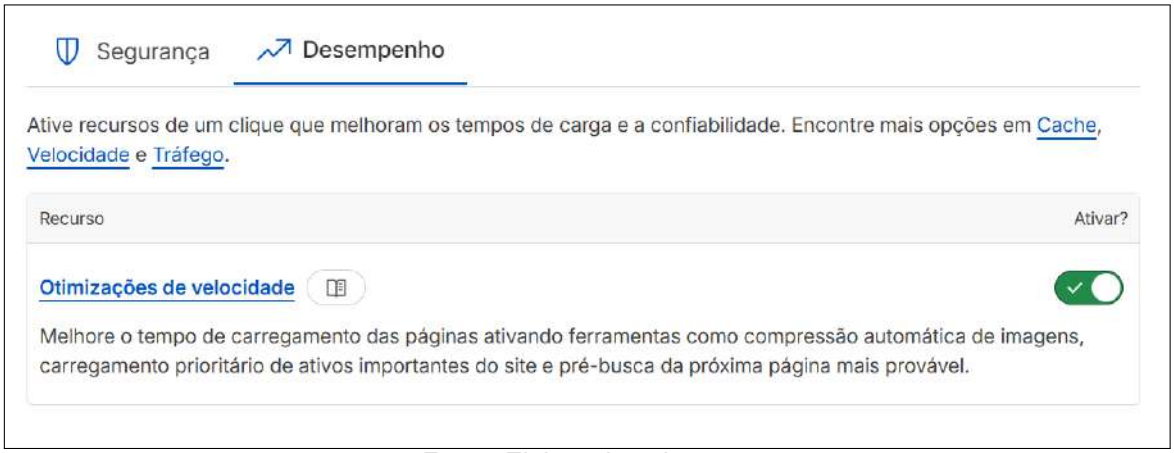
Figura 66 – Corrigindo TTL das *variants*



Fonte: Elaborado pelo autor

Além de todas configurações citadas, foi feita a ativação do recurso de ”Otimização de velocidade”, o qual garante um melhor tempo de carregamento das páginas, fazendo compressão automática de imagens, carregamento prioritário de arquivos estáticos e uma busca antecipada da próxima página mais provável, exemplificado pela figura 67.

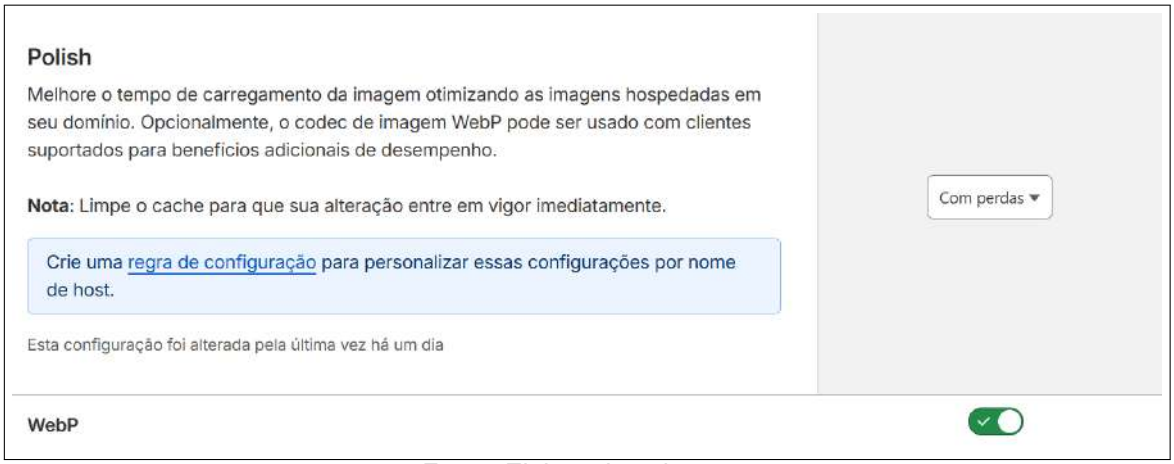
Figura 67 – Otimização de velocidade



Fonte: Elaborado pelo autor

Foi possível utilizar a otimização automática de ativos visuais também, configurada no modo com perdas. A abordagem aplica compressão que reduz drasticamente o tamanho dos arquivos, retirando metadados desnecessários, mantendo também a integridade visual, ilustrado pela figura 68.

Figura 68 – Polimento de images pelo Cloudflare



Fonte: Elaborado pelo autor

A implementação descrita até aqui, nesta seção, resultou em uma otimização significativa no desempenho, reduzindo muito o tempo de respostas das requisições de imagens, para uma média consistente entre 40ms e 95ms, como pode ser visto na figura 69.

Figura 69 – Imagens após otimização

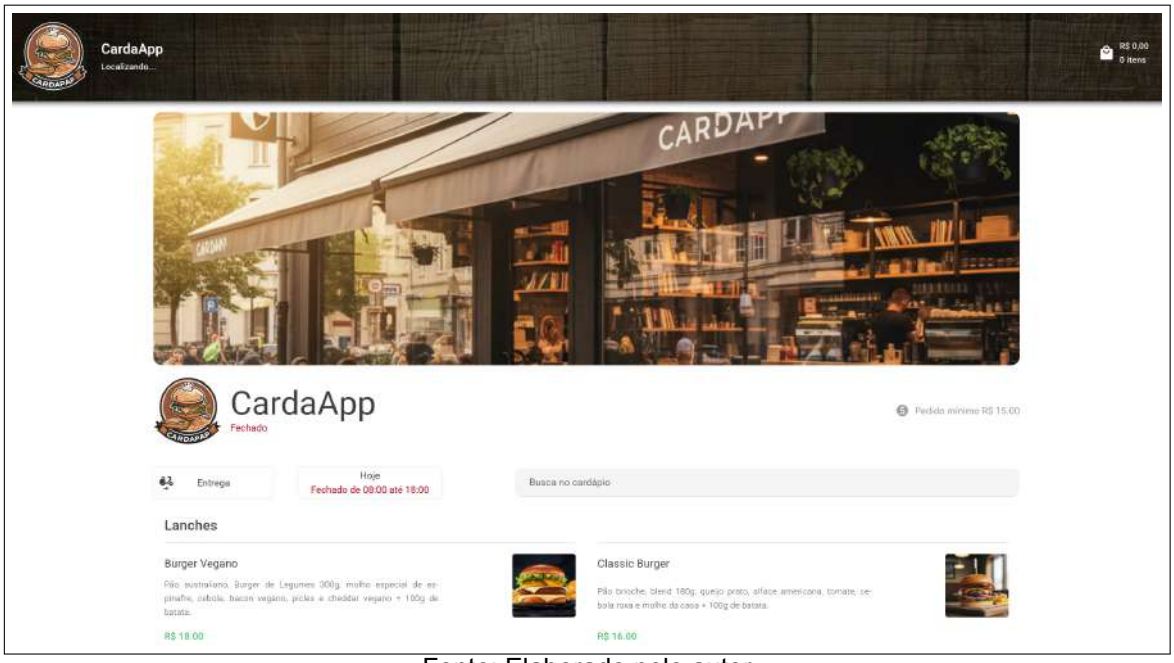
Name	Status	Type	Initiator	Size	Time
logo	200	avif	Other	9.5 kB	95 ms
headerbackground	200	avif	Other	56.5 kB	93 ms
banner	200	avif	Other	56.7 kB	93 ms
products	200	avif	Other	1.5 kB	84 ms
products	200	avif	Other	2.7 kB	64 ms
products	200	avif	Other	2.9 kB	64 ms
products	200	avif	Other	3.1 kB	59 ms
products	200	avif	Other	2.8 kB	59 ms
products	200	avif	Other	2.8 kB	58 ms
products	200	avif	Other	3.2 kB	57 ms
products	200	avif	Other	3.1 kB	56 ms
products	200	avif	Other	3.1 kB	56 ms
products	200	avif	Other	3.5 kB	55 ms
products	200	avif	Other	2.6 kB	55 ms
products	200	avif	Other	4.7 kB	51 ms
products	200	avif	Other	2.4 kB	51 ms
products	200	avif	Other	3.7 kB	51 ms
products	200	avif	Other	4.0 kB	51 ms
products	200	avif	Other	4.3 kB	51 ms
products	200	avif	Other	5.1 kB	51 ms
products	200	avif	Other	1.7 kB	50 ms
products	200	avif	Other	4.1 kB	49 ms
products	200	avif	Other	5.8 kB	49 ms
products	200	avif	Other	2.9 kB	49 ms
products	200	avif	Other	2.7 kB	47 ms
products	200	avif	Other	2.8 kB	46 ms
products	200	avif	Other	4.3 kB	46 ms
products	200	avif	Other	3.2 kB	45 ms
products	200	avif	Other	2.6 kB	44 ms

34 / 62 requests | 224 kB / 679 kB transferred | 215 kB / 1,077 kB resources | Finish: 2.70 s

Fonte: Elaborado pelo autor

A conversão automática dos arquivos para o formato AVIF permitiu que ficassem bem leves, mesmo assim mantendo a qualidade visual, o qual é ilustrado na figura 70. Garantindo que a entrega de conteúdo, nesse caso imagens, seja sempre feito pelos servidores de CDN da Cloudflare, os quais são distribuidos regionalmente.

Figura 70 – Site após otimização das imagens mantém qualidade



Fonte: Elaborado pelo autor

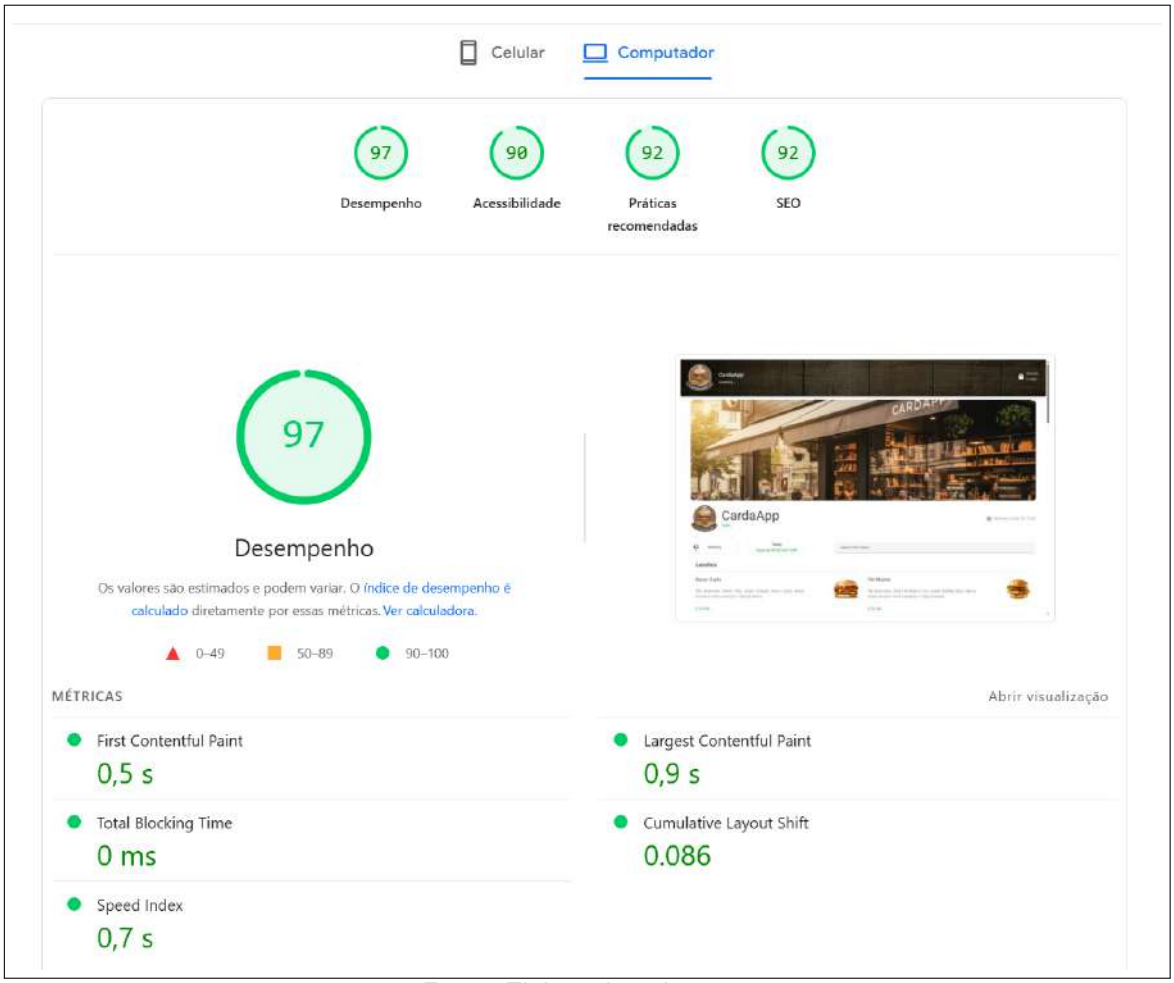
5.2.3 Resultado dos testes no Lighthouse - PageSpeed

Após a execução dos procedimentos descritos pela seção 5.2.2, a coleta de dados via PageSpeed Insights demonstra uma evolução significativa em todos os aspectos abordados, com destaque maior para a otimização de entrega de imagens.

Conforme ilustrado na figura 71, os indicadores de performance atingiram níveis bem superiores: o FCP foi reduzido para 0.5s, o LCP para 0.9s e o CLS ficou estabilizado em 0,086.

Estes resultados demonstram o impacto da utilização do Cloudflare Images e a importância da estratégia de *cacheamento* de arquivos estáticos para a redução de latência. Houve também uma melhora significativa nos índices de SEO e Acessibilidade. Ressaltando também que as outras métrica atingiram o 100%.

Figura 71 – PageSpeed pós otimização



Fonte: Elaborado pelo autor

5.2.4 Implementação do Cloudflare Worker para cache

O Worker nomeado como ”route-caching”, o qual foi utilizado para interceptar as requisiçoes http, antes que as mesmas alcancem no servidor de destino, como por exemplo o *frontend* fazendo requisição para o *backend* em busca de dados de cardápio. O mesmo usa a lógica de *fetch event*, onde a primeira resposta dessa rota possui um TTL, configurado para 30 dias.

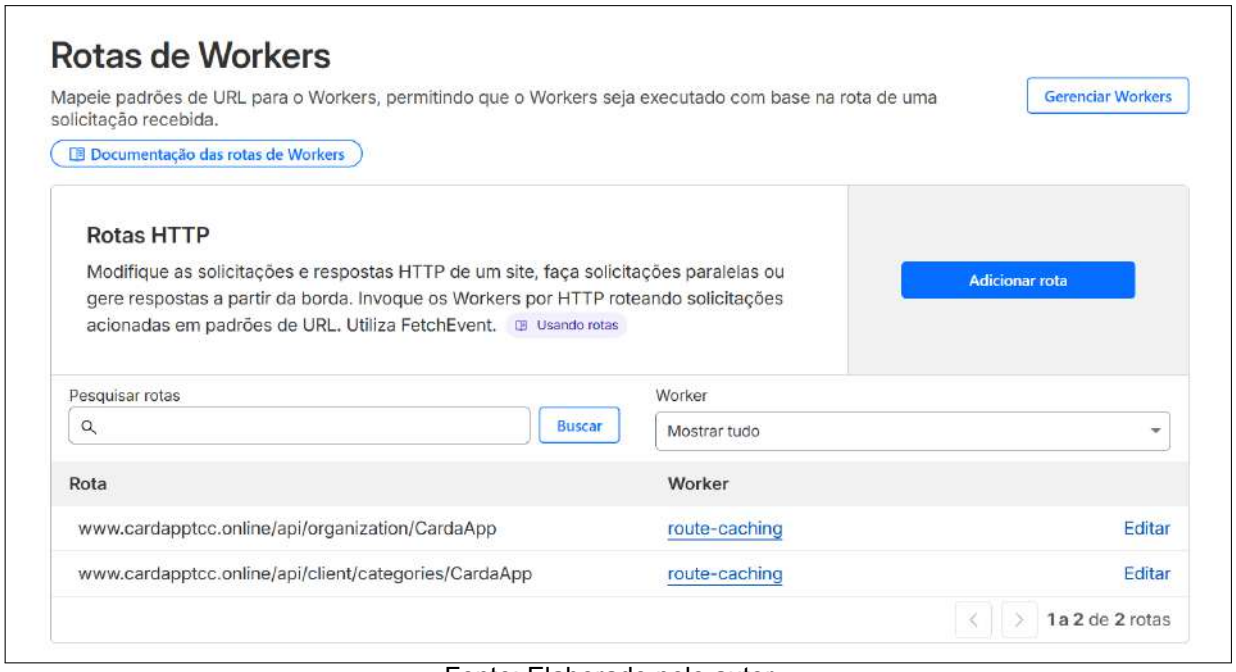
Foi necessário a utilização do ”Rota de Worker”, possibilitando adicionar este worker como *proxy* entre a requisição, o mesmo está interceptando duas rotas, sendo elas:

a) `api/organization/CardaApp`

b) `api/client/categories/CardaApp`

Esta estratégia foi utilizada para mapear rotas, as quais podem ser vistas na figura 72 que possuem um conteúdo mais estático e pesado, o que só faz sentido limpar o cache em caso de modificação. Isso também só foi possível por motivos de mesmo domínio, como a Cloudflare está cuidando do proxy tanto do *frontend* como do *backend*.

Figura 72 – Rotas Worker



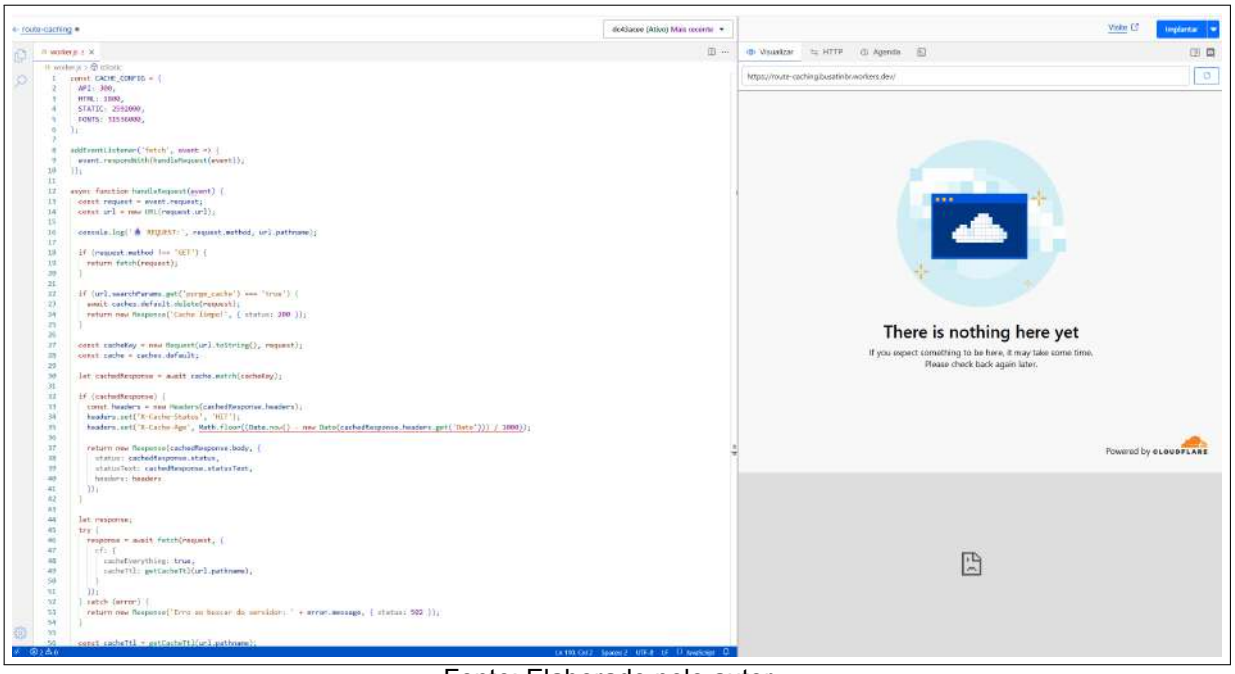
Fonte: Elaborado pelo autor

O *script* executado no Cloudflare Worker, ilustrado pela figura 73, implementa uma estratégia de *Edge Caching* utilizando o Cloudflare Worker, o qual atua como um intermediário interceptando requisições HTTP do tipo GET, configurado no *script*. Ele aplica TTL dinâmicos baseados na extensão do arquivo ou rota: APIs mantêm um cache curto de até 5 minutos, para garantir a atualização dos dados, enquanto para arquivos estáticos e fontes, dura por até um ano, para maximizar a performane.

Além da gestão do TTL, o código realiza o tratamento de segurança e da otimização dos cabeçalhos de resposta. Antes de salvar ou entregar o conteúdo, o Worker remove os cookies para impedir o *cache* de sessões privadas, adiciona algumas permissões de CORS para acesso entre origens. O mais importate: todo o processo de gravação no *cache* é feito de forma assíncrona, para não bloquear a resposta ao cliente.

O Worker prioriza a entrega do conteúdo já cacheado na borda (*Cache HIT*), apenas acessando o servidor de origem quando necessário.

Figura 73 – Código do Worker



Fonte: Elaborado pelo autor

Embora o cache seja utilizado para um conteúdo com pouca probabilidade de alteração, a Cloudflare permite invalidar esse cache por requisição post, podendo ser feita via servidor *backend*.

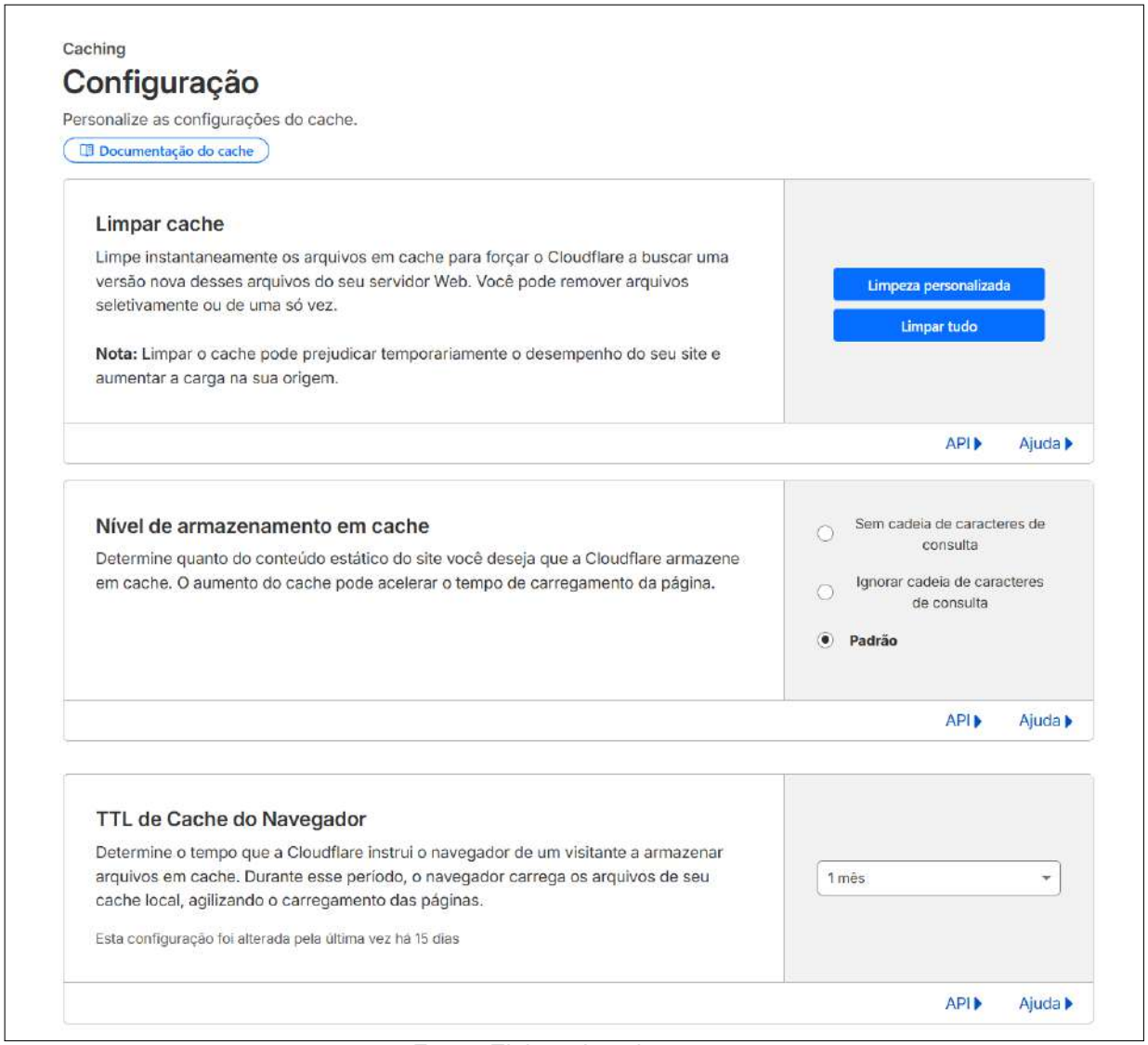
5.2.5 Cache genérico inteligente da aplicação pela Cloudflare

O cache genérico da Cloudflare garante que arquivos estáticos como JS, CSS e imagens sejam cacheados a nível de rede entre o usuário e o servidor, a configuração do mesmo é apresentada pela figura 74. Isso garante que em requisições subsequentes, a página não exija muito recurso da Cloudflare.

O sistema também possui funcionalidade para limpeza de cache tanto a nível de domínio completo quanto de forma personalizada, permitindo que seja filtrado especificamente. A configuração de TTL define o tempo que o conteúdo permanece em cache, sendo o padrão de um mês.

Algumas opções de cache também foram ativadas, que serão explicadas nas próximas subseções.

Figura 74 – Cache genérico Cloudflare

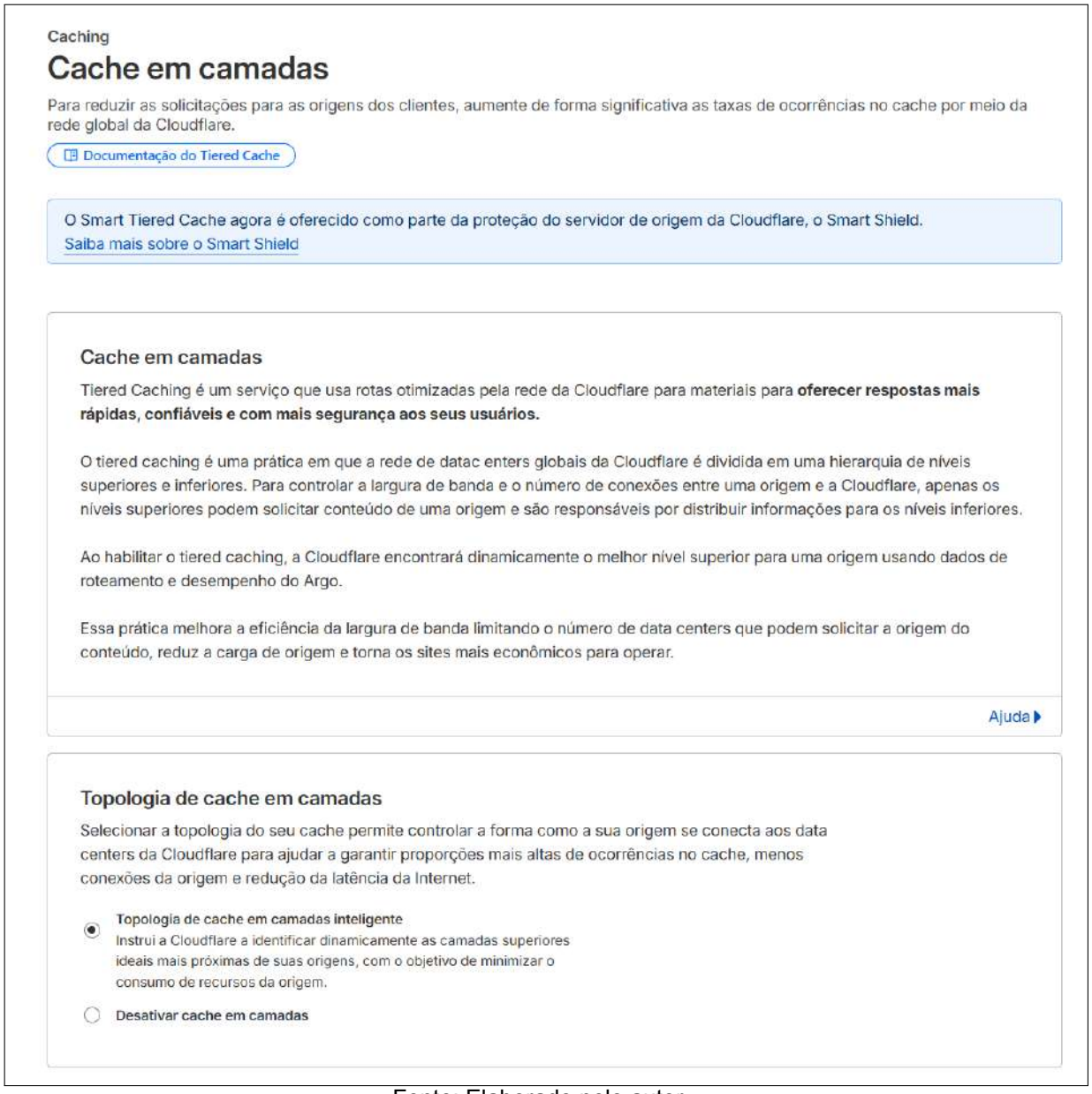


Fonte: Elaborado pelo autor

5.2.5.1 Cache em Camadas (Tiered Cache)

A Cloudflare implementa uma arquitetura de cache chamada *Tiered Cache*, que é uma arquitetura hierárquica o qual organiza os datacenters em diferentes níveis de prioridade, no qual as consultas são feitas de *datacenter* em *datacenter* e caso nenhum tenha o *cache* solicitado, será feita a requisição para o servidor de origem (CLOUDFLARE, 2024a). A configuração é ilustrada pela imagem 75.

Figura 75 – Cache em camada

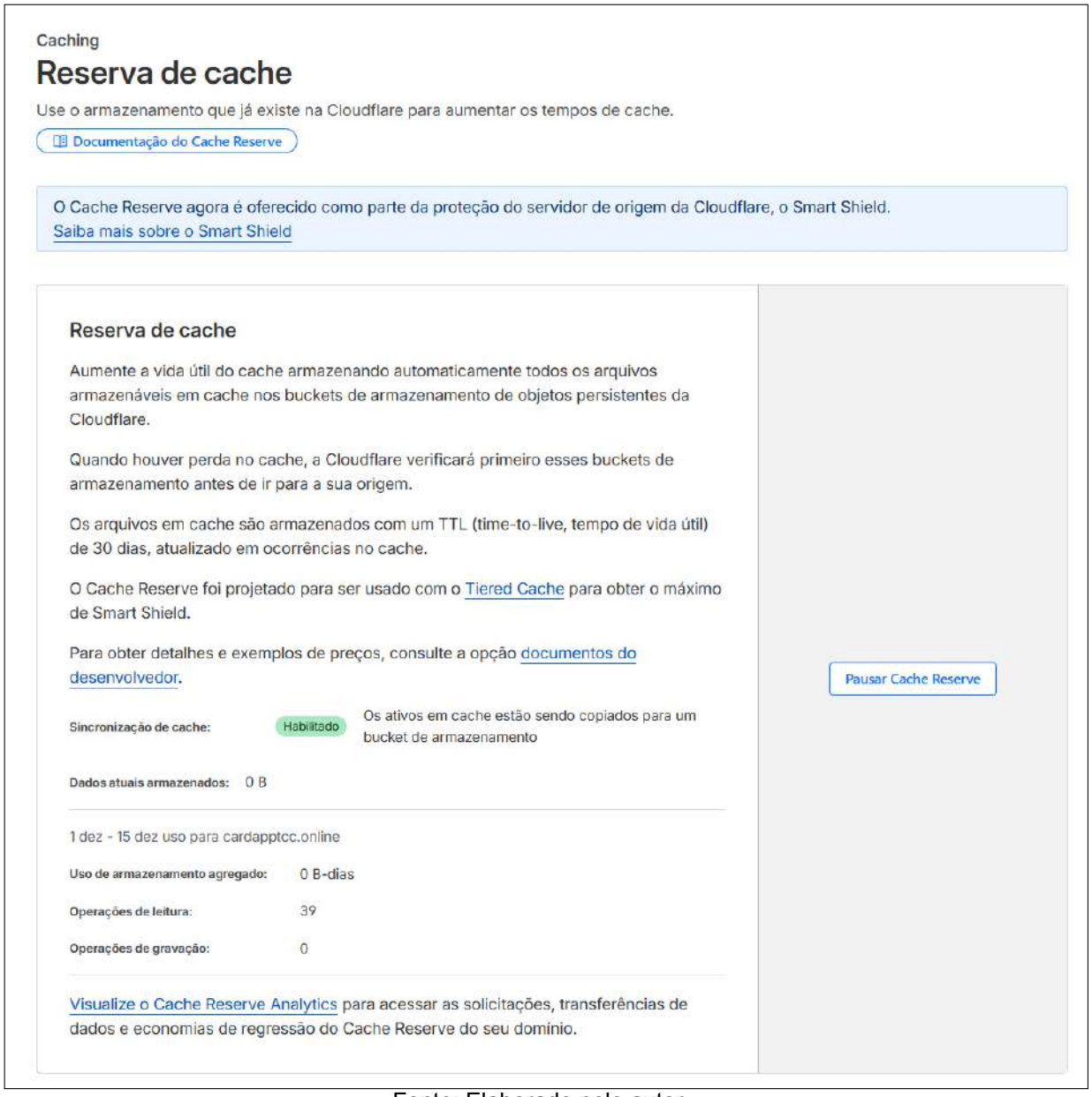


Fonte: Elaborado pelo autor

5.2.5.2 Cache Reserve

Já o cache reserve é uma camada adicional para persistência de cache, que utiliza também o serviço de armazenamento R2, podendo manter o contéudo por períodos prolongados (CLOUDFLARE, 2024a). A configuração é representada pela imagem 76.

Figura 76 – Cache reserva

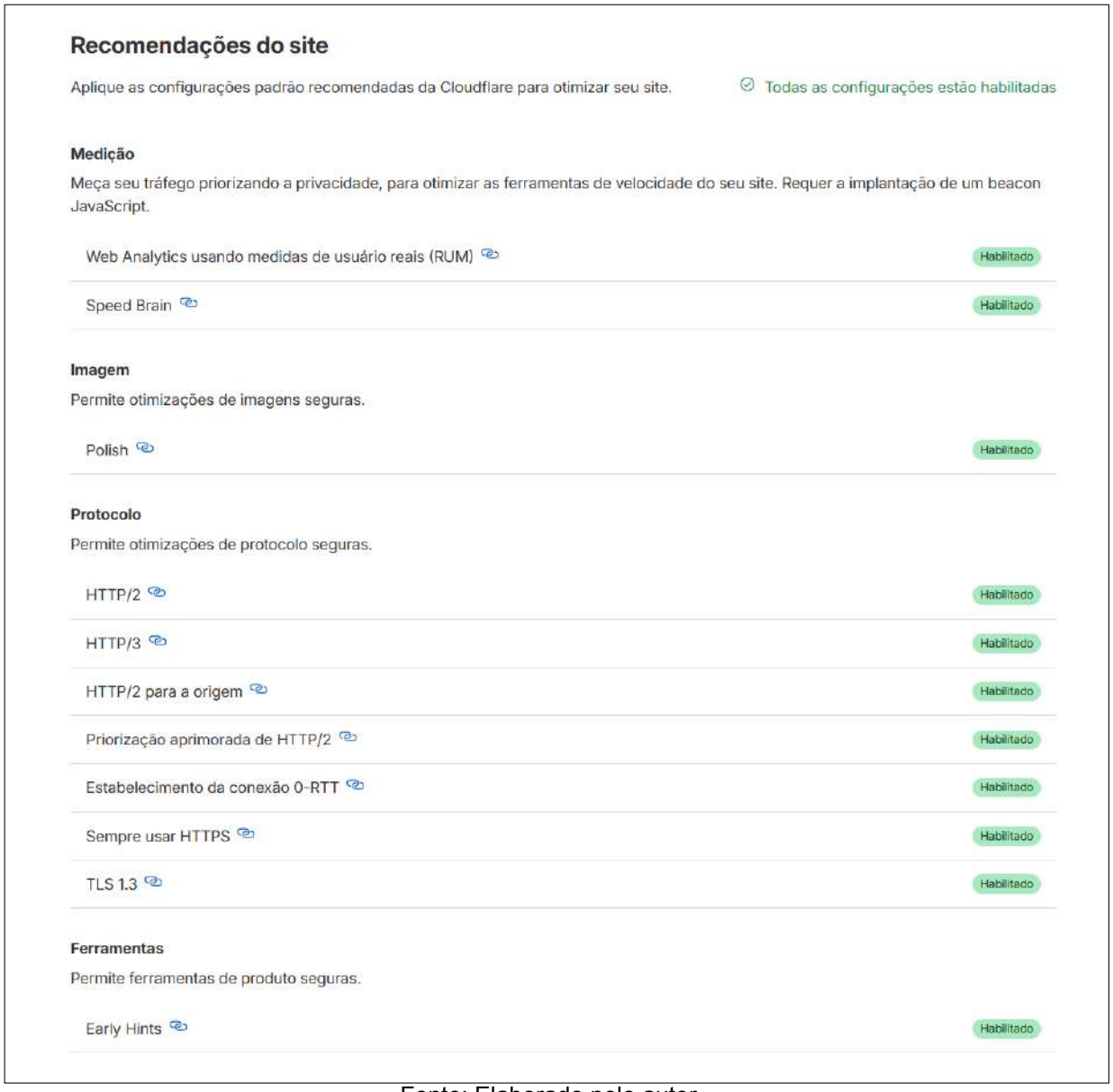


Fonte: Elaborado pelo autor

5.2.6 Configurações adicionais de desempenho

Algumas configurações adicionais foram aplicadas no painel da Cloudflare, de acordo com a figura 77, as quais tem como principal objetivo tratar a camada de transporte e de segurança da aplicação em questão, garantindo que tenha uma entrega de conteúdo mais protegida. Foram habilitados os protocolos de última geração, como o próprio HTTP/3 e o TLS 1.3, juntamente também com o *0-RTT Connection Establishment*, que reduz a latência em conexões criptografadas. Também conta com o aprimoramento do HTTP/2, forçando todas conexões a serem HTTPS (CLOUDFLARE, 2024a).

Figura 77 – Configurações adicionais de desempenho da Cloudflare

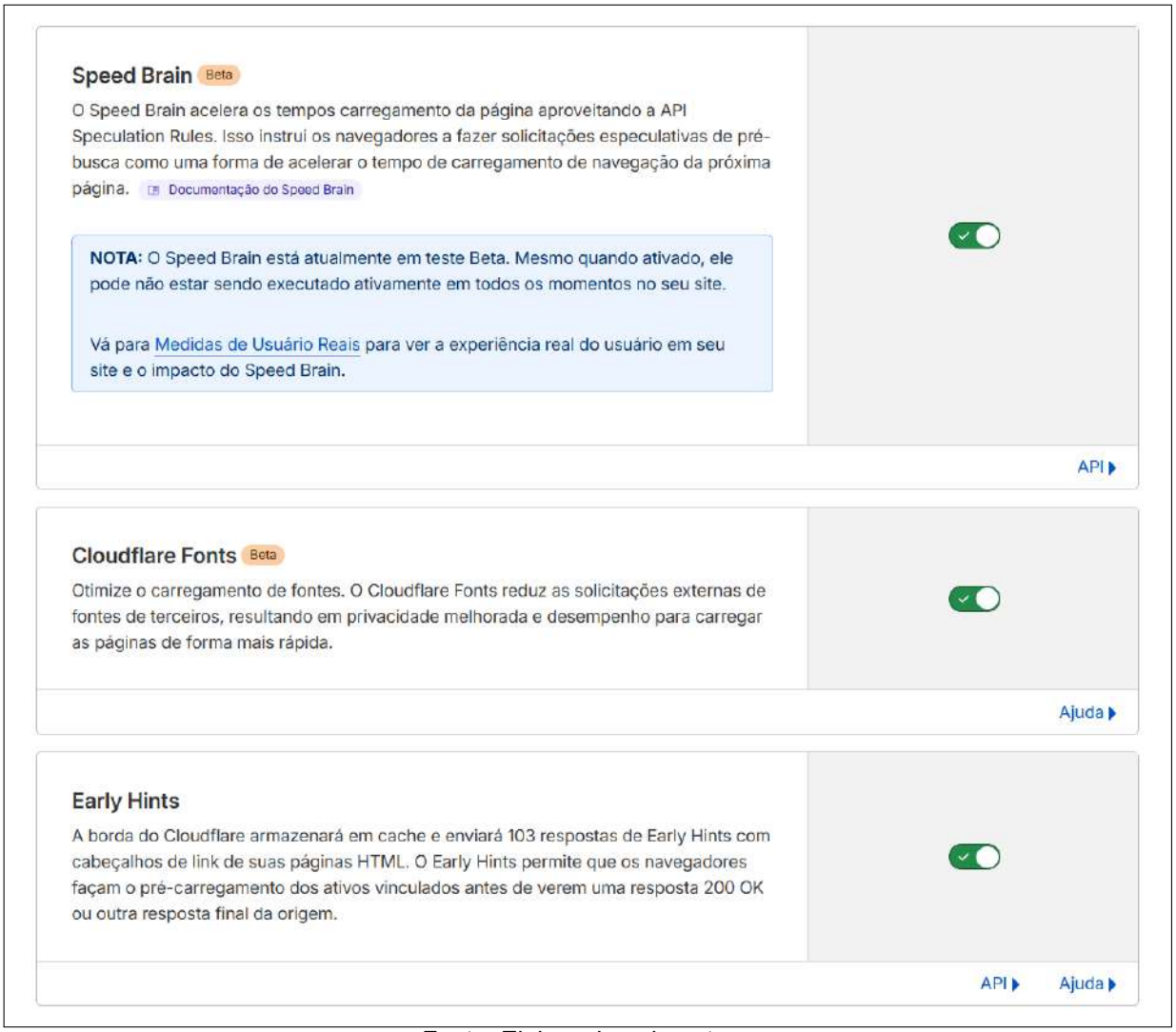


Fonte: Elaborado pelo autor

A plataforma também oferece mais funcionalidades avançadas para otimização de conteúdo, ilustradas pela figura 78, cujo objetivo é reduzir a latência percebida pelo próprio usuário, contando com estratégia de localização de conteúdo. O recurso *Speed Brain* utilizando o *Speculation Rules API* para antecipar e especular a próxima ação do usuário, realizando um carregamento antecipado especulado das páginas e tornando a navegação quase que instantânea (CLOUDFLARE, 2024a).

Além disso, também faz uso do Cloudflare Fonts, o qual aprimora a performance e a privacidade com o carregamento das tipografias da aplicação, eliminando as requisições externas a servidores de fontes (CLOUDFLARE, 2024a).

Figura 78 – Mais configurações da Cloudflare



Fonte: Elaborado pelo autor

5.2.7 Resultados dos Testes com Puppeteer - Pós Configuração

A execução deste teste seguiu a mesma configuração descrita na seção 5.1.2, com o objetivo de simular um cenário real de navegação com carga simultânea.

Sendo assim, mapeando as interações dos 200 usuários virtuais, permitindo o monitoramento pela instância EC2, o comportamento do Load Balancer e as métricas de Web Analytics. Os resultados de desempenho desse teste estão apresentadas no quadro 11.

A partir deste teste foi realizada a coleta de dados e métricas internas na plataforma da Cloudflare, AWS.

Quadro 11 – Resultados do teste de desempenho com Puppeteer - 200 usuários

Métrica	**Valor**	**Avaliação**
Resumo Geral		
Total de testes	200	—
Taxa de sucesso	100%	Todos os testes concluídos
Falhas	0	Nenhuma falha registrada
Tempo total	100s (1,7 min)	Aproximadamente 0,50s por usuário
Core Web Vitals (Médias)		
LCP (Largest Contentful Paint)	912ms	BOM - Dentro do ideal (<2500ms)
FCP (First Contentful Paint)	833ms	BOM - Dentro do ideal (<1800ms)
CLS (Cumulative Layout Shift)	0,000	BOM - Dentro do ideal (<0,1)
Tempo de navegação médio	1588ms	Tempo de carregamento inicial
Análise de Performance		
Estabilidade visual	Excelente	CLS zerado indica estabilidade total
Velocidade de carregamento	Excelente	LCP e FCP otimizados (<1s)
Confiabilidade	Alta	100% de taxa de sucesso

Fonte: Elaborado pelo autor.

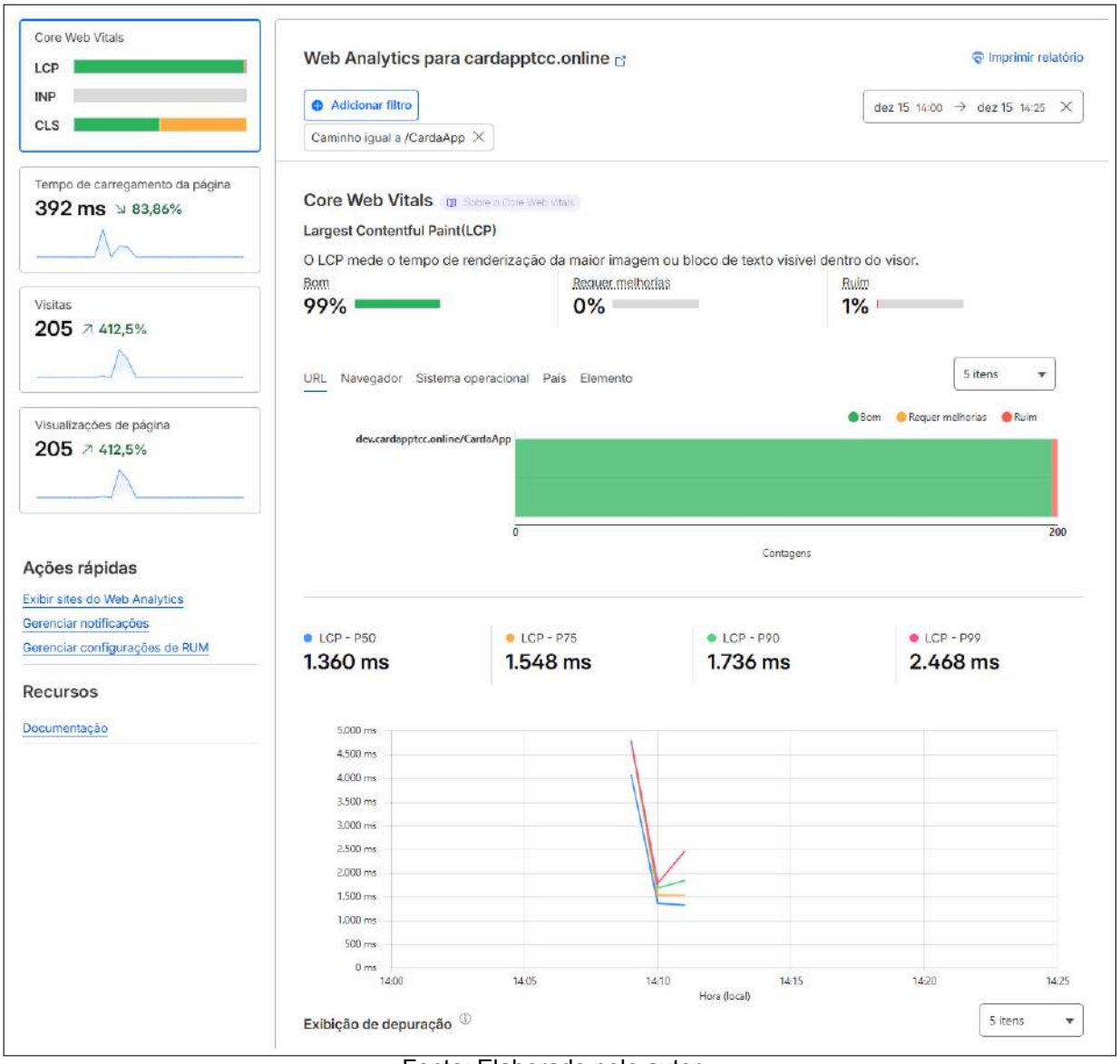
O teste realizado pelo Puppeteer mostra um desempenho bem superior da aplicação sob a mesma carga, destaca-se primeiramente a confiabilidade do sistema, que no caso alcançou uma taxa de sucesso de 100%, sem qualquer tipo de falha. O tempo de processamento pra cada requisição ficou em uma média de 0,5 segundos por interação.

Em respeito as métricas de experiência de usuário, os resultados mostram uma média bem superior comparados com as métricas do Google, como o FCP de 833ms e o LCP com 912ms. Contando também com o CLS de 0,000.

5.2.7.1 Resultado Puppeteer: Análise dos Dados do Google Analytics - Core Web Vitals

Com as otimizações de cache e infraestrutura, a coleta do Web Analytics reflete uma transformação significativa na experiência do usuário. Conforme demonstrado na figura 79, os indicadores do Core Web Vitals alcançaram níveis de excelência, completamente oposto ao cenário crítico anterior.

Figura 79 – Web Analytics pós otimização



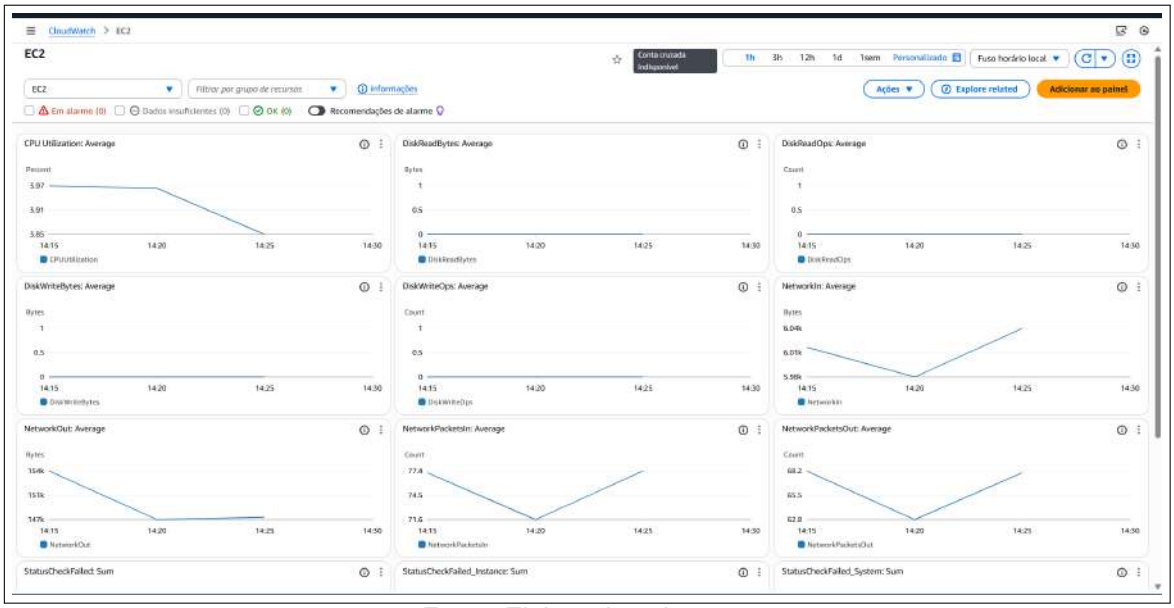
Fonte: Elaborado pelo autor

Mantendo-se o mesmo volume de tráfego, o LCP, que antes apresentava falhas em 99%, agora registra o total contrário. Os tempos de renderização sofreram uma redução drástica, estabilizando-se. O gráfico confirma estabilidade, o qual não mostra mais os picos de alta latência como anteriormente.

5.2.7.2 Resultado Puppeteer: Análise EC2

O monitoramento novamente foi feito via AWS CloudWatch, durante o cenário otimizado, o qual demonstra um comportamento de baixa utilização de recursos. Conforme representado pela figura 80. A EC2 chegou até a operar em período de ociosidade durante a duração do teste.

Figura 80 – Resultado: EC2 pós otimização



Fonte: Elaborado pelo autor

A utilização de CPU permaneceu reduzida, oscilando abaixo dos 4% durante todo o período. Da mesma forma que o tráfego de rede registrou um volume mínimo, abaixo de 6KB.

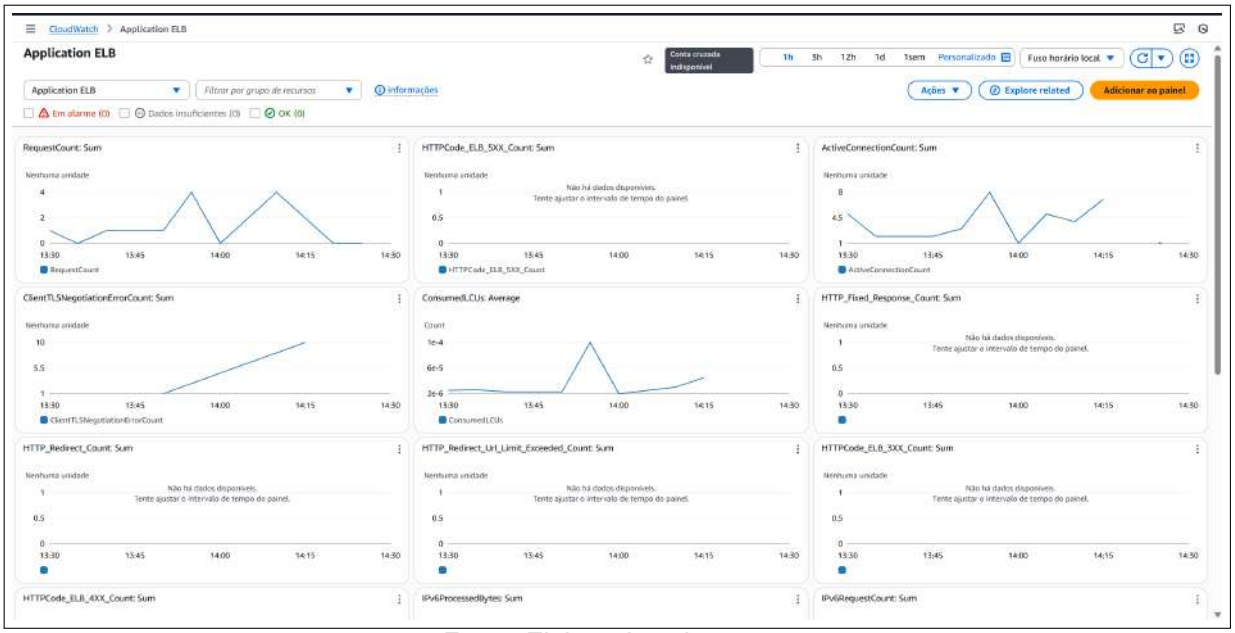
5.2.7.3 Resultado Puppeteer: Análise Load Balancer

O monitoramento do ELB via Amazon CloudWatch mostra a eficiência da camada de *cache* implementada no Worker. Conforme demonstrado pela figura 81, o comportamento das métricas mostra diretamente o impacto do quanto o cache fez uma retenção inteligente do tráfego, isolando o ambiente de *backend* do volume massivo de requisições exageradas e repetitivas.

A análise da métrica *RequestCount* ilustra que o tráfego de 120 mil requisições processado pela Cloudflare foi quase inteiramente anulado pela CDN, não chegando a impactar a camada do balanceamento da AWS. O gráfico registra uma atividade mínima, com picos de apenas 4 requisições, volume exclusivo das rotinas automáticas de *health check* da aplicação.

Como consequência disso, a utilização dos recursos no servidor de origem permaneceram em níveis baixos. O número de conexões ativas apresentou no máximo até 8 conexões simultâneas.

Figura 81 – ELB pós otimização

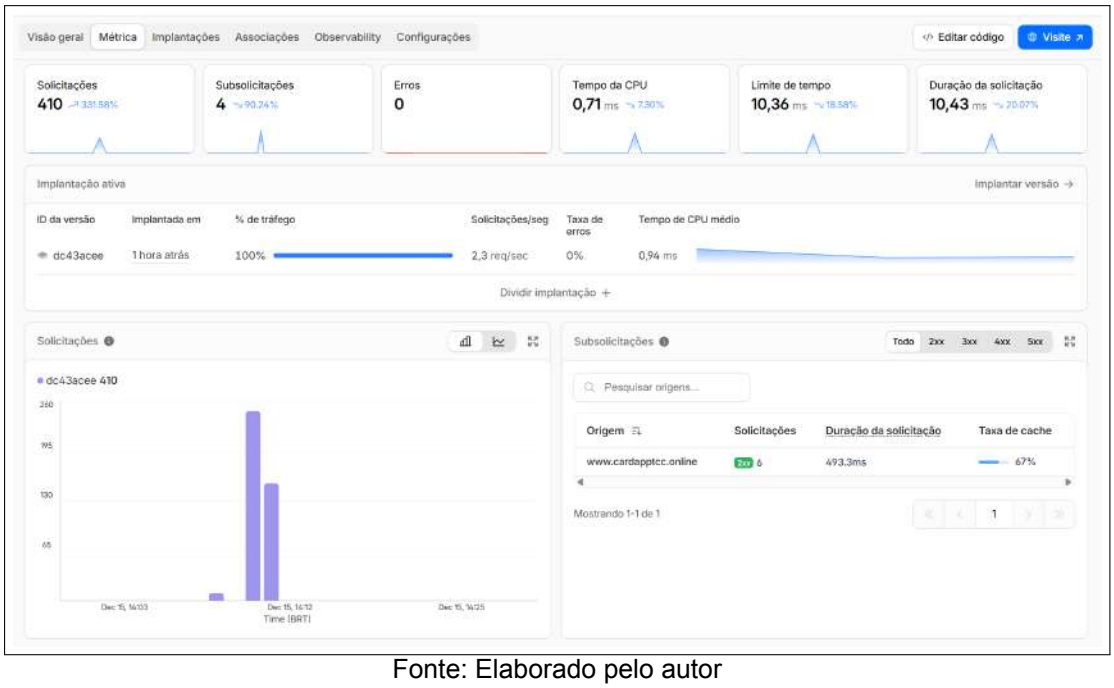


Fonte: Elaborado pelo autor

5.2.7.4 Resultado Puppeteer: Análise Worker

As métricas coletadas do Worker durante a execução do teste com o Puppeteer, confirma a eficiência do *script* na manipulação de tráfego. Conforme ilustrado pela página de métricas na figura 82, houve um pico de requisições relacionadas ao período do teste, totalizando 400 solicitações.

Figura 82 – Métricas Worker

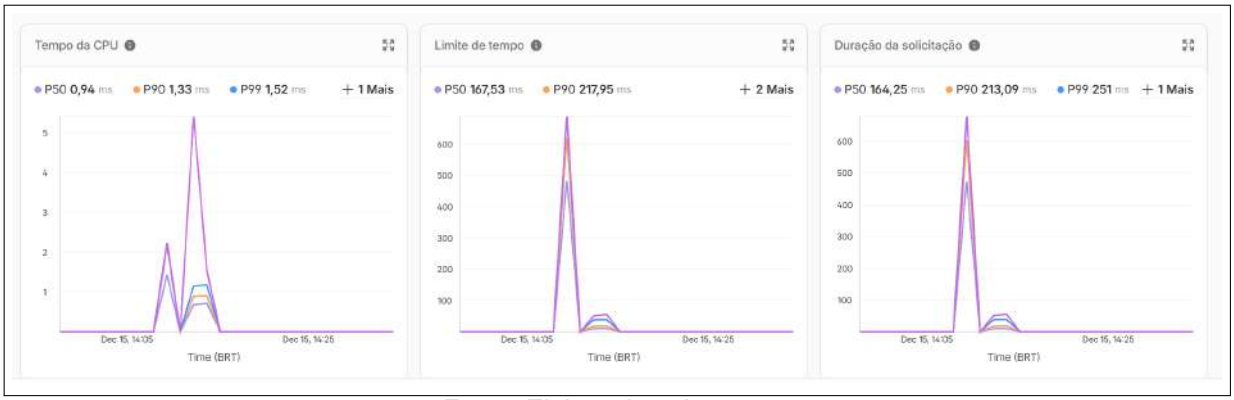


Fonte: Elaborado pelo autor

Considerando que houve um cenário de teste com 200 usuários, onde cada sessão disparou chamadas para duas rotas que são gerenciadas pelo Worker, o total processado de 400 requisições se alinha ao teste.

Em termos de desempenho de máquina, detalhado na figura 83, os gráficos apresentam no eixo horizontal a janela de tempo de execução do teste e no vertical a latência em milissegundos. O indicador de Tempo da CPU registrou uma mediana (P50) de apenas 0.94ms, o que indica que o *script* é extremamente leve e não causa impacto nas requisições.

Figura 83 – Dados de acesso Worker



Fonte: Elaborado pelo autor

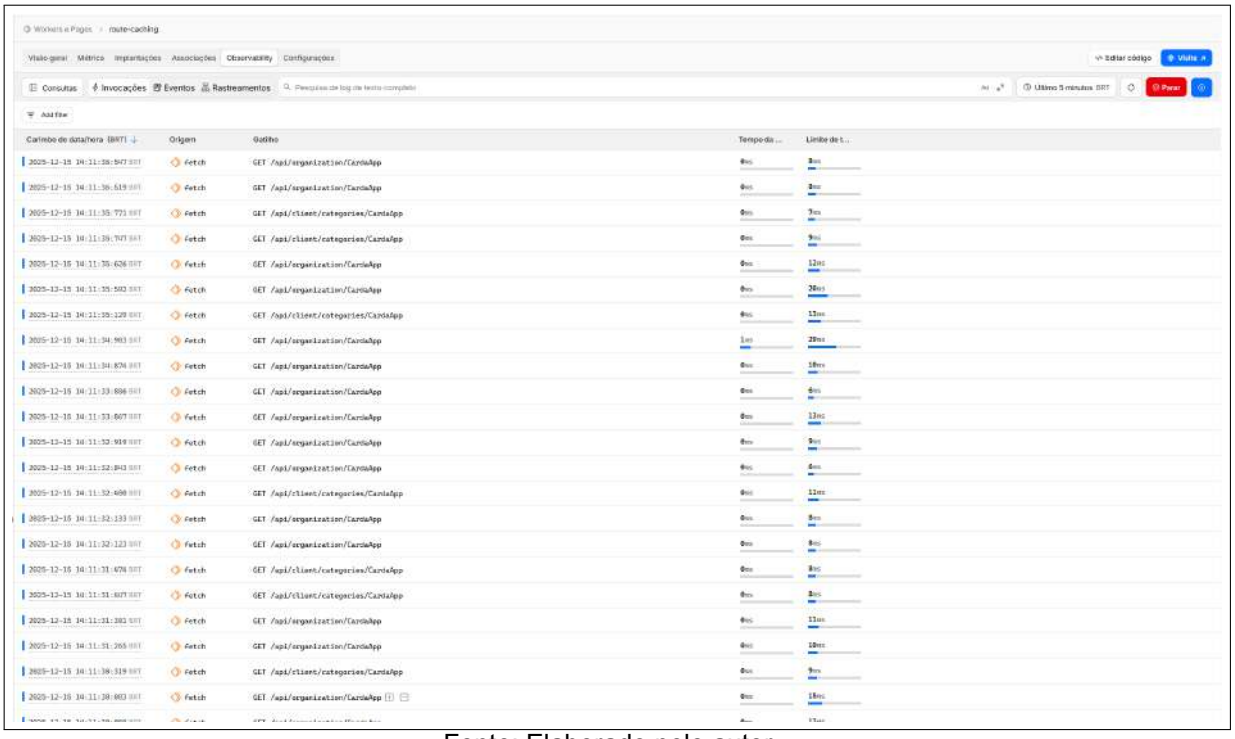
Já as métricas de Limite de tempo e Duração da solicitação apresentaram uma mediana de 167.53ms e 164.25 respectivamente. Diferente do tempo de CPU, esses valores representam o tempo total de vida do Worker, englobando todo o ciclo de vida do mesmo.

A consistência desses tempos, mesmo durante o pico de tráfego que houve, mostra que a comunicação entre a borda e o *backend* se manteve estável.

5.2.7.4.1 Monitoramento em Tempo Real (Live Tail)

Para realizar uma auditoria no comportamento do cache, foi utilizado a ferramenta *Live Tail*, que permite inspecionar os *logs* em tempo real. A visualização do fluxo de requisições, apresentada na figura 84, demonstra todas as solicitações que são redirecionadas para o Worker durante o teste, impedindo o acesso direto à API.

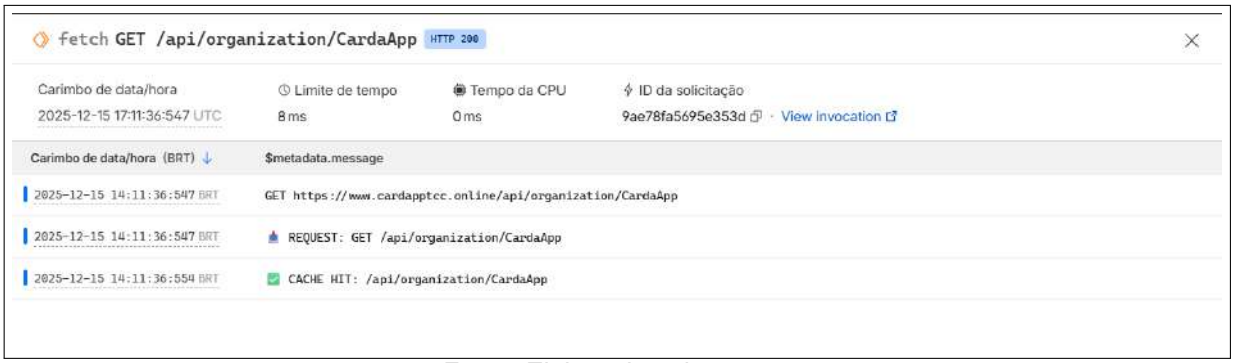
Figura 84 – Worker Live Page



Fonte: Elaborado pelo autor

A eficácia da lógica de *cache* pode ser vista na figura 85. O registro do *log* mostra a mensagem *CACHE HIT*, com um tempo de CPU de 0ms. Isso prova que o Worker interceptou a requisição e entregou o conteúdo em *cache*, sem acionar o servidor de origem.

Figura 85 – Worker Cache Hit



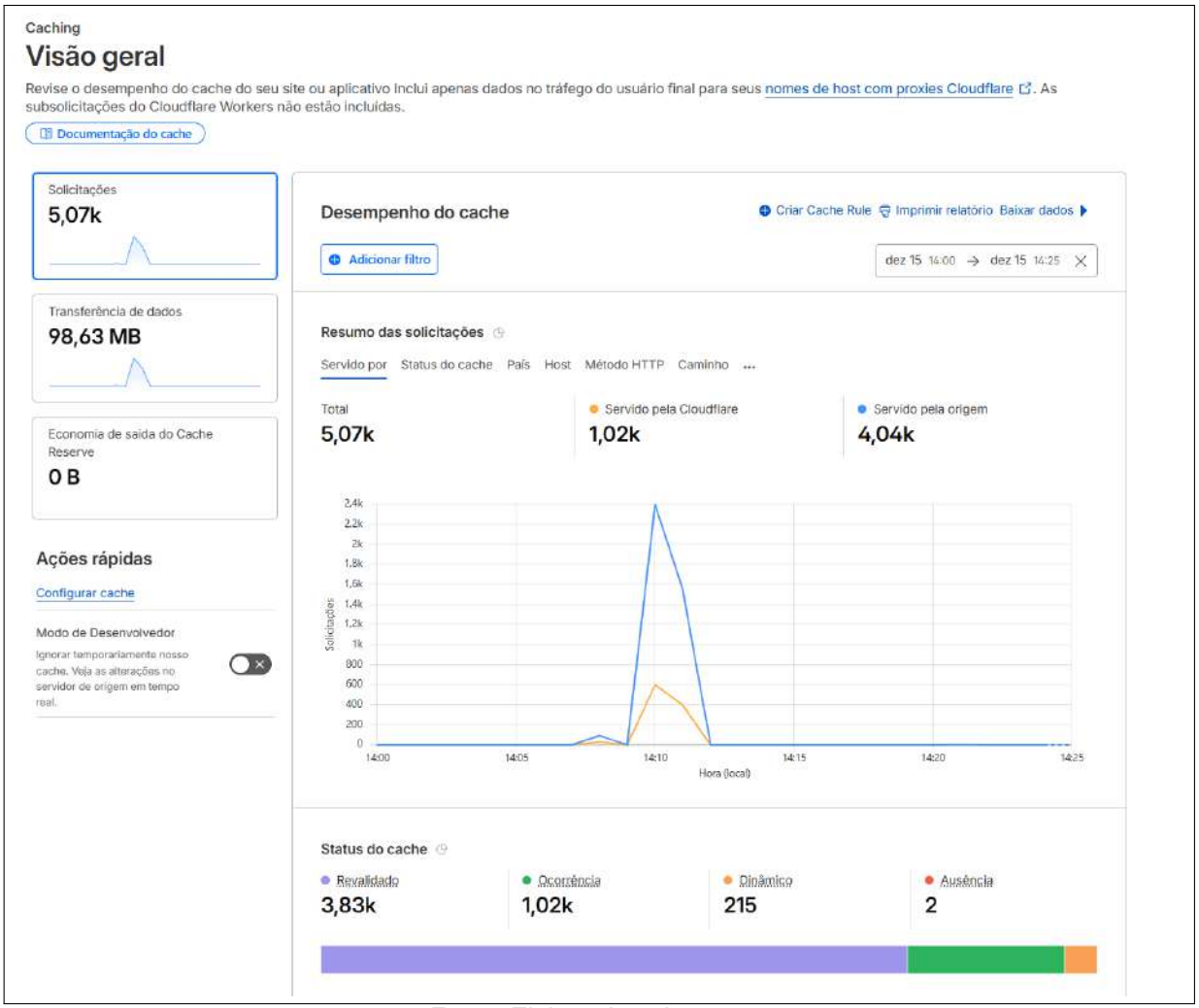
Fonte: Elaborado pelo autor

5.2.7.5 Resultado Puppeteer: Cache Analytics

A análise do tráfego de *cache* durante a execução do teste, de acordo com a figura 86, registra um total de 5.07 mil requisições, gerando um tráfego de 98.83MB. A distribuição do gráfico indica que 1.02 requisições foram atendida pela borda da

Cloudflare, no caso, o *Cache Hit*, enquanto 4.04 mil requisições foram servidas pela própria origem.

Figura 86 – Cache Analytics



Fonte: Elaborado pelo autor

O detalhe dos *status* de *cache* demonstra que a maioria das requisições à origem, totalizando os 3.83 mil, se enquadram na categoria Revalidado. Isso significa que, mesmo apresentando comunicação direta com o *backend*, que no caso foi para validação do conteúdo, não houve transferência desnecessária de dados mais pesados, resultando em uma economia maior de banda. As requisições dinâmicas somaram apenas 215 interações, e o índice de *Cache Miss* foi pequeno, o que isolou bastante o servidor de origem de arquivos estáticos.

6 Resultados Alcançados e Considerações Finais

Com os resultados obtidos neste trabalho, foi possível analisar o impacto significativo direto na implementação dos serviços da Cloudflare. A análise comparativa entre os cenários antes e depois da otimização revelam melhorias em todas as métricas, mostrando a eficácia da estratégia de *cacheamento* na borda, otimização de imagens e configuração adequada da infraestrutura de entrega do conteúdo. Em relação aos objetivos do trabalho, temos a seguinte comparação final entre os resultados:

- **Objetivo 1 - Infraestrutura da aplicação:** A implementação do WAF com as regras básicas e personalizadas garantiu a proteção contra injeção SQL e ataques XSS. A proteção contra anti-DDoS também foi ativada, porém, não foi possível coletar métricas suficientes por falta de ferramentas para executar o ataque de forma eficaz.
- **Objetivo 2 - Otimização da aplicação web:** A configuração de cache reduziu significativamente a carga no *backend*. As rotas de API mais acessadas foram interceptadas e cacheadas na borda da rede. Além isso, o *Cloudflare Images* teve bastante impacto no desempenho da aplicação, resultando em uma redução de aproximadamente 90% no tamanho dos arquivos de imagem.
- **Objetivo 3 - Melhoria na experiência do usuário:** Os resultados demonstraram melhorias significativas nas métricas do Core Web Vitals. O LCP foi reduzido de 8.500ms para 900ms, representando uma melhoria de 89%. O FCP apresentou uma redução de 75%, passando de 3.284ms para 833ms, já o CLS atingiu valores praticamente nulos, indicando uma estabilidade visual quase perfeita. Essas questões garantiram a melhoria no SEO da aplicação.

6.1 Análise dos Resultados Obtidos

Os resultados obtidos neste trabalho demonstram o impacto significativo na implementação dos serviços da Cloudflare, são eles:

Teste de Carga do Loader.io

- **Redução de 30% no tempo médio de resposta:** de 43ms para 30ms;
- **Redução de 63% no tempo máximo de resposta:** de 771ms para 284ms;
- **Taxa de sucesso elevada:** de 99,26% para 100%;
- **Eliminação total de falhas:** de 444 para 0 requisições falhadas;

- **Confiabilidade:** Eliminação de *timeouts* e erros de rede.

Core Web Vitals (Realizado com Puppeteer

- **LCP:** Redução de 72% (de 3.256ms para 912ms);
- **FCP:** Redução de 75% (de 3.284ms para 833ms);
- **CLS:** Melhoria de 100% (de 0,002 para 0,000).

Métricas do PageSpeed Insights

- **LCP:** Redução de 89% (de 8.500ms para 900ms);
- **FCP:** Melhoria de 0,6s para 0,5s;
- ***Score* de *Performance*:** Elevação significativa com todas métricas atingindo status ”Bom”.

Impacto na Infraestrutura AWS

- **Utilização de CPU na EC2:** Redução para menos de 4%;
- **Tráfego de rede na EC2:** Redução de 3,02MB para menos de 6KB;
- **Requisições no Load Balancer:** Redução de 1.200+ para apenas 4 (*health checks*);
- **Conexões ativas no ELB:** Redução de 755 para 8.

Eficiência do Cache

- ***Cache Hit Rate*:** 1,02k requisições atendidas pela borda;
- **Revalidação:** 3,83k requisições revalidadas sem transferência de dados pesados;
- **Processamento Dinâmico:** Apenas 215 requisições processadas diretamente no servidor;
- **Economia de Banda:** Estimada em aproximadamente 95%.

6.2 Considerações finais

Com o desenvolvimento do projeto em questão, foi possível aplicar e consolidar conhecimentos adquiridos no curso de Sistemas de Informação. Os quais foram: noções de modelagem de sistemas, gerenciamento e levantamento de requisitos, conhecimentos gerais em programação tanto como *frontend* como *backend*, além do conhecimento para gerenciamento de banco de dados relacional.

Além de utilizar os conhecimentos da graduação, também foram exploradas novas fontes e o estudo de tecnologias mais atuais do mercado. Nesse contexto, entra a computação em nuvem, uma vez que se tornou uma das áreas mais importantes e que está em constante evolução.

Também foi possível utilizar práticas em plataformas modernas e mais utilizadas do mercado, como a AWS e a Cloudflare. O foco principal não foi o desenvolvimento do *software* em si, mas sim a investigação de como os serviços de nuvem, em específico a Cloudflare, podem contribuir para a otimização e desempenho da aplicação *web*, tanto quanto a forma de distribuição.

Durante o desenvolvimento, foram reforçados conhecimentos práticos em: configuração de instâncias EC2 e grupos de segurança, emissão e aplicação de certificados SSL/TLS via AWS Certificate Manager, configuração de Load Balancer para distribuição de tráfego e aplicação do certificado, gerenciamento de DNS e domínios, implementação de *pipelines* CI/CD com GitHub Actions, containerização com Docker, estratégias de *cache* e CDN, desenvolvimento de Cloudflare Worker, otimização de imagens e monitoramento de métricas de desempenho.

6.2.1 Trabalhos futuros

Mesmo com os resultados positivos obtidos, este estudo apresentou algumas limitações que devem ser consideradas para melhorias futuras:

- **Limitação do ambiente:** Os testes foram executados em uma instância EC2 do tipo t2.micro, que contém apenas 1vCPU e 1GB de memória RAM. Foi possível demonstrar alguns conceitos, mas essa configuração não representa cenários de produção em alta escala.
- **Aplicação de teste simples:** O estudo em questão utilizou uma plataforma mais simplificada, que possui complexidade baixa. Aplicações mais complexas que exigem filas, mensageria, processamento assíncrono, *cache* distribuído como o Redis e comunicação em tempo real, os quais podem mostrar comportamentos e gargalos de performance distintos dos observados neste trabalho.

- **Carga de teste pequena:** Os testes foram realizados com 200 a 1000 usuários simultâneos, porém, cenários de alta escala com milhares de usuários não foram avaliados, o que poderia revelar outros comportamentos.
- **Ausência de testes de segurança reais:** Não foi possível realizar testes de segurança reais, embora o WAF tenha sido configurado e testado de forma simplificada, não foi possível a realização de testes de penetração ou simulação DDoS, SQL Injection ou XSS, por falta do conhecimento e equipamento necessário.
- **Testes na mesma região geográfica:** Os testes foram realizados em uma única região geográfica. O comportamento do CDN e a latência podem variar dependendo dos usuários e das regiões.

Com base nas limitações citadas acima e nos resultados obtidos, seria interessante o direcionamento pra trabalhos futuros: Realizar um estudo comparativo detalhado dos custos totais de operação, considerando a própria AWS pura com AWS integrada ao Cloudflare, incluindo todos serviços adicionais.

Realizar testes de segurança mais práticos, conduzindo testes de penetração como ferramentas OWASP para validação de ataques de SQL Injection, XSS e DDoS.

Estudo comparativo entre ferramentas da AWS em comparação com as da Cloudflare, como: Cloudflare e Cloudfront, Workers e Lambda, entre outros.

O desenvolvimento dessa pesquisa foi um grande desafio, contando não apenas com o custo financeiro, mas também com a complexidade da organização pessoal e da gestão de tempo. Porém, o investimento foi necessário diante do objetivo maior de compartilhar conhecimento prático do mercado atual com outros estudantes. A intenção central foi demonstrar como o uso de ferramentas de ponta e em nuvem pode auxiliar, otimizar e elevar todo o nível do desenvolvimento. Além disso, o projeto também proporcionou aprimorar os conhecimentos em questão profissional.

Referências

AKAMAI. *Akamai: Security, Cloud, and Content Delivery Solutions*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.akamai.com/pt>. 19, 20

AL-DULAIMI, A. A. et al. Websites optimization metrics: A systematic literature review. *Prospective Researches Journal*, v. 24, n. 2, p. 42–53, April 2024. Acesso em: 9 dez. 2024. Disponível em: <https://jpr.hu.edu.iq/index.php/Prospective_Researches_Journal/article/view/421>. 17

ALURA. *Cloud: o que é, História e Guia da computação em nuvem*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.alura.com.br/artigos/cloud>. 17

AWS. *Amazon Web Services (AWS) - Cloud Computing Services*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://aws.amazon.com/>. 18, 19, 20, 28

AWS. *O que é uma API RESTful?* 2025. Acesso em: 27 nov. 2025. Disponível em: <https://aws.amazon.com/pt/what-is/restful-api/>. 30

AXIOS. *Axios - Promise based HTTP client for the browser and Node.js*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://axios-http.com/>. 23

BAJRAMI, E.; IDRIZI, F.; RUSHITI, A. Enhancing website speed and security with cloudflare cdn: a case study on wordpress websites. *Journal of Natural Sciences and Mathematics of UT*, v. 9, n. 17–18, p. 294–304, October 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.researchgate.net/publication/384807865_ENHANCING_WEBSITE_SPEED_AND_SECURITY_WITH_CLOUDFLARE_CDN_A_CASE_STUDY_ON_WORDPRESS_WEBSITES>. 16

BÖHM, M. et al. *Cloud computing and computing evolution*. 2011. Acesso em: 9 dez. 2024. Disponível em: <https://www.researchgate.net/publication/268011245_Cloud_Computing_and_Computing_Evolution>. 16

CLOUDFLARE. *2023 Impact Report*. 2023. Acesso em: 9 dez. 2024. Disponível em: <https://cf-assets.www.cloudflare.com/slt3lc6tev37/3Z7xOV53lGEIAwCqw5SncT/69eaca7bd5d2395ee0274b15e7854dd6/2023_Impact_Report.pdf>. 14, 24, 25, 26, 27, 28, 63

CLOUDFLARE. *The Connectivity Cloud: a way to take back IT and security control*. 2023. Acesso em: 21 nov. 2024. Disponível em: <https://cf-assets.www.cloudflare.com/slt3lc6tev37/1v0V2tXAJnhYN4momCWDEx/1220ab3c3ad8d4c912fe55d45008ed7d/eBook_The-connectivity-cloud-A-way-to-take-back-IT-and-security-control_Brazilian_Portuguese_20230929.pdf>. 14, 20

CLOUDFLARE. *Cloudflare Developers: Documentation, Guides, and API Reference*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://developers.cloudflare.com/>. 93, 94, 95, 96

CLOUDFLARE. *Wingie Enuygun & Cloudflare | Customer Story | Cloudflare*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.cloudflare.com/pt-br/case-studies/wingie-enuygun/>. 14

DEVMEDIA. *Conhecendo o modelo arquitetural REST*. 2013. Acesso em: 13 jan. 2026. Disponível em: <https://www.devmedia.com.br/conhecendo-o-modelo-arquitetural-rest/28052>. 31

DEVMEDIA. *Introdução ao padrão MVC*. 2013. Acesso em: 27 nov. 2025. Disponível em: <https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308>. 42

Docker Inc. *Docker Overview*. 2024. Acesso em: 16 abr. 2025. Disponível em: <https://docs.docker.com/get-started/docker-overview/>. 60

FILAMENT. *Laravel Admin Panel*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://filamentphp.com/use-cases/admin-panel>. 23

GICELLE. *Akamai vs Cloudflare: 2024 CDN Comparison on Price, Performance & Security*. 2024. Acesso em: 21 out. 2024. Disponível em: <https://www.sohofiy.com/akamai-vs-cloudflare-compare/>. 22

GITHUB. *Sobre o Git*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://docs.github.com/pt/get-started/using-git/about-git>. 28, 55, 56

GOOGLE. *SEO Starter Guide*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://developers.google.com/search/docs/fundamentals/seo-starter-guide?hl=pt>. 29, 30

GUPTA, B.; MITTAL, P.; MUFTI, T. A review on amazon web service (aws), microsoft azure & google cloud platform (gcp) services. *EAI Endorsed Transactions on Scalable Information Systems*, 2024. Acesso em: 9 dez. 2024. Disponível em: <https://eudl.eu/doi/10.4108/eai.27-2-2020.2303255>. 13

HASSAN, O. F. et al. The impact of cloud computing in promoting economic growth through smes in the united states. *International Journal of Computer Science and Information Technology*, v. 16, n. 2, p. 11–23, April 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.researchgate.net/publication/380330462_The_Impact_of_Cloud_Computing_in_Promoting_Economic_Growth_through_SMEs_in_the_United_States>. 14

HEUSER, C. A. *Projeto de Banco de Dados*. [S.l.]: Comunicação Pessoal do Autor, 1998. A publicação comercial deste texto está planejada. 40

HOSTINGER. *DDoS: o que é, Como funciona e Como se Proteger de Ataques Maliciosos na Internet*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-ddos-e-como-se-proteger-de-ataques>. 18

IBM. *Cloud Computing*. [S.l.], 2007. Acesso em: 9 dez. 2024. Disponível em: <https://www.ibm.com/developerworks/websphere/zones/hipods/>. 16, 37

JOHN, J. et al. Detecting and preventing of ddos attack in cloud computing environment based on hybrid technique (cloudflare and waf). *Newport International Journal of Engineering and Physical Sciences*, v. 3, n. 3, p. 28–40, November 2023. Acesso em:

9 dez. 2024. Disponível em: <https://www.researchgate.net/publication/375751095_Detecting_and_Preventing_of_DDoS_Attack_in_Cloud_Computing_Environment_Based_on_Hybrid_Technique_Cloudflare_and_WAF>. 18

LARAVEL. *Laravel - The PHP Framework for Web Artisans*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://laravel.com/>. 23, 45, 47

LIGHTHOUSE. *Google Chrome Developer Tools*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://developer.chrome.com/docs/lighthouse/overview?hl=pt-br>. 23, 24

LOADER. *Loader.io*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://loader.io/>. 28, 29

MOLINA-RíOS, J.; PEDREIRA-SOUTO, N. Comparison of development methodologies in web applications. *Information and Software Technology*, v. 119, p. 106238, November 2019. Acesso em: 9 dez. 2024. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0950584919302551>. 13

MOZILLA. *CSS Documentation*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. 23

MYSQL. *MySQL - The World's Most Popular Open Source Database*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.mysql.com/>. 23

NADEEM, M. et al. Preventing cloud network from spamming attacks using cloudflare and knn. *Computers, Materials & Continua*, v. 74, n. 2, p. 2641–2659, October 2022. Acesso em: 9 dez. 2024. Disponível em: <https://www.sciencedirect.com/org/science/article/pii/S1546221822003976>. 13

PHP. *Hypertext Preprocessor*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://www.php.net/manual/pt_BR/faq.general.php>. 22

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 9. ed. São Paulo: McGraw Hill, 2021. 35, 37

QUASAR. *Quasar Framework - Build high-performance VueJS user interfaces*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://quasar.dev/>. 23

RAFIQUE, K. et al. Cloud computing economics opportunities and challenges. In: *2011 4th IEEE International Conference on Broadband Network and Multimedia Technology*. Shenzhen, China: [s.n.], 2011. p. 401–406. Acesso em: 9 dez. 2024. Disponível em: <https://ieeexplore.ieee.org/abstract/document/6155965>. 13

SOMMERVILLE, I. *Engenharia de Software*. 9. ed. São Paulo: Pearson, 2011. 38

STEGAROIU, C.-E. The necessity of a specific security in the cloud. *Ideas.repec.org*, 2019. Acesso em: 9 dez. 2024. Disponível em: <https://ideas.repec.org/a/cbu/jrnlec/y2019v6p219-225.html>. 13

TUBER, D.; KARAAGAOGLU, O. *Network performance update: Birthday Week 2023*. 2023. Acesso em: 9 dez. 2024. Disponível em: <https://blog.cloudflare.com/network-performance-update-birthday-week-2023/>. 20, 21

VUE.JS. *Vue.js - The Progressive JavaScript Framework*. 2024. Acesso em: 9 dez. 2024. Disponível em: <https://vuejs.org/>. 23

APÊNDICE A - Descrição das tabelas do banco de dados no modelo físico

Quadro 12 – Tabela: organization_types

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador único do tipo	PK
name	VARCHAR(255)	Nome do tipo de organização	
description	TEXT	Descrição do tipo de organização	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 13 – Tabela: products

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador único do produto	PK
uuid	UUID	Identificador universal do produto	Único
name	VARCHAR(255)	Nome do produto	
price	DECIMAL	Preço do produto	
organization_id	INTEGER	Organização relacionada	FK → organizations.id
description	TEXT	Descrição detalhada do produto	Opcional
is_active	BOOLEAN	Produto está ativo	
default_image	VARCHAR(255)	Imagem padrão do produto	Opcional
images	JSON	Imagens adicionais do produto	Opcional
in_stock	INTEGER	Quantidade disponível em estoque	
is_stock	BOOLEAN	Controla estoque?	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 14 – Tabela: product_histories

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador do histórico	PK
product_id	INTEGER	Produto relacionado	FK → products.id
organization_id	INTEGER	Organização relacionada	FK → organizations.id
status	VARCHAR(255)	Status do produto	
price	DECIMAL	Preço no momento do histórico	
quantity	INTEGER	Quantidade registrada	
description	TEXT	Descrição adicional	Opcional
metadata	JSON	Dados adicionais do histórico	Opcional
created_at	DATETIME	Data de criação do histórico	
updated_at	DATETIME	Data de atualização do histórico	

Fonte: Elaborado pelo autor.

Quadro 15 – Tabela: customers

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador do cliente	PK
name	VARCHAR(255)	Nome do cliente	
email	VARCHAR(255)	Email do cliente	
phone_number	VARCHAR(20)	Telefone do cliente	
organization_id	INTEGER	Organização relacionada	FK → organizations.id
is_active	BOOLEAN	Cliente está ativo?	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 16 – Tabela: categories

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador da categoria	PK
name	VARCHAR(255)	Nome da categoria	
image	TEXT	Imagem da categoria	Opcional
organization_id	INTEGER	Organização relacionada	FK → organizations.id
is_active	BOOLEAN	Categoria ativa?	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 17 – Tabela: organization_users

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador único do vínculo entre organização e usuário	PK
organization_id	INTEGER	Referência à organização vinculada	FK → organizations.id
user_id	INTEGER	Referência ao usuário vinculado	FK → users.id
role	STRING	Papel do usuário dentro da organização	
created_at	DATETIME	Data de criação do registro	
updated_at	DATETIME	Data de atualização do registro	

Fonte: Elaborado pelo autor.

Quadro 18 – Tabela: product_additionals

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador do adicional	PK
name	VARCHAR(255)	Nome do adicional	
price	DECIMAL	Preço adicional	
organization_id	INTEGER	Organização relacionada	FK → organizations.id
is_active	BOOLEAN	Está ativo?	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 19 – Tabela: organization_addresses

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador do endereço	PK
organization_id	INTEGER	Organização relacionada	FK → organizations.id
street	VARCHAR(255)	Rua	
number	VARCHAR(10)	Número	
complement	VARCHAR(255)	Complemento	Opcional
neighborhood	VARCHAR(255)	Bairro	
city	VARCHAR(255)	Cidade	
state	VARCHAR(2)	Estado	
zip_code	VARCHAR(10)	CEP	
is_main	BOOLEAN	É o endereço principal?	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 20 – Tabela: customer_addresses

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador do endereço	PK
customer_id	INTEGER	Cliente relacionado	FK → customers.id
street	VARCHAR(255)	Rua	
number	VARCHAR(10)	Número	
complement	VARCHAR(255)	Complemento	Opcional
neighborhood	VARCHAR(255)	Bairro	
city	VARCHAR(255)	Cidade	
state	VARCHAR(2)	Estado	
zip_code	VARCHAR(10)	CEP	
is_main	BOOLEAN	É o endereço principal?	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 21 – Tabela: organization_configurations

Campo	Tipo	Descrição	Restrições
id	INTEGER	Identificador da configuração	PK
organization_id	INTEGER	Organização relacionada	FK → organizations.id
key	VARCHAR(255)	Chave da configuração	
value	TEXT	Valor da configuração	
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	

Fonte: Elaborado pelo autor.

Quadro 22 – Tabela: category_product

Campo	Tipo	Descrição	Restrições
category_id	INTEGER	ID da categoria	PK, FK → categories.id
product_id	INTEGER	ID do produto	PK, FK → products.id
created@	DATETIME	Data de criação do registro	Opcional
updated@	DATETIME	Data de atualização do registro	Opcional

Fonte: Elaborado pelo autor.

Quadro 23 – Tabela: product_products_additional

Campo	Tipo	Descrição	Restrições
product_id	INTEGER	ID do produto	PK, FK → products.id
products_additional_id	INTEGER	ID do adicional do produto	PK, FK → products_additionals.id
created@	DATETIME	Data de criação do registro	Opcional
updated@	DATETIME	Data de atualização do registro	Opcional

Fonte: Elaborado pelo autor.