UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

ANA LUIZA DA SILVA MAGALHÃES
LUCAS MONTIJO DO NASCIMENTO

CapiCrop: uma proposta de serviço de manipulação de imagens

Prof. Filipe Braida do Carmo, D.Sc.
Orientador

Nova Iguaçu, Dezembro de 2023

CapiCrop: uma proposta de serviço de manipulação de imagens

Ana Luiza da Silva Magalhães

Lucas Montijo do Nascimento

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____

Ana Luiza da Silva Magalhães

_____

Lucas Montijo do Nascimento

Aprovado por:

_____

Prof. Filipe Braida do Carmo, D.Sc.

_____

Prof. Bruno José Dembogurski, D.Sc.

_____

Prof. Marcelo Panaro de Moraes Zamith, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Dezembro de 2023

DOCUMENTOS COMPROBATÓRIOS Nº 28350/2023 - CoordCGCC (12.28.01.00.00.98)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 22/12/2023 21:52)
BRUNO JOSE DEMBOGURSKI
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: ###249#4

(Assinado digitalmente em 21/12/2023 18:18)
FILIPE BRAIDA DO CARMO
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: ###295#4

(Assinado digitalmente em 21/12/2023 19:34)
MARCELO PANARO DE MORAES ZAMITH
CHEFE DE DEPARTAMENTO
DeptCC/IM (12.28.01.00.00.83)
Matrícula: ###810#1

(Assinado digitalmente em 22/12/2023 18:09)
ANA LUIZA DA SILVA MAGALHÃES
DISCENTE
Matrícula: 2019######9

(Assinado digitalmente em 23/12/2023 00:19)
LUCAS MONTIJO DO NASCIMENTO
DISCENTE
Matrícula: 2019######4

*Visualize o documento original em https://sipac.ufrrj.br/documentos/ informando seu número: **28350**, ano: **2023**, tipo: **DOCUMENTOS COMPROBATÓRIOS**, data de emissão: **21/12/2023** e o código de verificação: **db238bcc2d***

Agradecimentos

Ana Luiza da Silva Magalhães

Agradeço primeiramente ao meu pai, Leoni Braga, que já não está mais aqui, mas foi a pessoa responsável pelo esforço diário que me possibilitou chegar até onde estou hoje, e a minha mãe, Jane Lucimar, que me incentivou e forneceu suporte durante toda graduação. Mãe, muito obrigada por nunca ter me deixado sentir desamparada. Agradeço a minha irmã, Mariana, por ser a minha melhor amiga e pessoa com quem sempre pude contar. Agradeço aos meus cachorros, Lucy, Lola e Picles, por terem sido as melhores companhias durante todo o tempo em que dediquei a esse trabalho.

Agradeço também às pessoas que me acompanharam durante esse projeto final, o meu amigo Lucas Montijo, que esteve comigo durante todos os trabalhos da graduação, não poderia ser diferente agora. Lucas, obrigada por ter tornado tudo mais leve, adotando o papel não só de dupla, como também de psicólogo. Agradeço ao nosso orientador, Filipe Braida, pela orientação, paciência, e por toda inspiração que nos forneceu durante a graduação.

Por fim, agradeço aos amigos que fiz pela vida e graduação. Obrigada, Julia Bartolo, por ser a minha primeira amizade no curso, sendo responsável por muitas risadas e descontração. Com você e o Lucas, até os momentos mais tensos conseguiram ser mais alegres. Agradeço também ao Merhi e ao Herbert, que também foram grandes amizades que fiz durante a faculdade. Aos meus amigos de vida, Ana Flávia, Jorge Lucas e Thiago, muito obrigada por cada momento em que vocês estiveram ao meu lado desde a infância. Sou muito grata por tê-los em minha vida, amo todos vocês.

Lucas Montijo do Nascimento

Agradeço em primeiro lugar à minha família, em especial meus pais, Ediléa Montijo e Sérgio do Nascimento, por todo apoio e incentivo oferecidos desde sempre. Obrigado por nunca terem medido esforços para que eu chegasse até aqui. À minha irmã, Talita Corrêa, sou grato por estar sempre ao meu lado em todos os melhores -e piores- momentos. Um agradecimento mais que especial à Chanel Kimberly, por encher todos os meus dias de alegria desde 2017. Sou grato por vocês me incentivarem a dar o meu melhor a todo tempo, amo vocês!

Agradeço também a todas as pessoas que tornaram esse trabalho possível, em especial a Ana Luiza por todas as reuniões que incluíram todo tipo de Graça e Bobagens. Obrigado por todos os trabalhos juntos, noites sem dormir, GRANDES risadas e shows insalubres (que eu te incentivei a ir). Agradeço também ao professor Filipe Braida pela paciência e orientação, não somente durante o período desse trabalho, mas também desde 2019.

Ao Gustavo Lima, quero expressar meu carinho e gratidão pelos 8 anos ao meu lado. Obrigado acreditar em mim em momentos onde até mesmo eu não acreditei. Essa conquista também é um pouco sua, amo você!

Sou grato às amizades que fiz dentro e fora da faculdade. Em especial à Julia Bartolo e Ana Luiza. Todos os dias com vocês foram mais leves e divertidos, em especial na nossa *Caixa de Pandora*. Agradeço também aos meus amigos Herbert Mariano e Merhi Osolins. As risadas que tivemos madrugada adentro durante os trabalhos tornaram minha experiência acadêmica mais leve e especial. Ao Arthur Cipolieschi, agradeço pela grande amizade que desenvolvemos tão espontaneamente em especial nas *Dailies*. Obrigado por tudo!

RESUMO

CapiCrop: uma proposta de serviço de manipulação de imagens

Ana Luiza da Silva Magalhães e Lucas Montijo do Nascimento

Dezembro/2023

Orientador: Filipe Braida do Carmo, D.Sc.

Com a popularização do uso da internet em diversos dispositivos, o contexto do uso de imagens passou por significativas transformações. Ao mesmo tempo, a pressa dos usuários em acessar conteúdo online se tornou uma preocupação entre a comunidade de desenvolvedores Web. As principais soluções procuradas por eles são a eficiência e adaptabilidade na entrega de imagens online, onde enfrentam diversos desafios, como a criação manual de várias versões da mesma imagem, gerando trabalho adicional e dificultando o gerenciamento dos arquivos salvos. Este trabalho apresenta o CapiCrop, uma ferramenta que simplifica a manipulação de imagens diretamente pela entrega por meio de uma API, incluindo uma interface para o gerenciamento de imagens salvas pelos usuários. O objetivo é aprimorar a experiência online para o usuário final, de forma a diminuir também as dificuldades enfrentadas pelos desenvolvedores.

ABSTRACT

CapiCrop: uma proposta de serviço de manipulação de imagens

Ana Luiza da Silva Magalhães and Lucas Montijo do Nascimento

Dezembro/2023

Advisor: Filipe Braida do Carmo, D.Sc.

With the popularization of internet usage across various devices, the context of image usage has undergone significant transformations. Simultaneously, users' sense of urgency to access online content has become a concern within the Web development community. The primary solutions used by developers include efficiency and adaptability in delivering images online, where they encounter various challenges such as manually creating multiple versions of the same image, generating additional work, and complicating the management of saved files. This paper introduces CapiCrop, a tool that simplifies image manipulation directly on user delivery via an API, including an interface for managing images saved by users. The goal is to enhance the online experience for the end user while also reducing the challenges faced by developers.

Lista de Figuras

Figura 2.1: Representação de imagens em meios digitais (Imagem: Extra) . . 5

Figura 2.2: Exemplo do funcionamento básico da requisição de recursos da página web no protocolo HTTP 6

Figura 2.3: Exemplo do carregamento de uma página web com *Lazy Loading* (Fonte: Webdev) 8

Figura 2.4: Exemplo básico do funcionamento de uma *Application Programming Interface* (API) 10

Figura 2.5: Exemplo do uso da API Dall-e 11

Figura 3.1: Média de número de requisições por elemento nas páginas web . . 13

Figura 3.2: Exemplo de mudança de resolução de imagens em diferentes dispositivos 14

Figura 3.3: Interface de usuário do Sirv 16

Figura 3.4: Interface de usuário do ImageEngine 16

Figura 3.5: Carregamento de imagens usando o Imagekit.io, com qualidades em 100% (à esquerda) e 60% (à direita) (Fonte: ImageKit.io) . . . 17

Figura 3.6: Diagrama C4 ilustrando interação dos usuários com o sistema . . 19

Figura 3.7: Diagrama C4 a nível de *container* ilustrando o funcionamento do sistema no lado do servidor 22

Figura 3.8: Diagrama C4 de componentes ilustrando a comunicação entre os elementos da Web API 25

Figura 3.9: Diagrama entidade relacionamento do sistema CapiCrop utilizando notação UML 26

Figura 4.1: Estrutura de pastas do projeto do sistema CapiCrop 34

Figura 4.2: Tela de Autenticação do sistema CapiCrop 35

Figura 4.3: Tela de gerenciamento de *tokens*. 36

Figura 4.4: Tela de upload e gerenciamento de imagens. 36

Figura 4.5: Exemplo de diretório em que as imagens são armazenadas no S3. 37

Figura 4.6: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=800x600 (à direita). 39

Figura 4.7: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=800x600 e *rotate*=90 (à direita). 39

Figura 4.8: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=400x600 e *color*=41x183x234 (à direita). 40

Figura 4.9: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=300x200 e *crop*=90x80x90x15 (à direita). 40

Lista de Tabelas

Tabela 2.1: Tamanhos de arquivo de uma imagem de 48 megapixels em diferentes formatos[3] 7

Lista de Abreviaturas e Siglas

CDN	Content Delivery Network
API	*Application Programming Interface*
DAM	Digital Asset Manager
HTTP	*Hypertext Transfer Protocol*
DOM	*Document Object Manager*
AJAX	JavaScript assíncrono + XML
JS	JavaScript
MVC	*Model-View-Controller*
JWT	*JSON Web Token*
URL	*Uniform Resource Locator*
RGB	*Red, Green and Blue*
SAAS	*Software as a service*

Sumário

Agradecimentos **ii**

Resumo **iv**

Abstract **v**

Lista de Figuras **vi**

Lista de Tabelas **viii**

Lista de Abreviaturas e Siglas **ix**

1 Introdução **1**

1.1 Objetivo 2

1.2 Organização do Trabalho 3

2 Fundamentação Teórica **4**

2.1 Imagens 4

2.2 Desenvolvimento Web e entrega de imagens 6

2.3 Software como Serviço 9

2.4 Web APIs 10

3 Proposta **12**

3.1 Motivação 12

3.2 Trabalhos Relacionados 15

3.3 CapiCrop 18

3.3.1 Interação com o usuário 19

3.3.2 Lado do servidor 21

3.3.3 Gerenciamento e processamento das imagens 24

4 Desenvolvimento **28**

4.1 Tecnologias utilizadas 28

4.1.1 Ambiente de desenvolvimento 29

4.1.2 TypeScript 30

4.1.3 Node.js 30

4.1.4 AdonisJS 31

4.1.5 Varnish 31

4.1.6 Bibliotecas 32

4.2 Desenvolvimento 32

4.2.1 Organização do código e padrão arquitetural 33

4.2.2 Cadastro do usuário e autenticação 34

4.2.3 Upload e armazenamento de imagens 35

4.2.4 Manipulação de imagens 37

5 Conclusão **41**

5.1 Considerações finais 41

5.2 Limitações e trabalhos futuros 42

Referências **44**

Capítulo 1

Introdução

Desde o surgimento da fotografia no século XIX, as imagens têm sido empregadas de diversas formas em nosso cotidiano, desde a simples captura de momentos até seu uso no marketing, educação, culinária, etc. A partir da popularização da internet, o uso de imagens tornou-se onipresente, transformando a comunicação visual. Redes sociais, memes, compartilhamento rápido e acesso fácil a bancos de imagens impactaram significativamente a forma como as pessoas se comunicam e consomem conteúdo visual online, fazendo com que as imagens se tornassem o conteúdo com segundo maior número de requisições. (FELIZARDO; SAMAIN, 2007)

Por outro lado, a crescente dependência dos usuários em serviços cada vez mais rápidos faz com que eles estejam cada vez mais impacientes em relação à entrega dos conteúdos (NIELSEN, 2010). Nesse cenário, a popularização de diferentes tipos de dispositivos também evidenciou a necessidade da criação de imagens específicas para cada plataforma (JEHL; MARCOTTE, 2014). Dessa forma, o uso personalizado de imagens tem se tornado o padrão entre a comunidade de desenvolvedores. Essa abordagem implica na criação de diversas versões da mesma imagem, que serão utilizadas em diferentes dispositivos. Isso faz com que o carregamento seja otimizado, já que em dispositivos com telas menores, imagens menores serão enviadas.

No entanto, ao assumir a responsabilidade de criar diferentes versões da mesma imagem, os desenvolvedores dependem de ferramentas externas para realizar a

manipulação. Além disso, a escolha da melhor versão da imagem para cada dispositivo, com o uso de imagens responsivas[1], é terceirizada ao navegador do usuário, o que pode fazer com que a aplicação desenvolvida não funcione de maneira esperada. Adicionalmente, ao fazer o versionamento da mesma imagem, gera-se também a dificuldade em gerenciar os diversos arquivos gerados com a manipulação.

Diante do contexto e dos desafios abordados, surgiu a necessidade de criar uma aplicação que oferecesse suporte aos desenvolvedores *Web* na gestão, manipulação e distribuição eficiente de imagens. Nesse sentido, este trabalho propõe o desenvolvimento do CapiCrop, um sistema de gerenciamento e manipulação de imagens, complementado por uma API *Web* que se destina a facilitar na entrega de imagens aos usuários finais.

1.1 Objetivo

O principal objetivo deste trabalho é propor e implementar uma ferramenta de gerenciamento e manipulação de imagens, juntamente a uma API que se integra às aplicações do desenvolvedor, facilitando a entrega de conteúdo aos usuários finais. Dessa forma, os desenvolvedores poderão utilizar imagens em seus projetos de forma personalizada, sem depender de ferramentas externas para ter acesso a funções básicas de manipulação.

Com intuito de atender ao objetivo deste trabalho, será proposta uma solução baseada em serviços, que aborda diversas estratégias para a entrega eficiente de imagens ao usuário final, bem como para o armazenamento dessas imagens no servidor e aprimoramento da interface da aplicação. A proposta será feita com base na eficiência de entrega e de uso de recursos do servidor.

Outro ponto que destaca o propósito deste trabalho é a implementação prática da solução proposta. Isso envolverá o desenvolvimento do CapiCrop, o sistema de gerenciamento e manipulação de imagens, juntamente com a integração da API Web associada. A implementação será guiada com base na facilidade dos desenvolvedores

[1] Imagens responsivas, MDN Web Docs

no gerenciamento e entrega das imagens.

Por fim, o último pilar consiste em realizar uma avaliação geral da solução desenvolvida. Serão analisados os pontos positivos, destacando os benefícios obtidos com a implementação do CapiCrop, bem como as limitações observadas durante o processo de desenvolvimento. Além disso, serão identificadas e discutidas oportunidades de aprimoramento e próximos passos.

1.2 Organização do Trabalho

Este trabalho está organizado da seguinte forma:

- **Capítulo 2**: Serão discutidos os tópicos essenciais para entendimento deste trabalho, envolvendo representação digital de imagens, contexto do desenvolvimento web, utilização de Web APIs, etc.
- **Capítulo 3**: Serão abordados os principais fatores que motivaram o desenvolvimento desta solução. Além disso, serão discutidos outros trabalhos relacionados a este. Por fim, será apresentada a proposta e a modelagem do sistema.
- **Capítulo 4**: Serão abordados tópicos sobre o desenvolvimento do sistema, como tecnologias utilizadas, ambiente de implementação e a implementação da proposta em si.
- **Capítulo 5**: Serão discutidos os pontos positivos sobre a solução implementada, bem como pontos de melhoria e próximos passos a serem desenvolvidos.

Capítulo 2

Fundamentação Teórica

Este capítulo aprofunda alguns assuntos essenciais para a compreensão deste trabalho. Inicialmente, a Seção 2.1 elabora sobre o uso de imagens em contextos digitais. A Seção 2.2 aborda tópicos referentes ao desenvolvimento web, bem como técnicas utilizadas na entrega eficiente de imagens. Já na Seção 2.3, será introduzido o conceito de *Software as a service* (SAAS). Por fim, a Seção 2.4 inclui a definição das APIs e *Web services*.

Levando em consideração os tópicos a serem abordados, espera-se que seja ressaltada a relevância das imagens no contexto da web, bem como questões relacionadas a entrega de conteúdo para o usuário final. Esse é um ponto que se representa como um desafio para os desenvolvedores, que precisam lidar com fatores como o tamanho do arquivo e resolução ideal da imagem para cada dispositivo a fim de garantir a melhor experiência de usuário.

2.1 Imagens

Desde que as câmeras fotográficas foram inventadas no século XIX, o uso de imagens se tornou cada vez mais comum (RODRIGUES, 2007). Com o avanço tecnológico, as imagens deixaram de ser apenas fotografias físicas e passaram a incluir suas representações digitais. Essa transição é desafiadora, pois representar

digitalmente uma imagem de forma perfeita, capturando cada detalhe exato da realidade, é uma tarefa impossível. Ainda assim, as imagens são responsáveis por grande parte do tráfego na internet e podem ajudar no engajamento de uma aplicação web (BENDELL et al., 2016).

De forma digital, as imagens são representadas por uma matriz de pixels. Cada pixel representa a menor unidade de uma imagem, e é composto essencialmente por 3 unidades de números que variam de 0 a 255, representando a presença de cor vermelha, verde e azul (RGB), como representado na Figura 2.1. Quanto mais alto o número, maior será a presença de luz da respectiva cor naquele pixel e a combinação das 3 variáveis pode representar mais de 16 milhões de cores.

Figura 2.1: Representação de imagens em meios digitais (Imagem: Extra)

Dessa forma, a manipulação de imagens no contexto digital também é a manipulação de matrizes. Por exemplo, o recorte pode ser realizado selecionando apenas os elementos da matriz referentes ao conteúdo requerido, a correção de cor pode ser realizada adicionando ou subtraindo valores aos pixels e a filtragem de uma imagem pode ser realizada multiplicando a matriz da imagem por uma matriz de filtro que, por exemplo, aumenta o brilho incrementando valores de cada pixel.

2.2 Desenvolvimento Web e entrega de imagens

O desenvolvimento de aplicações para a internet teve grande avanço desde a construção do primeiro *website* e navegador por Tim Berners-Lee em 1991[1]. O cenário tecnológico evoluiu significativamente, moldando não apenas a aparência das páginas, mas também a funcionalidade e a experiência do usuário ao longo das décadas.

A troca de informações entre cliente e servidor, como destacado por Berners [1], é possibilitada pelo *Hypertext Transfer Protocol* (HTTP). O funcionamento desse protocolo baseia-se na comunicação bidirecional entre clientes (por meio de requisições) e servidores (mediante respostas)[2]. Essa dinâmica ainda é fundamental nas aplicações modernas, pois permite o acesso remoto de recursos mais avançados de forma facilitada, como ilustrado na Figura 2.2.

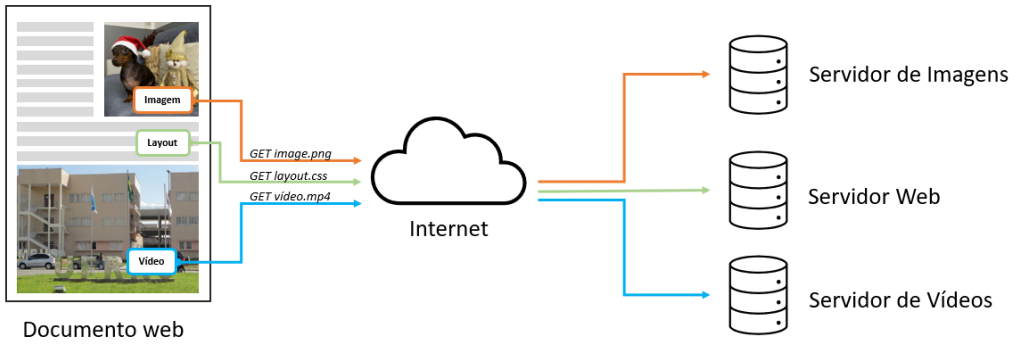

Figura 2.2: Exemplo do funcionamento básico da requisição de recursos da página web no protocolo HTTP (Fonte: Mozilla).

A evolução tecnológica do desenvolvimento web não se limitou ao protocolo HTTP. O crescimento da capacidade de processamento dos computadores, juntamente com a introdução do JavaScript (JS), foram essenciais para o avanço das aplicações web. O JS permitiu que as páginas da web fossem mais dinâmicas e interativas, possibilitando a criação de formulários interativos, jogos e aplicativos web, e a personalização da experiência dos usuários.

Nesse contexto, o surgimento de ferramentas como jQuery facilitou ainda mais o

[1] <https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt>
[2] <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>

desenvolvimento de aplicações mais complexas, com ainda menos linhas de código. Isso foi atingido pela simplificação de tarefas de manipulação do *Document Object Manager* (DOM), o que permitiu ocultar elementos da página sob demanda. Ainda, a introdução JavaScript assíncrono + XML (AJAX) facilitou na atualização individual de elementos do documento, sem precisar que toda a página recarregue (YORK, 2015).

Além das ferramentas desenvolvidas, que evoluíram a estrutura das aplicações, o conteúdo dos websites também mudou com o tempo. Hoje, o uso correto de imagens pode ajudar no engajamento de uma aplicação web, atraindo mais a atenção dos usuários. Contudo, nem sempre a entrega das imagens do servidor para o cliente final é uma tarefa fácil, dependendo de fatores como o tamanho da imagem e velocidade da conexão. Levando em consideração que os atrasos na entrega de conteúdo comprometem a experiência dos usuários (NIELSEN, 2010), a entrega de imagens de forma eficiente tem sido uma preocupação para os desenvolvedores.

Considerando o cenário de entrega de uma imagem com 48 megapixels (milhões de pixels), em um formato de 8 mil x 6 mil pixels, que possui aproximadamente 72MB no formato *Lossless JPEG*[3], como ilustrado na Tabela 2.1. Em uma conexão de 100 Mbps, o carregamento dessa imagem levaria aproximadamente 5.8 segundos, que já é percebido como lentidão pelo o usuário final.

Formato	**Tamanho**
JPG 100% 24bit/pixel	*9,8MB*
JPG 90% 24 bit/pixel	*4,9MB*
GIF 8bit/pixel, 255 colors	*14,7MB*
PNG losless 24bit/pixel	*27,9MB*
Lossless JPEG	*72MB*

Tabela 2.1: Tamanhos de arquivo de uma imagem de 48 megapixels em diferentes formatos[3]

Porém, há técnicas que diminuem o tamanho do arquivo, como a mudança do formato da imagem. A mesma imagem pode ser entregue em um formato comprimido, como o *JPG 90% 24 bit/pixel*, que reduz em 93% o tamanho do arquivo, se comparado ao formato *Lossless JPEG*, como exemplificado na Tabela 2.1. Existe também a

[3]https://toolstud.io/

possibilidade de redimensionar a imagem de acordo com o dispositivo, entregando imagens em tamanhos e resolução menores para dispositivos com telas menores, melhorando ainda mais o tempo de carregamento para o usuário final.

Tão como o redimensionamento ou compressão da imagem, outras técnicas podem ser empregadas para entregar imagens de forma mais eficiente. Uma boa parte das imagens presentes em aplicações web é carregada de forma desnecessária, já que 80% das imagens não estão visíveis para o usuário assim que ele acessa a aplicação. Para resolver isso, os desenvolvedores empregam o *lazy loading*, que atrasa o carregamento de imagens que ainda não estão visíveis na página. Quando o usuário acessa a parte da página onde a imagem está localizada, ela é carregada. O funcionamento da técnica pode ser entendido na Figura 2.3 (BENDELL et al., 2016).

Figura 2.3: Exemplo do carregamento de uma página web com *Lazy Loading* (Fonte: Webdev)

Além da compressão e otimização de imagens, o *cache* é uma técnica importante para a entrega eficiente de imagens na web. No contexto de entrega de imagens,

o *cache* consiste em armazenar temporariamente as imagens mais requisitadas na memória, o que reduz o tempo de carregamento, pois não é necessário esperar pelo servidor para carregar as imagens (SALHI et al., 2018).

Existem 2 tipos principais de cache: do lado do cliente e do lado do servidor. Em aplicações Web, o cache do cliente utiliza o armazenamento do navegador para guardar as informações requisitadas, evitando que sejam enviadas novamente ao servidor, caso já tenham sido solicitadas anteriormente (SöDERMARK, 2023). Já o cache do servidor armazena o resultado das requisições mais recentes ou populares, evitando que o servidor reprocesse a requisição, retornando o resultado diretamente aos usuários, reduzindo o tempo de resposta do sistema[4].

2.3 Software como Serviço

A integração de sistemas tornou-se essencial em diversos setores, desde a gestão empresarial até a automação de processos mais específicos como análise de dados. No entanto, o investimento necessário para o desenvolvimento e manutenção de *software* pode ser significativo, podendo ser insustentável em alguns cenários. Diante disso, surge o conceito de SAAS, uma abordagem na qual um provedor disponibiliza o sistema como um serviço, enquanto o cliente o utiliza sem a necessidade de se preocupar com as complexidades da manutenção (MELO et al., 2007).

Além disso, destaca-se o conceito de SAAS de código aberto, representado por sistemas que adotam o modelo de serviços enquanto disponibilizam seu código-fonte para acesso e modificação, que muitas vezes são gratuitos. Essa abordagem não se limita a oferecer somente a colaboração da comunidade para a evolução do *software*, mas a disponibilização dele de forma transparente. Esse é o caso do WordPress[5], um projeto de código aberto, amplamente utilizado para criar sites, blogs e aplicativos web.

[4] <https://www.codingninjas.com/studio/library/server-side-caching-and-client-side-caching>
[5] <https://br.wordpress.org/>

2.4 Web APIs

Existem várias maneiras de interagir com os *softwares*, como as interfaces de usuário, que facilitam o acesso às funcionalidades das aplicações pelas pessoas. No entanto, essa abordagem mostra-se limitada quando os sistemas precisam se comunicar, acessar e utilizar funcionalidades de maneira automatizada. Nesse contexto surgem as APIs. Conceitualmente, uma API pode ser definida como um a interface que permite a interação entre diferentes sistemas (BIEHL, 2015).

Uma Web API, também conhecida como *Web Service*, representa um tipo específico de API, sendo definida como um conjunto de módulos que encapsulam e abstraem recursos por meio da Web, possibilitando um processo de desenvolvimento mais ágil e eficiente (ZHANG et al., 2023). Ao utilizar essa abordagem, os desenvolvedores conseguem abstrair grande parte da complexidade do código, optando por estruturas mais simples e facilitando o processo de implementação[6].

A Figura 2.4 ilustra o funcionamento de uma Web API de recuperação de arquivos, que elimina a necessidade do desenvolvedor interagir diretamente com o servidor ou a base de dados. Juntamente à obtenção de dados de servidores, as APIs Web são utilizadas para diversos fins, como desenhar elementos gráficos na tela, serviços de mapa, clima, redes sociais e pagamentos. Com a abstração, é possível até mesmo criar imagens do zero utilizando inteligência artificial[7] com um pequeno trecho de código, como ilustrado na Figura 2.5.

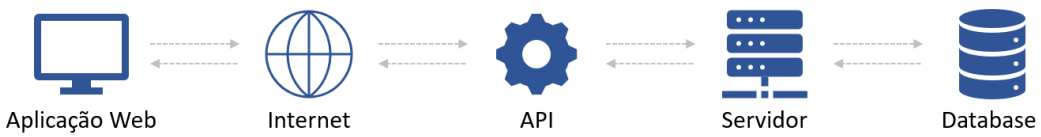

Figura 2.4: Exemplo básico do funcionamento de uma API (Fonte: DHTMLX)

O *endpoint* de uma Web API pode ser definido como um ponto de acesso pelo

[6]<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Client-side_web_APIs/Introduction>

[7]<https://openai.com/blog/dall-e-api-now-available-in-public-beta>

qual serão acessadas as funcionalidades da interface (SINGJAI et al., 2021). No caso do exemplo dado na Figura 2.5, são encaminhados parâmetros ao *endpoint* que controlarão as funcionalidades requeridas, como o *prompt*, que dita o que estará na imagem gerada e o *size*, que define o tamanho da imagem gerada.

Figura 2.5: Exemplo do uso da API Dall-e (Fonte: Dall-e).

Além dos parâmetros para funcionamento da aplicação, também é enviado um parâmetro de identificação do cliente, a *authorization*, ou *API Key*. Essa chave serve para autenticar e autorizar o acesso do cliente aos recursos da aplicação e desempenha um papel crucial na segurança da comunicação entre o cliente e a aplicação, garantindo que apenas usuários autorizados possam interagir com os serviços oferecidos.

Capítulo 3

Proposta

Neste capítulo, o sistema proposto será introduzido, abordando suas motivações. Inicialmente, na Seção 3.1, será discutida a problemática relacionada ao uso de imagens no processo de desenvolvimento web. Em seguida, na Seção 3.2, serão discutidos trabalhos que têm relação com este. Finalmente, na Seção 3.3, será apresentada a proposta e a modelagem do sistema, com base nas problemáticas previamente abordadas.

3.1 Motivação

As imagens são elementos essenciais em qualquer site ou aplicativo web. Elas podem ser usadas para transmitir informações, chamar a atenção e atrair usuários. O seu uso correto pode gerar um aumento significativo no engajamento das plataformas, com estudos mostrando que imagens com alta qualidade geram um aumento de 5% em vendas no *eBay* e 2x mais comentários no *Facebook* (BENDELL et al., 2016).

O aumento da popularidade das imagens fez com que elas se tornassem o segundo conteúdo com mais solicitações em páginas web, conforme a Figura 3.1. O número de requisições, em conjunto com o tamanho das imagens, desempenham um papel fundamental no tempo de carregamento da página, já que quanto maior a quantidade e o tamanho desses elementos, maior será o tempo de carregamento para o usuário

final (MUNYARADZI; MAXMILLAN; AMANDA, 2013).

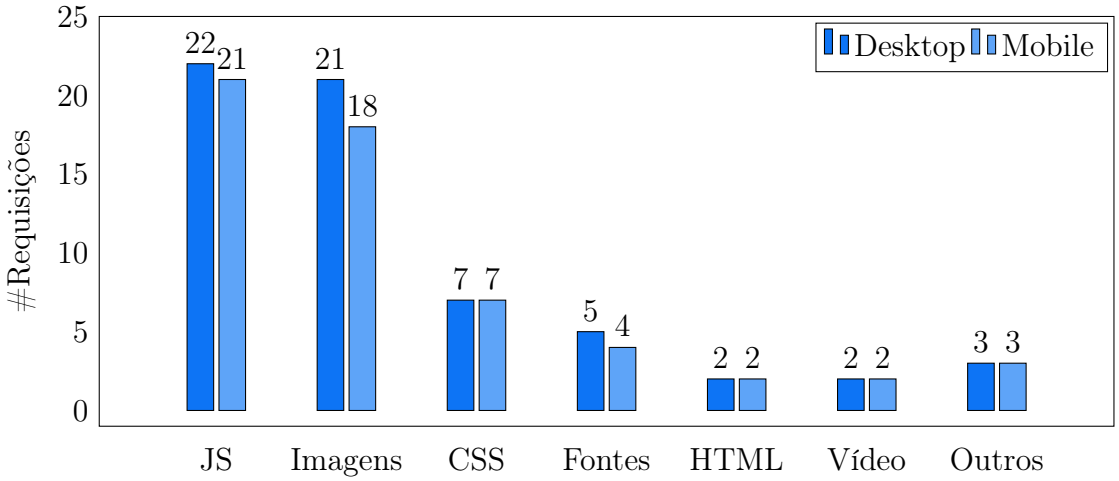

Figura 3.1: Média de número de requisições por elemento nas páginas web (HTTP Archive).

Nesse contexto, a eficiência na entrega de imagens é uma consideração técnica que influencia diretamente na experiência dos usuários. Sites com tempo de carregamento menor têm uma taxa de permanência maior e menos rejeição (NIELSEN, 2010). Segundo estudos, o estresse causado pela demora na resposta de um site é semelhante ao de assistir a um filme de terror ou resolver um problema matemático[1].

Outros fatores que aumentaram o desafio na entrega de imagens foram o surgimento de dispositivos móveis, como smartphones, e o surgimento do design responsivo. Os dispositivos atuais possuem telas de tamanhos diferentes, o que exige que as imagens sejam fornecidas também em resoluções diferentes (JEHL; MARCOTTE, 2014). Por conta disso, o uso de imagens responsivas é cada vez mais comum em sites, como exemplificado na Figura 3.2.

Para atender a essa demanda, os desenvolvedores precisam criar várias versões da mesma imagem com diferentes resoluções. Isso cria a dependência em ferramentas externas de edição de imagens, como o GIMP[2], para tarefas simples, como recorte, redimensionamento e rotação. No entanto, o uso desse tipo de ferramenta pode causar trabalhos manuais e repetitivos, prejudicando a produtividade.

[1] Vídeo que não carrega e conexões lentas estressam usuários móveis, Ericsson
[2] <https://www.gimp.org/>

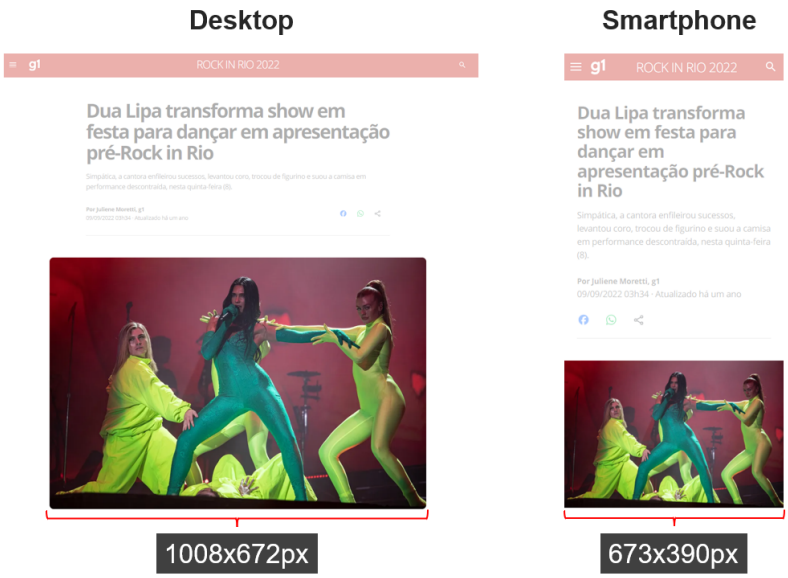

Figura 3.2: Exemplo de mudança de resolução de imagens em diferentes dispositivos (Fonte: G1).

Após a criação das versões da imagem, o desenvolvedor as carrega na aplicação como um único objeto de imagem responsiva[3], terceirizando para o navegador a escolha da melhor versão. No entanto, essa abordagem pode resultar em um comportamento inesperado da aplicação, como uso de tamanhos de imagem que não são ideais para o dispositivo em questão, especialmente se o desenvolvedor não tiver gerado uma versão apropriada para ele.

Paralelamente, a gestão de arquivos também se apresenta como um desafio, especialmente em aplicações de grande porte. Há ferramentas disponíveis que minimizam esse problema, como um Digital Asset Manager (DAM), que funciona como uma biblioteca centralizada de arquivos, e que tem bons resultados no processo de desenvolvimento web (CECCONI et al., 2020). Contudo, essa solução é mais cara e voltada ao meio empresarial, sem que haja preocupação com a entrega dos arquivos para os usuários finais.

Portanto, a gestão eficiente de imagens para web é um desafio para desenvolvedores, que precisam criar diversas versões da mesma imagem para atender a diferentes dispositivos e tamanhos de tela, o que é uma tarefa manual e repetitiva. Uma pesquisa

[3]Imagens responsivas, MDN Web Docs

conduzida pela *Cloudinary*[4] mostra que o uso de um serviço de gerenciamento e manipulação de imagens pode salvar mais 240 horas de trabalho no desenvolvimento de um projeto por mês. Além disso, permitiria aos desenvolvedores (i) Armazenar imagens em um local central, facilitando o acesso, distribuição e entrega aos usuários; (ii) Criar diferentes versões de imagens de forma automática, sem a necessidade de intervenção manual ou ferramentas externas; (iii) Manipular imagens com facilidade, permitindo redimensionar, recortar e aplicar filtros.

3.2 Trabalhos Relacionados

Considerando o cenário de problemas descrito na Seção 3.1, diversas ferramentas de entrega de imagens já foram desenvolvidas. Contudo, poucas opções são disponibilizadas de forma gratuita e, as que são disponibilizadas, possuem uma limitação em seu uso e na quantidade de ferramentas disponíveis. Há ainda casos onde a usabilidade é comprometida pela falta de uma interface gráfica amigável para que o desenvolvedor gerencie suas imagens hospedadas.

O Sirv[5] é um serviço proprietário de armazenamento e processamento de imagens que tem como foco principal a otimização da visualização de imagens em *websites*. Ele permite o redimensionamento e recorte, bem como a aplicação de filtros pre-estabelecidos, tendo como principal diferencial a possibilidade de visualização de imagens em 360°. Em sua versão gratuita, a ferramenta possui apenas 0.5GB de armazenamento, com opções de planos mensais de $19 e $999 dólares, equivalentes a R$98 e R$5.160, em fevereiro de 2023. Além disso, o Sirv possui uma interface de usuário que facilita o gerenciamento dos arquivos hospedados, representada pela figura 3.3.

O ImageEngine[6] é outro serviço proprietário de armazenamento e processamento de imagens que tem como objetivo otimizar o carregamento de imagens em sites, visando a melhoria na experiência do usuário final. Ele utiliza uma Content Delivery

[4]The State of Media, 2020 Report
[5]<https://sirv.com/>
[6]<https://imageengine.io/>

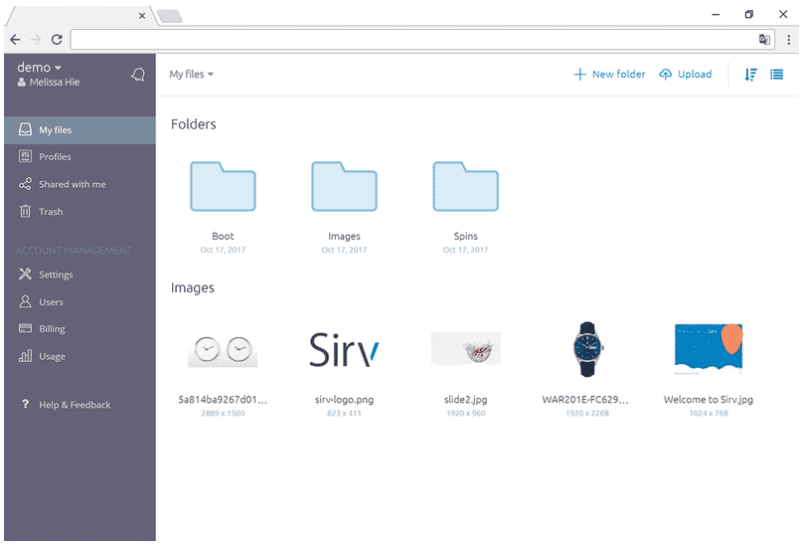

Figura 3.3: Interface de usuário do Sirv (Fonte: Sirv).

Network (CDN) própria para otimização da entrega de conteúdo para o usuário. Seu principal diferencial é o reconhecimento de características do dispositivo do usuário, o que permite saber, por exemplo, a densidade de pixels da tela onde a imagem será exibida, permitindo que o redimensionamento seja feito da melhor forma, sem perder qualidade significativa na exibição final. Não existe uma opção gratuita de uso, e os planos mensais equivalem a $49 e $99 dólares, equivalentes a R$253 e R$511, em fevereiro de 2023. O ImageEngine também possui uma interface do usuário, representada pela figura 3.4, por onde os usuários podem gerenciar os arquivos hospedados e medir o uso da ferramenta.

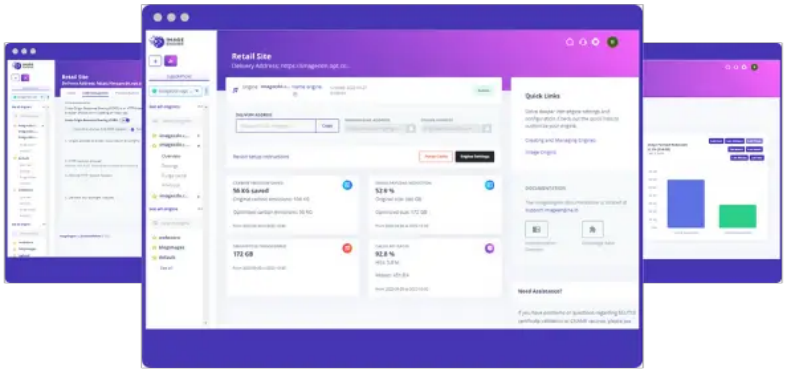

Figura 3.4: Interface de usuário do ImageEngine (Fonte: ImageEngine.io).

O ImageKit.io[7] é um serviço proprietário de armazenamento e processamento de

[7] <https://imagekit.io/>

imagens e vídeos para otimização da entrega de conteúdo para o usuário. Ele utiliza algoritmos de aprendizado para processamento de imagens, possibilitando o recorte e redimensionamento inteligentes, bem como ferramentas mais gerais como a remoção do fundo de imagens. Além disso, existe um serviço de gerenciamento do conteúdo hospedado, facilitando o uso pelos desenvolvedores. Na figura 3.5 temos um exemplo de como o serviço possibilita diminuir a quantidade de detalhes de uma imagem de 100% para 60%, sem perder informações úteis para o usuário final, diminuindo o seu tempo de carregamento. A versão gratuita da ferramenta possui 20GB de armazenamento, mas o número de operações é limitado a 500, impossibilitando o seu uso gratuito em projetos de grande escala.

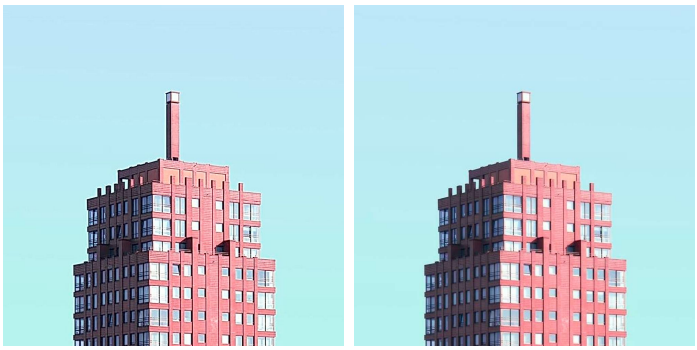

Figura 3.5: Carregamento de imagens usando o Imagekit.io, com qualidades em 100% (à esquerda) e 60% (à direita) (Fonte: ImageKit.io)

Por fim, o Thumbor[8] é um serviço de código aberto de processamento de imagens sob demanda que permite recortar, redimensionar e aplicar filtros a imagens armazenadas localmente ou por meio de uma *Uniform Resource Locator* (URL). Por ser destinado à exibição de miniaturas de imagens em sites, o principal destaque do Thumbor é o uso de algoritmos de aprendizagem para reconhecimento de rostos ou objetos em imagens, permitindo que o recorte e redimensionamento sejam aplicados de maneira inteligente. O ponto negativo no uso da ferramenta é a falta de um serviço de gerenciamento de imagens, o que pode se tornar uma dor de cabeça para o desenvolvedor.

[8] <http://www.thumbor.org/>

3.3 CapiCrop

Conforme discutido nas seções anteriores, imagens têm se mostrado essenciais em todos os tipos de aplicações voltadas para a web. Contudo, existem diversos desafios relacionados ao uso de imagens em sites, em especial na fase de desenvolvimento. Dentre esses, pode-se destacar a velocidade de entrega, armazenamento e recuperação, além da manipulação da imagem, não somente por fatores estéticos como também para a garantia da responsividade do conteúdo.

Inicialmente, a entrega de imagens surge como um desafio no contexto de desenvolvimento web. Isso ocorre, em grande parte, devido a necessidade cada vez maior por experiências instantâneas, tornando a velocidade de carregamento do conteúdo um fator crucial para o sucesso das aplicações.

Além disso, a introdução de diferentes tipos de dispositivo, como *smartphones* e *tablets*, introduziu o uso das imagens responsivas, aquelas que se adaptam ao tamanho da tela do dispositivo. Para fazer o uso desse tipo de estrutura, os desenvolvedores precisam criar diversas versões de uma única imagem, gerando a dependência em ferramentas externas e dificuldade com o armazenamento e gerenciamento dos arquivos.

Por fim, buscando garantir a melhor adaptação do conteúdo à página, gera-se um processo repetitivo de edição de imagens. A partir disso, cria-se a necessidade do uso de ferramentas externas de manipulação para processos corriqueiros, como o redimensionamento, rotação, recorte, dentre outras funcionalidades de processamento de imagens, adicionando ainda mais tempo ao processo de desenvolvimento.

Nesse cenário, a implementação de um sistema que busca facilitar o gerenciamento e a entrega de imagens pode agregar grande valor ao processo de desenvolvimento web. O CapiCrop, sistema de armazenamento e entrega de imagens aqui proposto, visa ajudar nisso, fornecendo um meio prático e centralizado de entrega e manipulação de imagens. O sistema atua em duas frentes: uma interface por onde os usuários terão acesso ao gerenciamento das imagens hospedadas, e uma API por onde serão entregues e manipuladas.

Portanto, nas próximas seções será apresentada a proposta do sistema, abordando questões referentes tanto à interação do usuário com o sistema, como pontos mais técnicos que envolvem a tomada de decisões relacionadas a arquitetura do sistema.

3.3.1 Interação com o usuário

No lado do cliente, usuários do sistema contarão com uma interface simples e intuitiva para o envio, visualização e gerenciamento das imagens hospedadas. Haverá suporte a usuários administradores, que terão permissão de incluir ou excluir outros usuários do sistema. As interações básicas dos usuários com o sistema podem ser entendidas na Figura 3.6.

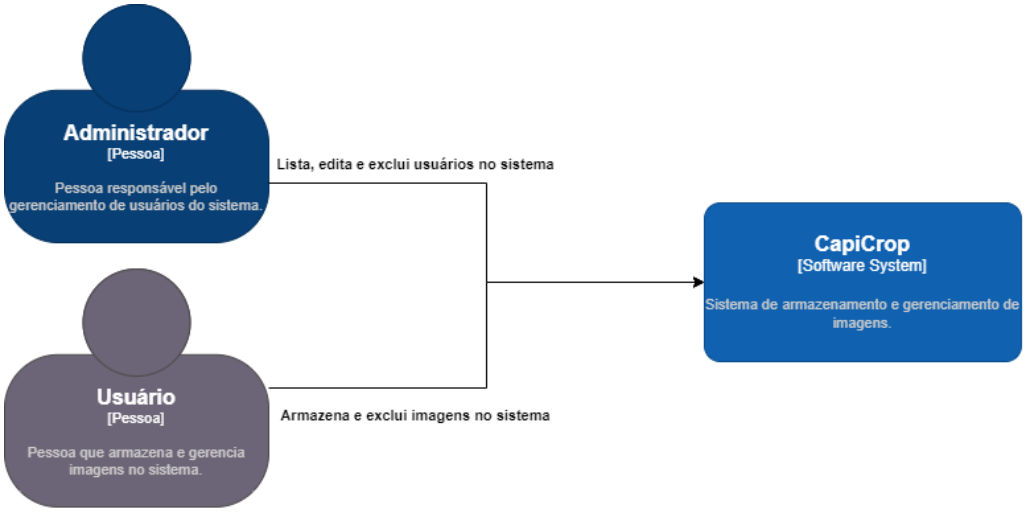

Figura 3.6: Diagrama C4 ilustrando interação dos usuários com o sistema

Além da interface do usuário, será disponibilizada uma API Web, que os desenvolvedores poderão integrar às suas aplicações. Por ela, os usuários poderão recuperar e manipular as imagens hospedadas com ferramentas como recorte, redimensionamento e rotação. As chamadas da API que permitirão a manipulação e exibição das imagens armazenadas serão baseadas no protocolo HTTP e seguirão o seguinte formato de URL:

```
[host]/api/capi/[parâmetros de processamento]/[ID da imagem]
```

Onde:

- **Host**: Refere-se ao endereço onde a aplicação está hospedada.
- **Parâmetros de processamento**: Representa os argumentos enviados à API para realizar manipulações específicas na imagem.
- **ID da imagem**: É o identificador único atribuído à imagem quando armazenada no sistema. Ele será apresentado ao usuário após o *upload* das imagens.

Através da manipulação por meio da API, é eliminada a necessidade do desenvolvedor criar diversas versões da mesma imagem, simplificando a implementação. A construção do projeto deve facilitar a inclusão de novas ferramentas de manipulação, garantindo uma estrutura que permite a expansão das funcionalidades disponíveis. Com isso, alguns possíveis parâmetros são:

- `resize=AxB` redimensiona a imagem para o formato desejado, com largura A e altura B.
- `crop=AxB` recorta a imagem de forma centralizada, para o tamanho AxB.
- `rotate=X` rotaciona a imagem em X^{o}.
- `color=RxGxB` aplica um filtro de cor na imagem, recebendo como parâmetro valores na escala *Red, Green and Blue* (RGB)

Além disso, a segurança é outro fator a ser considerado durante a implementação do sistema. A atribuição de um identificador único e não previsível a cada imagem é uma medida crucial para evitar que elas sejam obtidas por *crawlers*, rastreadores de conteúdo online capazes de extrair grandes volumes de dados de páginas web em tempo real (WAN; LI; SUN, 2019).

Essa técnica opera essencialmente a partir de uma URL inicial, onde o processo consiste em obter os links presentes nela e, posteriormente, repetindo esse padrão. Também existem métodos para inferir URLs de conteúdos específicos, especialmente quando o identificador do arquivo segue uma sequência. Isso possibilita a extração de

maneira sistemática dos arquivos de uma aplicação web, sendo, portanto, dificultado quando os nomes de arquivos não seguem um padrão sequencial. Dessa forma, o identificador das imagens será composto por seu nome original, formato do arquivo e o código único, de forma que:

- **Nome original do arquivo:** `imagem.jpg`
- **Exemplo de código gerado automaticamente:** `434C4C50`
- **Identificador final da imagem:** `imagemjpg-434C4C50`

3.3.2 Lado do servidor

As solicitações de armazenamento e manipulação de imagens serão processadas no lado do servidor de forma assíncrona, ou seja, enquanto a requisição está sendo processada pelo servidor, o restante do código continua a ser executado. O diagrama apresentado na Figura 3.7 ilustra a interação entre os elementos do lado do servidor, delineando o fluxo de processamento.

Um dos principais requisitos do sistema é a compatibilidade com os formatos de imagem amplamente utilizados atualmente, tais como JPEG, PNG e WebP. Dessa forma, garante-se uma abrangência eficaz para atender as necessidades e preferências dos usuários, proporcionando uma experiência mais personalizada aos diferentes cenários de uso.

Há diferentes abordagens que podem ser usadas para gerenciar o armazenamento de conteúdo no sistema de arquivos. Essas escolhas têm implicações diretas no tempo de resposta e recursos usados pelo servidor para armazenar e processar as imagens. Entre as estratégias disponíveis para o armazenamento dos arquivos, destacam-se as opções:

1. **Criar versões predefinidas ao salvar a imagem no servidor**: Com processamentos predefinidos, a implementação do projeto é facilitada, mas podem haver versões das imagens que nunca seriam usadas, gerando armazenamento

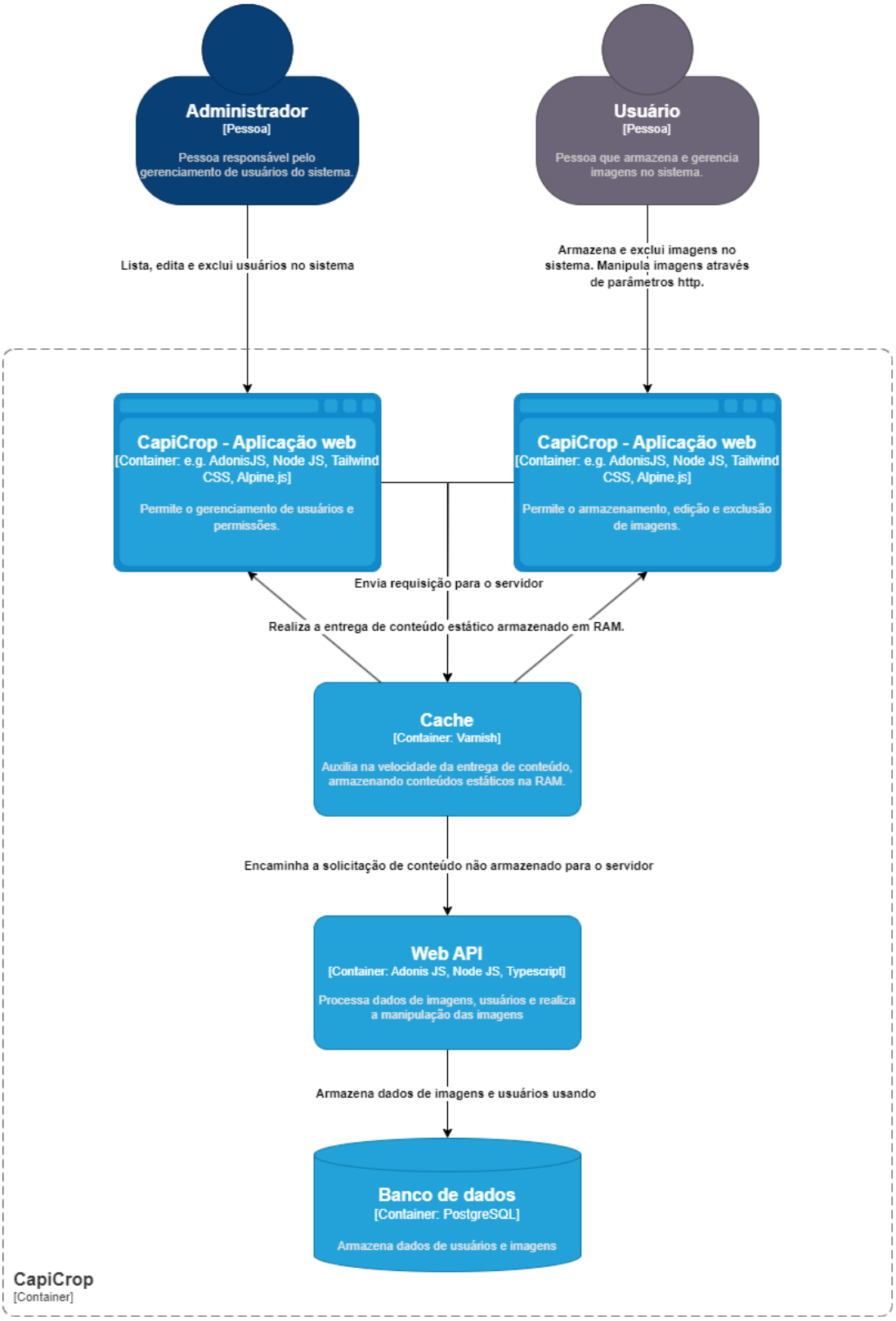

Figura 3.7: Diagrama C4 a nível de *container* ilustrando o funcionamento do sistema no lado do servidor

desnecessário no servidor. Além disso, não há como prever com certeza quais versões da imagem serão requisitadas.

2. **Processar a imagem toda vez em que há uma requisição**: Dessa forma, há possibilidade de sobrecarga no servidor e aumento na latência, já que o ele terá que lidar com mais operações por vez. Além disso, é um uso ineficiente de recursos, já que o mesmo processamento pode ser requisitado repetidas vezes e em todas elas o servidor usará recursos para reprocessar a imagem, caso ela não esteja hospedada no cache do servidor.

3. **Salvar uma cópia das imagens processadas no servidor**: Caso cada imagem processada tenha uma cópia salva, ela pode ser reutilizada quando uma solicitação com os mesmos parâmetros for feita. Isso diminui o tempo de resposta do servidor, pois a imagem não será reprocessada. Porém, caso uma requisição específica seja feita uma única vez, a cópia da imagem será salva no servidor por tempo indeterminado, consumindo recursos de armazenamento.

Considerando as opções, a mais adequada ao projeto proposto é a 3ª (salvar uma cópia das imagens processadas), pois gera uma resposta mais rápida com menos recursos. Para contornar o problema do armazenamento de cópias não utilizadas, haverá um tempo máximo de permanência da imagem processada no servidor após sua última requisição. Assim, requisições mais frequentes terão sempre uma cópia salva, diminuindo seu tempo de carregamento. Para o usuário, apenas a imagem original é visível no gerenciador de arquivos.

Para otimizar a entrega de imagens, será implementado um cache antes de encaminhar as requisições para o servidor de processamento, permitindo o armazenamento temporário em memória das imagens processadas. Com essa abordagem, a entrega ao usuário é mais rápida, pois elimina a necessidade de processamento ou busca das imagens no servidor. Caso a imagem não esteja armazenada no cache, a requisição é enviada ao servidor, onde será processada. Portanto, os seguintes passos serão realizados durante o ciclo da requisição:

Passo 1 : O navegador do usuário verifica se a mesma requisição já foi feita antes e

se a imagem está salva em cache (do navegador). Se estiver salva, a requisição não será encaminhada para o servidor. Caso contrário, ela será encaminhada (Passo 2).

Passo 2 : Uma requisição é feita para a API, com os parâmetros de processamento da imagem, caso haja.

Passo 3 : O cache do servidor verifica se a a imagem requisitada está armazenada. Se estiver, a retorna para o usuário. Caso contrário, a requisição será encaminhada para pré-processamento (Passo 4).

Passo 4 : O servidor verifica no sistema de arquivos se existe uma cópia salva da imagem, seguindo os parâmetros requisitados. Se existir, retorna a imagem para o usuário. Caso contrário, a requisição será encaminhada para processamento (Passo 5).

Passo 5 : O servidor solicita ao sistema de arquivos a versão original da imagem. Caso haja processamento a ser feito, o servidor o realiza e salva uma cópia da imagem processada. A imagem é retornada ao usuário.

3.3.3 Gerenciamento e processamento das imagens

Ainda no lado do servidor, as imagens são armazenadas e processadas. O diagrama representado na Figura 3.8 ilustra como os elementos da Web API se relacionam para armazenar e processar imagens.

Para possibilitar o armazenamento e gerenciamento das imagens, alguns módulos serão necessários no lado do servidor. Como cada usuário terá seu próprio acesso ao sistema, é preciso que haja um componente responsável pela gestão de dados relacionados a eles.

Além disso, para promover a integridade do sistema, cada usuário possuirá credenciais compostas pelo endereço de e-mail e uma senha, que serão armazenados no banco de dados. A senha precisará ser armazenada de forma criptografada, para evitar o acesso indevido à informação em caso de vazamentos. Haverá ainda a possibilidade de criação de *tokens* de acesso dedicados à integração da API.

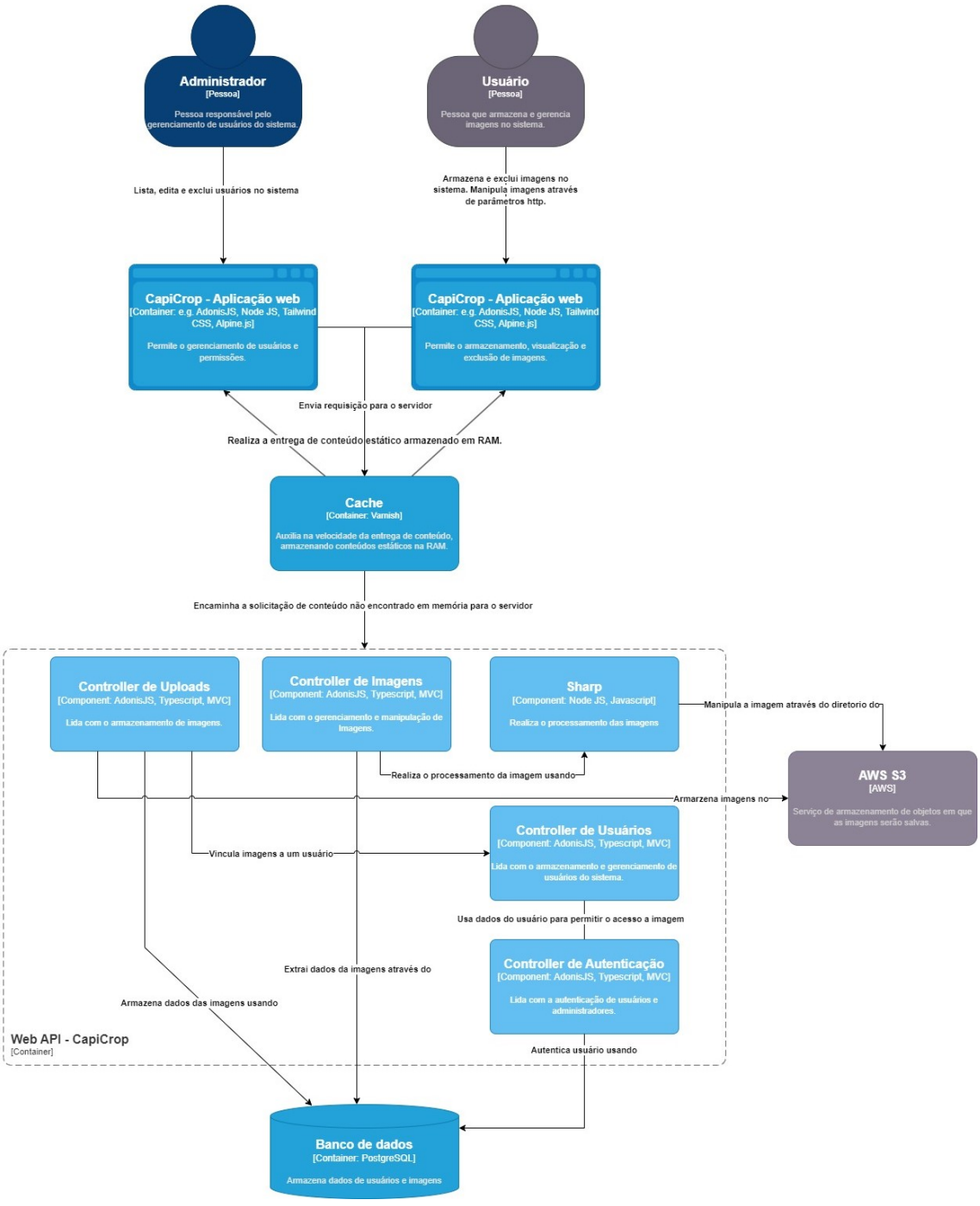

Figura 3.8: Diagrama C4 de componentes ilustrando a comunicação entre os elementos da Web API

Nesse contexto, também é fundamental que haja um componente responsável pela autenticação dos usuários, comparando as informações fornecidas durante o *login* com aquelas armazenadas no banco de dados.

Uma vez autenticado, o usuário terá acesso às imagens hospedadas, bem como a possibilidade de enviar outras. Esse processo de *upload* será gerenciado por um componente dedicado, responsável por armazenar as imagens no sistema de arquivos selecionado. Além disso, será persistido no banco de dados informações como a relação usuário-imagem e o diretório de armazenamento do arquivo. A figura 3.9 ilustra a relação entre as entidades do sistema.

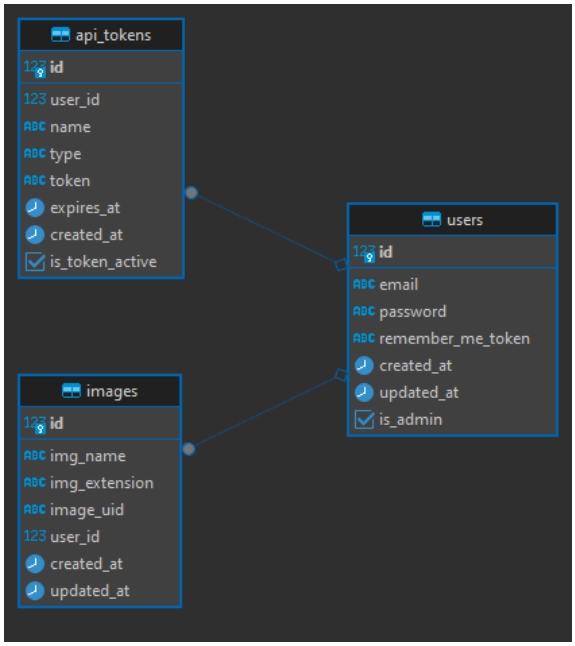

Figura 3.9: Diagrama entidade relacionamento do sistema CapiCrop utilizando notação UML

Para melhorar ainda mais a eficiência na entrega, um sistema de arquivos em nuvem, como Amazon S3[9] ou Google Cloud[10], será usado para o armazenamento das imagens, devido ao seu ótimo desempenho a um custo acessível. Além disso, esse tipo de arquitetura geralmente possui soluções como escalabilidade e confiabilidade, que permitem que o conteúdo seja disponibilizado de maneira rápida, armazenado com segurança e recuperado em casos de falha.

[9] <https://aws.amazon.com/pt/s3/>
[10] <https://cloud.google.com/>

Após o processo de *upload*, o gerenciamento das imagens será feito através de um componente dedicado a elas. Ele será responsável pela recuperação, exclusão e processamento da imagem.

A manipulação das imagens será feita em conjunto com uma biblioteca responsável pelo seu processamento. O componente de imagens enviará os parâmetros necessários para cada tipo de operação, como por exemplo o diretório da imagem no armazenamento do servidor, a altura, largura e outras instruções pertinentes. Após o processamento, o arquivo será armazenado temporariamente no servidor e seu diretório resultante será retornado para o componente das imagens, que exibirá a imagem para o usuário.

Por fim, será necessário que o servidor seja regido por uma rotina que, dado um período de tempo predefinido, realizará a limpeza do servidor, removendo os arquivos temporários resultantes de processos de manipulação das imagens. Essa prática garante a remoção de arquivos desnecessários, evitando o uso de armazenamento no serviço de armazenamento escolhido e, consequentemente, economizando recursos.

Capítulo 4

Desenvolvimento

Neste capítulo, serão discutidos os tópicos referentes à implementação da solução proposta. Na Seção 4.1 serão destacadas as principais tecnologias utilizadas durante o desenvolvimento deste trabalho. Já na Seção 4.2, serão abordados tópicos sobre o desenvolvimento, como padrão arquitetural, estrutura do projeto e comunicação de elementos do sistema.

4.1 Tecnologias utilizadas

O desenvolvimento do sistema proposto foi feito utilizando TypeScript em conjunto com o AdonisJS, um *framework* de desenvolvimento para Node.js. Para uma resposta mais rápida, foi utilizado o Varnish, uma ferramenta de cache que atua de maneira intermediária entre cliente e servidor, armazenando temporariamente em memória os conteúdos estáticos, nesse caso, as imagens. Também foram utilizadas bibliotecas adicionais no processo de desenvolvimento, que auxiliaram na implementação do projeto. Esse é o caso do Sharp, biblioteca que auxilia na manipulação de imagens, e o Lucid Slugify, que cria um identificador único para cada imagem.

4.1.1 Ambiente de desenvolvimento

O projeto foi desenvolvido usando o Visual Studio Code (vscode)[1], um editor de código-fonte desenvolvido pela Microsoft que possui suporte para TypeScript, plugins desenvolvidos pela comunidade e complementação inteligente do código. Tais funcionalidades contribuíram com a facilidade e agilidade durante o processo de desenvolvimento do projeto. O vscode possui integração total com o Github[2], uma plataforma de hospedagem de códigos que usa o Git[3] como sistema de versionamento de códigos-fonte e que foi utilizada para armazenamento e versionamento do código.

Para armazenar as imagens enviadas pelos usuários, foi escolhido o sistema de armazenamento de arquivos na nuvem da Amazon Web Services, o S3[4]. A escolha da ferramenta foi motivada pela alta escalabilidade, durabilidade e desempenho consistente do serviço, levando mais confiabilidade ao uso do CapiCrop.

Para o banco de dados, foi utilizado o PostgreSQL[5], um banco de dados objeto-relacional de código aberto. Para a manipulação do banco de dados, foi utilizado o DBeaver Community[6], uma ferramenta que tem como objetivo conectar e manipular várias bases de dados, incluindo o PostgreSQL. No contexto de banco de dados, foi utilizado o Lucid, um mapeador de objetos relacionais (ORM) do AdonisJS que aproxima o paradigma dos bancos de dados relacionais ao paradigma de orientação a objetos do TypeScript.

Para avaliar o funcionamento da API desenvolvida, bem como verificar suas respostas às solicitações, foi utilizado o *Thunder Client*[7], uma extensão integrada ao vscode. Essa ferramenta possui suporte a uma variedade de envio solicitações através de métodos HTTP, permitindo a análise das respostas da API diretamente do ambiente de desenvolvimento.

[1] <https://code.visualstudio.com/>
[2] <https://github.com/>
[3] <https://git-scm.com/>
[4] <https://aws.amazon.com/pt/s3/>
[5] <https://www.postgresql.org/>
[6] <https://dbeaver.io/about/>
[7] <https://www.thunderclient.com/>

4.1.2 TypeScript

Por muito tempo, as páginas web eram majoritariamente estáticas e ofereciam aos usuários uma experiência limitada e sem grande interatividade. Porém, com a introdução do JavaScript, esse cenário teve uma transformação significativa, permitindo que as páginas seguissem um formato mais dinâmico, contando com a interação dos usuários em tempo real.

O JS é uma linguagem de programação interpretada amplamente empregada na criação de websites modernos, considerada praticamente onipresente. A criação da linguagem se deu por conta da necessidade em atender às demandas criadas a partir da modernização do *Front-End*, que permitiu ocultar e mostrar elementos nas páginas web por meio da manipulação do DOM. Mantendo-se como a linguagem mais utilizada pelos desenvolvedores[8], o JS sustenta sua popularidade ao longo dos anos devido a sua versatilidade, uma vasta comunidade de desenvolvedores e frameworks consolidados, como o React e Angular (FLANAGAN, 2012). Na linguagem, o processamento dos *scripts* é feito do lado do cliente, ou seja, o navegador do cliente processa os códigos em vez do servidor.

O TypeScript[9] é uma extensão do JS, sendo definida também como uma linguagem de programação de código aberto desenvolvida pela Microsoft. O funcionamento do Typescript se diferencia do JavaScript comum pois adiciona recursos como a tipagem específica - ou seja, variáveis podem ser tipadas - e também interfaces.

4.1.3 Node.js

A popularização do JavaScript foi tanta, que Ryan Dahi idealizou um *framework* que interpreta o JS diretamente do servidor. Nesse contexto surge o Node.js[10], um ambiente de desenvolvimento de código aberto e multiplataforma para JavaScript, que concretiza seu uso como linguagem utilizada tanto do lado do cliente como do lado do servidor. Com isso, elimina-se a necessidade de ter um navegador ativo para

[8] <https://octoverse.github.com/2022/top-programming-languages>
[9] <https://www.typescriptlang.org/pt/>
[10] <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>

interpretar os códigos do JavaScript, simplificando para apenas a instalação do Node, que irá interpretar os códigos do lado do servidor através de uma *engine* do Google , a V8 (CANTELON et al., 2014).

Uma característica importante do Node.js é a forma assíncrona de processamento de requisições. Isso permite que ele execute várias tarefas ao mesmo tempo, sem que o servidor seja bloqueado, o que faz do *framework* uma boa escolha para aplicações que precisam lidar com grande número de solicitações simultâneas.

4.1.4 AdonisJS

O Node.js facilitou na construção das Web APIs, permitindo que o desenvolvimento fosse feito de forma bem estruturada. Contudo, mesmo com sua introdução, grande parte da responsabilidade do desenvolvimento ainda ficou com o programador, o que gera certa dificuldade. Nesse cenário surge o AdonisJS. Em sua documentação[11], o AdonisJS é definido como um *framework* de *back-end* que auxilia na criação de páginas web e APIs. Ele pode apoiar os desenvolvedores Web em uma gama de etapas, como lidar com requisições HTTP, consultar bases de dados, autenticar usuários, enviar e-mails, etc, fazendo dele um *framework* completo para desenvolvimento do projeto.

4.1.5 Varnish

O Varnish[12] é definido como uma ferramenta intermediária entre cliente e servidor, que funciona como um Proxy HTTP reverso, ou seja, recebe as solicitações dos clientes e as redireciona a servidores internos. O uso do Varnish é importante para o armazenamento temporário, *i.e.* cache, no lado do servidor, que armazena temporariamente as requisições para que, caso uma requisição com os mesmos parâmetros seja refeita pelo cliente, em vez de reencaminhá-la para o servidor interno, o conteúdo referente à requisição é encaminhado diretamente ao usuário.

[11] <https://adonisjs.com/adonisjs-at-a-glance>
[12] <https://varnish-cache.org/>

4.1.6 Bibliotecas

Além das ferramentas supracitadas, foram utilizadas algumas outras que auxiliaram consideravelmente no processo de desenvolvimento do projeto. Uma das principais ferramentas utilizadas foi o Sharp[13], um módulo para Node.js voltado para a manipulação de imagens. Ele é capaz de redimensionar, recortar, aplicar filtros e converter formatos das imagens.

Outra ferramenta fundamental para o desenvolvimento foi o Lucid Slugify[14], uma ferramenta para o AdonisJS que permite a criação de cadeias de caracteres aleatórias únicas, utilizada para complementar o identificador das imagens.

Adicionalmente, foram utilizadas ferramentas para facilitar o desenvolvimento das telas pelas quais os usuários interagem com a aplicação. O Alpine.js[15] é um *framework* para JS desenvolvido para facilitar a criação da interatividade em páginas Web, permitindo a manipulação do DOM em curtos trechos de código. Houve ainda o uso do Tailwind[16], ferramenta altamente customizável que facilita a criação de designs sofisticados por meio de classes CSS predefinidas.

4.2 Desenvolvimento

Com base nos requisitos elucidados no capítulo anterior, esta seção fornece uma visão detalhada do processo de desenvolvimento do sistema CapiCrop. Com o objetivo de garantir o melhor formato de explicação, inicialmente será abordado a estruturação aplicada ao projeto do sistema. Em seguida, o desenvolvimento será discutido seguindo o fluxo de uso padrão esperado, começando pelo primeiro contato do usuário com o sistema - ou seja, o processo de cadastro - até a recuperação e manipulação de imagens.

[13] <https://sharp.pixelplumbing.com/>
[14] <https://github.com/adonisjs/lucid-slugify>
[15] <https://alpinejs.dev/>
[16] <https://tailwindcss.com/>

4.2.1 Organização do código e padrão arquitetural

Como mencionado anteriormente, o AdonisJS foi uma das tecnologias integradas ao sistema. Dentre a enorme gama de facilidades proporcionadas pelo *framework*, destaca-se o fornecimento de uma estrutura predefinida para organização do projeto. Isso se estende até mesmo ao *front-end*, devido a integração com uma *engine* que possibilita a criação de *templates* e renderizações que agilizam o desenvolvimento de interfaces.

Nesse contexto, considerando que um dos objetivos do CapiCrop se trata do fornecimento de uma interface para armazenamento e gerenciamento de imagens ao usuário, dentre as estruturas fornecidas pelo AdonisJS, a escolhida para o sistema foi a *web*. Sendo assim, as rotas desenvolvidas para o sistema interagem tanto com *views* quanto com *controllers*. A figura 4.1 representa a estrutura de pastas em que se organiza o sistema.

Para divisão de responsabilidade de cada componente do código, foi escolhido o padrão arquitetural *Model-View-Controller* (MVC). Suas características conceituais são simples, fator que possibilita facilidade de aplicação, manutenção e escalabilidade do código a longo prazo, tornando descomplicada a aplicação de novas funcionalidades no sistema.

Esse padrão é composto por três camadas que regem a divisão do código: *Model*, *View* e *Controller*. A camada *Model* se relaciona com a gestão de dados e regras de negócio do sistema, enquanto a *View*, representa a camada de interação com o usuário - ou seja, a interface -, e por fim, a *Controller* atua intermediando as interações entre as duas camadas anteriores (DEACON, 2009).

Em suma, no contexto do CapiCrop, a *Model* engloba as classes do sistema, como a imagem e os usuários, e aplicação de suas regras de negócio, sendo a única camada a se comunicar com o banco de dados. Já a *View* representa o *front-end* do sistema, por onde os usuários interagem com a aplicação. Por fim, o *Controller* atua como um orquestrador responsável por realizar a troca de informações entre essas duas, solicitando dados à *Model*, e formatando-os para retorno apropriado para a *View*.

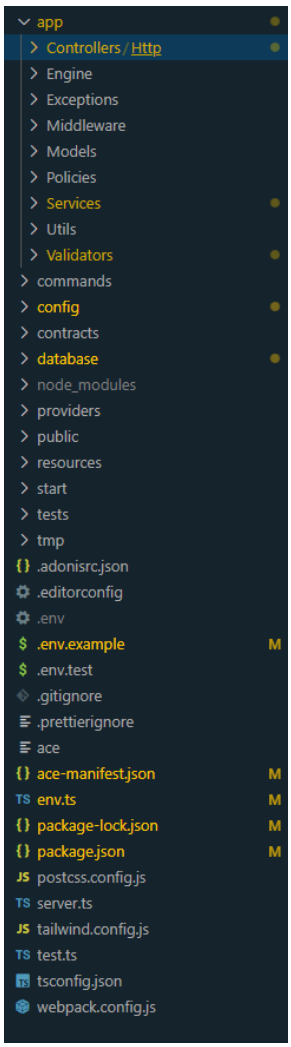

Figura 4.1: Estrutura de pastas do projeto do sistema CapiCrop

4.2.2 Cadastro do usuário e autenticação

Ao acessar o sistema pela primeira vez, o usuário, ainda não registrado, deverá criar uma nova conta. Para isso, foi desenvolvido um formulário de cadastro onde o usuário deve informar seu e-mail e senha. Uma vez cadastrado, o usuário pode acessar a aplicação através do formulário de *login* e armazenar imagens. A figura 4.2 ilustra a interface desenvolvida para autenticação.

O AdonisJS permite uma fácil integração de métodos de autenticação, possibilitando que grande parte desse processo seja abstraído pelo desenvolvedor. O AdonisJS *Web Guard*[17] é uma ferramenta que permite armazenar as informações

[17]<https://docs.adonisjs.com/guides/auth/web-guard>

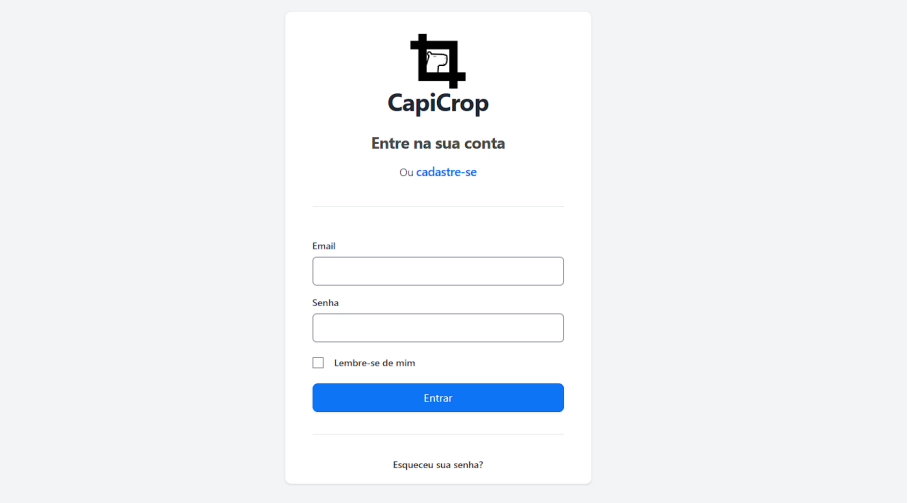

Figura 4.2: Tela de Autenticação do sistema CapiCrop

de autenticação do usuário em seu navegador por meio de *cookies* ou sessões. Essa ferramenta foi utilizada para a autenticação do usuário na aplicação web, ou seja, por onde os usuários gerenciam as imagens salvas.

Além da autenticação para a aplicação Web, foi preciso desenvolver uma outra forma de identificar o usuário em casos de integração da API a códigos ou aplicações externas, que serão autenticados pelo AdonisJS *API Guard*[18]. Como a melhor forma de autenticação, nesse caso, é o uso de chaves para a API[19], são gerados *tokens* do tipo *JSON Web Token* (JWT) para esse fim.

Foi desenvolvida uma página por onde os usuários podem gerar novos *tokens* ou acessar aqueles já criados em seu nome por meio da aplicação web, representada pela Figura 4.3.

4.2.3 Upload e armazenamento de imagens

Uma vez registrado, o usuário é liberado para fazer *upload* de imagens. Isso pode ser feito através da interface web, ou através de chamadas para API. Ao receber uma requisição para a rota de *upload* de imagens, alguns dados são armazenados no servidor, são esses: o id da imagem, nome, extensão do arquivo, o identificador único usado para requisitar a imagem e a chave estrangeira referente ao id do usuário

[18] <https://docs.adonisjs.com/guides/auth/api-tokens-guard>
[19] <https://cloud.google.com/endpoints/docs/openapi/when-why-api-key?hl=pt-br>

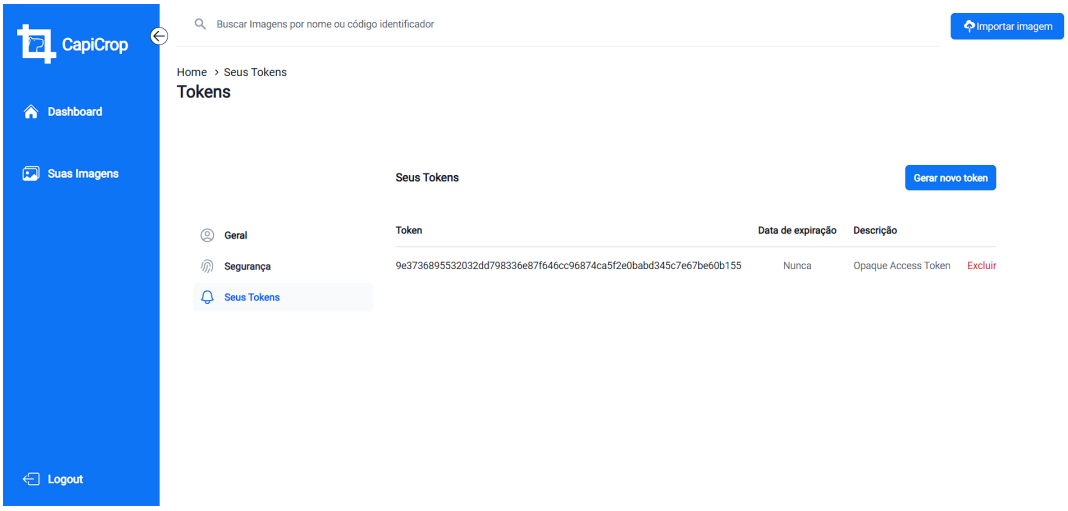

Figura 4.3: Tela de gerenciamento de *tokens*.

a qual ela pertence. A figura 4.4 ilustra a interface de upload e gerenciamento de imagens.

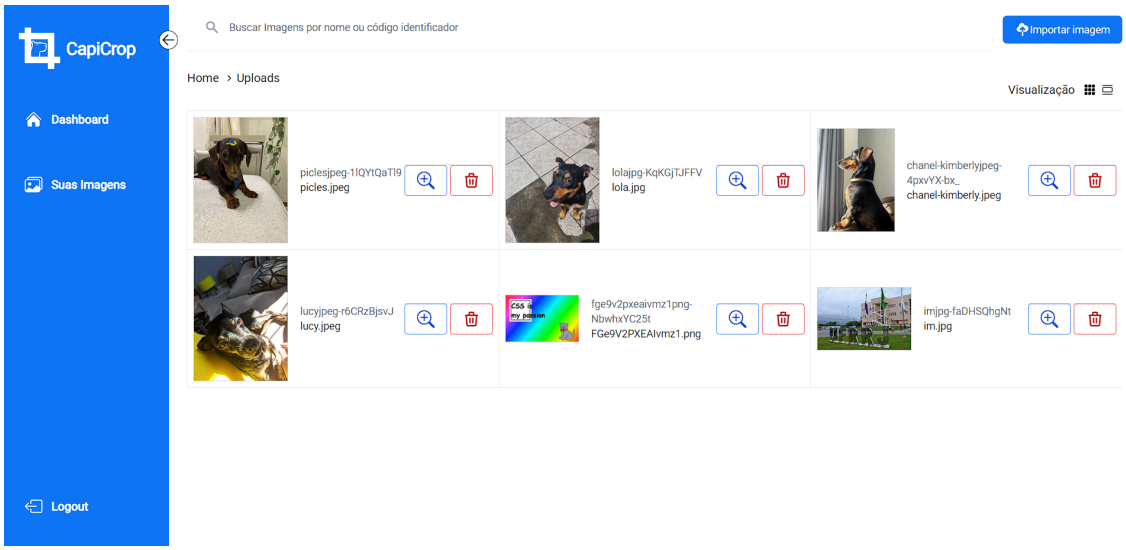

Figura 4.4: Tela de upload e gerenciamento de imagens.

Como abordado no capítulo anterior, um dos requisitos do sistema é ter suporte aos principais formatos de imagem, aumentando a compatibilidade da ferramenta. Dessa forma, os formatos suportados serão JPEG, PNG, WebP, AVIF, TIFF, e SVG, também suportados pelo Sharp. A informação do formato da imagem também é salva no banco de dados, juntamente a outras informações da imagem.

Para gerar o identificador único da imagem, foi utilizada a biblioteca *Lucid Slugify*.

Essa biblioteca permite a geração de cadeias de caracteres randômicos durante o tempo de armazenamento do banco de dados. Dessa maneira, é evitado que a mesma rota seja associada a duas imagens diferentes, já que o protocolo HTTP será usado para realizar as manipulações de imagens. Além disso, aumenta a dificuldade de crawlers recuperarem os arquivos.

Por fim, após o armazenamento dos dados no banco, a imagem é armazenada no sistema de arquivos da S3. A integração do CapiCrop ao serviço da Amazon foi feita pelo próprio Adonis, por meio do Adonis Drive[20]. Essa abordagem permitiu a abstração no uso do serviço em nuvem, de forma que o salvamento dos arquivos na S3 ocorresse de maneira tão transparente quanto num sistema de arquivos local.

Para possibilitar a recuperação em solicitações futuras, foi estruturado o seguinte padrão de diretório para a imagem:

```
[root]/[identificador-unico-da-imagem]/[nome-da-imagem].[extensão]
```

Onde, *root* é a pasta raiz de *uploads*. A figura 4.5 ilustra a estrutura de armazenamento das imagens em pastas no sistema de arquivos da S3.

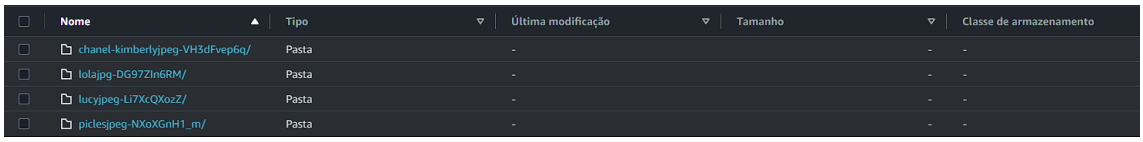

Figura 4.5: Exemplo de diretório em que as imagens são armazenadas no S3.

4.2.4 Manipulação de imagens

Após o armazenamento da imagem, o usuário pode exibi-la a qualquer momento através do envio de parâmetros seguindo o padrão discutido no capítulo anterior. Essa funcionalidade foi possibilitada graças ao uso da biblioteca de processamento de imagens *sharp* apresentada na Seção 4.1.6.

Desse modo, entre os principais motivos que regem a escolha da biblioteca para a aplicação, encontra-se não somente a facilidade de uso, como também o fato de ser

[20] <https://docs.adonisjs.com/guides/drive>

desenvolvida em Node JS, possibilitando uma descomplicada integração ao sistema. Além disso, o sharp possui um ótimo desempenho na manipulação de imagens em larga escala, graças a sua base composta por bibliotecas desenvolvidas em C++ [21].

A partir do uso da biblioteca, foram definidos 4 métodos de manipulação de imagem: *resize*, *crop*, *rotate* e *color*, cada um possuindo uma quantidade diferente de parâmetros necessários. Por exemplo: enquanto o método *resize* precisa de largura e altura para gerar a imagem redimensionada (levando em consideração a proporção mais próxima), o *rotate* requer apenas o ângulo em que a imagem será rotacionada.

Além disso, para permitir que múltiplas operações pudessem ser requisitadas para uma imagem na mesma solicitação, foram desenvolvidos alguns métodos que realizam o tratamento da cadeia de caracteres da URL, transformando cada operação e seus parâmetros em um *array* a ser iterado posteriormente para realizar as transformações chamando os respectivos métodos do sharp.

Por fim, após o tratamento dos parâmetros recebidos via URL e invocação das operações necessárias utilizando a biblioteca, foi obtida uma ferramenta funcional de manipulação de imagens. As figuras a seguir exemplificam o uso da API desenvolvida, com a imagem original à esquerda e a imagem processada à direita. A figura 4.6 exemplifica o uso com o parâmetro de redimensionamento da imagem. A figura 4.7 exemplifica o uso da ferramenta com os parâmetros de rotação e redimensionamento da imagem. Já a figura 4.8 exemplifica o uso da ferramenta com processamento de redimensionamento e aplicação de filtros de cor à imagem. Por fim, a figura 4.9 exemplifica o uso da ferramenta com redimensionamento e recorte da imagem a tamanhos pequenos, ideal para uso em ícones.

[21] <https://sharp.pixelplumbing.com/>

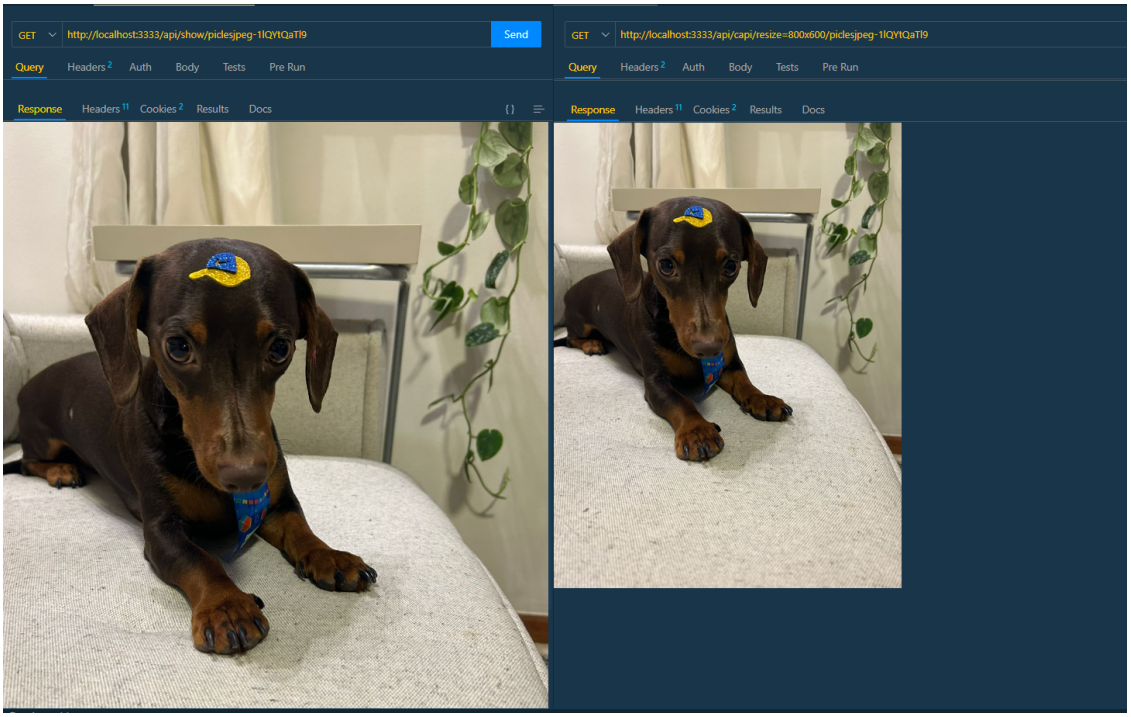

Figura 4.6: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=800x600 (à direita).

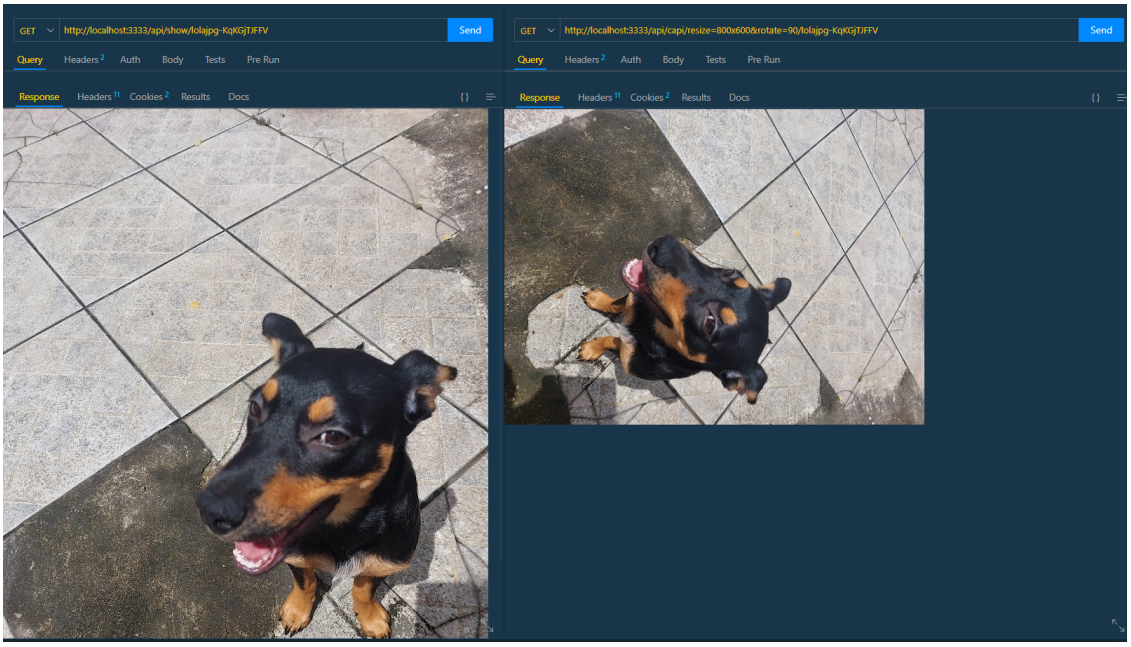

Figura 4.7: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=800x600 e *rotate*=90 (à direita).

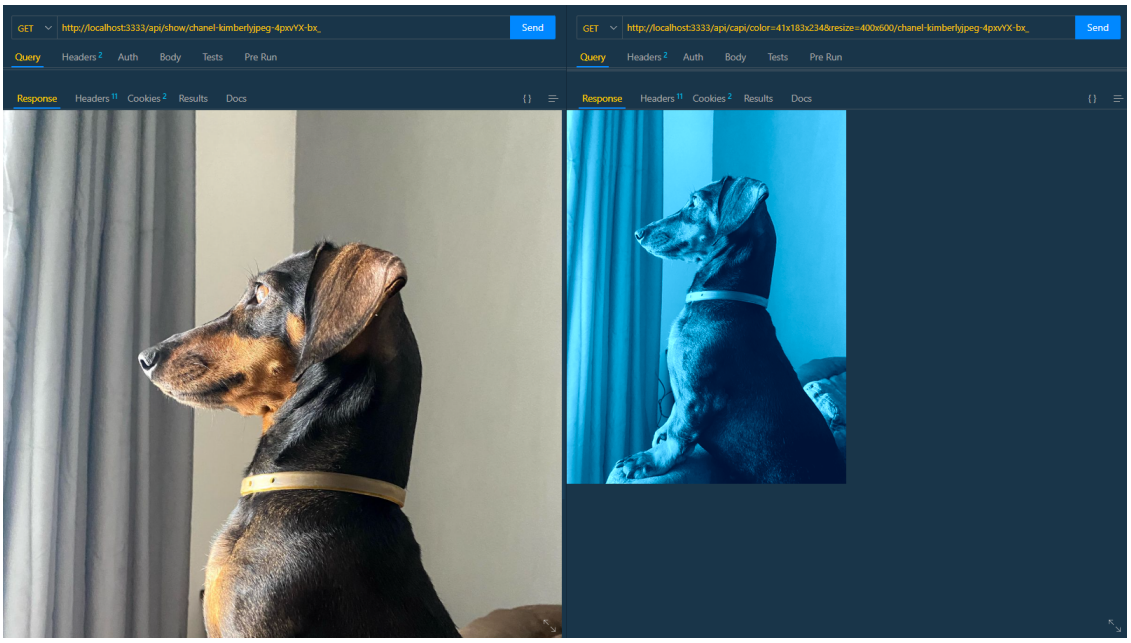

Figura 4.8: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=400x600 e *color*=41x183x234 (à direita).

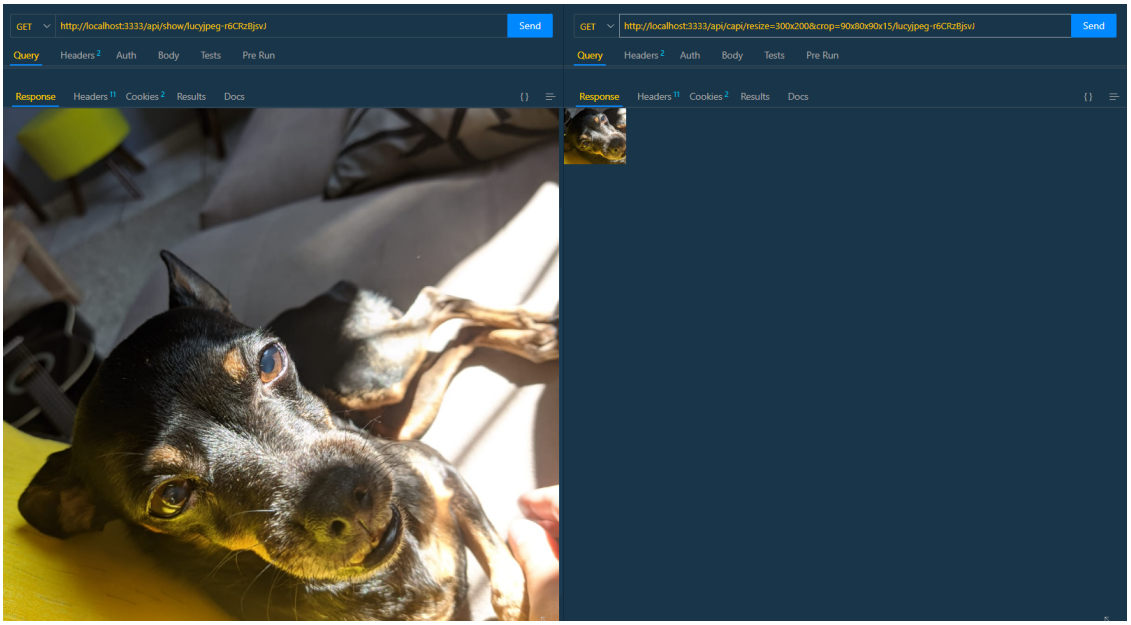

Figura 4.9: Exemplo da exibição de uma imagem original armazenada no sistema (à esquerda), e a mesma imagem após o envio de parâmetros de *resize*=300x200 e *crop*=90x80x90x15 (à direita).

Capítulo 5

Conclusão

Neste capítulo serão feitas as considerações finais sobre a solução proposta. Na Seção 5.1 serão abordados os pontos positivos, considerando os diferenciais do CapiCrop em relação aos trabalhos relacionados. Por fim, na Seção 5.2 serão abordados os pontos de melhoria encontrados durante o desenvolvimento, bem como os próximos passos envolvendo a proposta.

5.1 Considerações finais

As imagens são elementos importantes em aplicações web e seu uso pode gerar aumento no engajamento e permanência nos sites. A entrega eficiente das imagens aos usuários é uma questão importante, pois quanto maior o tempo de carregamento, maior será a insatisfação deles. Tendo isso em vista, nesse projeto foi desenvolvida uma solução de sistema voltado para a manipulação de imagens, que facilita a entrega de imagens aos usuários.

A solução implementada se destaca na facilidade do uso pelos usuários através da implementação de uma interface. Levando em consideração os projetos relacionados, a solução une a proposta de código aberto, encontrada em ferramentas como o Thumbor, a questões de eficiência na entrega através do uso de cache, presente nas propostas pagas apresentadas. Com isso, a solução se caracteriza pela junção dos

pontos fortes presentes nos projetos similares.

5.2 Limitações e trabalhos futuros

Durante o desenvolvimento do CapiCrop, foi observado que a implementação de uma ferramenta completa, que resolva os problemas relacionados ao uso de imagens em aplicações web de forma eficiente, seria um grande desafio para a solução proposta. Apesar dos pontos positivos notados, existem melhorias a serem incrementadas na solução que atendem com maior abrangência os problemas observados nesse contexto.

Uma melhoria significativa envolve a implementação de uma CDN, uma rede distribuída especializada na entrega eficiente de conteúdos, já que era uma das ideias originais desse projeto. Contudo, em um contexto mais reduzido, a implementação desse tipo de arquitetura se torna um desafio por conta da relação custo-benefício, complexidade na configuração geográfica dos servidores e restrições no escopo do projeto, incluindo o tempo de desenvolvimento.

Além disso, nas condições atuais, o CapiCrop armazena as imagens diretamente no sistema de arquivos do S3, da Amazon, restringindo seu uso a esse ambiente. No entanto, a integração com contas do próprio usuário em serviços de armazenamento em nuvem, como a própria S3, Google Cloud[1], Microsoft Azure[2], etc., pode se tornar outro grande diferencial da proposta em relação as ferramentas relacionadas.

Uma melhoria contínua a ser implementada na proposta desenvolvida é a incorporação de novas funcionalidades de manipulação, o que trará somente benefícios ao uso da ferramenta, especialmente com a facilidade de incorporá-las ao sistema. Não é possível antecipar com precisão quais ferramentas de manipulação de imagens serão usadas com mais frequência pelos desenvolvedores, pois isso varia conforme as necessidades individuais. A implementação, atualmente, inclui os recursos mais básicos, como recorte, redimensionamento, rotação e aplicação de filtros.

Outra melhoria que representa uma grande evolução em termos de otimização,

[1] <https://cloud.google.com/>
[2] <https://azure.microsoft.com/>

desempenho e compatibilidade é a implementação da capacidade de modificação do formato de imagens. A variação no tamanho dos arquivos devido a compressão presente em alguns formatos de imagem é uma consideração a ser feita, já que não só economizaria recursos de armazenamento do servidor, mas também contribui para a redução no tempo de entrega. Além disso, há formatos de imagem que não são reconhecidos em alguns dispositivos, o que é solucionado apenas na troca de formato do arquivo.

Algumas melhorias em relação à segurança do sistema podem ser implementadas, uma vez que ela depende atualmente de recursos como o *Adonis Guard*[3] para a autenticação dos usuários e o *Adonis Shield*[4], que protege o servidor contra alguns tipos de ataque. Além disso, já foi implementada uma solução de transformação do nome da imagem através da adição de um código aleatório, dificultando a previsibilidade dos endereços e, consequentemente, o uso de *crawlers* para obter os conteúdos hospedados. Funcionalidades como a possibilidade de tornar imagens privadas, que só são exibidas ou manipuladas mediante autenticação, incrementariam ainda mais a segurança no sistema proposto.

Por fim, há ainda a possibilidade incluir novas funcionalidades ao serviço, que facilitam no gerenciamento e organização das imagens, como a implementação de tags de organização e álbuns compartilhados, permitindo agrupar imagens com características semelhantes. Além disso, também é possível implementar níveis de permissão para grupos de usuários, fator que seria útil para contextos onde é necessário organizar usuários de acordo com a permissão.

[3] <https://docs.adonisjs.com/guides/auth/web-guard>
[4] <https://docs.adonisjs.com/guides/security/web-security>

Referências

BENDELL, C. et al. *High performance images: Shrink, load, and deliver images for speed*. [S.l.]: O'Reilly Media, 2016.

BIEHL, M. *API Architecture*. [S.l.]: API-University Press, 2015. v. 2.

CANTELON, M. et al. *Node. js in Action*. [S.l.]: Manning Greenwich, 2014.

CECCONI, F. R. et al. Digital asset management. *Digital transformation of the design, construction and management processes of the built environment*, Springer International Publishing, p. 243–253, 2020.

DEACON, J. Model-view-controller (mvc) architecture. *[Online][Citado em: 10 de março de 2023.]*, v. 28, 2009. <http://www.jdl.co.uk/briefings/MVC.pdf>.

FELIZARDO, A.; SAMAIN, E. A fotografia como objeto e recurso de memória. *Discursos fotográficos*, v. 3, n. 3, p. 205–220, 2007.

FLANAGAN, D. *JavaScript: o guia definitivo*. [S.l.]: Bookman Editora, 2012.

JEHL, S.; MARCOTTE, E. *Responsible Responsive Design*. A Book Apart, 2014. (Book Apart : brief books for people who make websites). ISBN 9781937557164. Disponível em: <https://books.google.com.br/books?id=M44prgEACAAJ>.

MELO, C. A. et al. Software como serviço: Um modelo de negócio emergente. *Paper Centro de Informática–Universidade Federal de Pernambuco (UFPE)*, p. 27, 2007.

MUNYARADZI, Z.; MAXMILLAN, G.; AMANDA, M. N. Effects of web page contents on load time over the internet. International Journal of Science and Research (IJSR), 2013. Disponível em: <https://www.ijsr.net/archive/v2i9/MzAwODEzMDM=.pdf>.

NIELSEN, J. *Website Response Times*. 2010. <https://www.nngroup.com/articles/website-response-times/> [Acesso em: 28 de junho de 2023].

RODRIGUES, R. C. Análise e tematização da imagem fotográfica. *Ciência da informação*, SciELO Brasil, v. 36, p. 67–76, 2007.

SALHI, H. et al. Benchmarking and performance analysis for distributed cache systems: A comparative case study. In: _____. [S.l.: s.n.], 2018. p. 147–163. ISBN 978-3-319-72400-3.

SINGJAI, A. et al. Patterns on designing api endpoint operations. October 2021. Disponível em: <http://eprints.cs.univie.ac.at/7194/>.

SöDERMARK, O. Client-side caching: Reducing server load and latency in a network traffic analysis tool. *Umeå universitet*, 2023.

WAN, S.; LI, Y.; SUN, K. PathMarker: protecting web contents against inside crawlers. *Cybersecurity*, Springer Science and Business Media LLC, v. 2, n. 1, fev. 2019. Disponível em: <https://doi.org/10.1186/s42400-019-0023-1>.

YORK, R. *Web development with jQuery*. [S.l.]: John Wiley & Sons, 2015.

ZHANG, N. et al. Web apis: Features, issues, and expectations – a large-scale empirical study of web apis from two publicly accessible registries using stack overflow and a user survey. *IEEE Transactions on Software Engineering*, v. 49, n. 2, p. 498–528, 2023.