

**GABRIEL HENRIQUE JUSTINO RIBEIRO  
GABRIELA VON STAA GUEDES  
LUANA VICENTE LEITE  
THIAGO FAGUNDES PINHO**

**PetFinder - Plataforma de auxílio a adoção e identificação de animais  
perdidos utilizando Reconhecimento De Imagem**

**São Paulo  
2020**

**GABRIEL HENRIQUE JUSTINO RIBEIRO  
GABRIELA VON STAA GUEDES  
LUANA VICENTE LEITE  
THIAGO FAGUNDES PINHO**

**PetFinder - Plataforma de auxílio a adoção e identificação de animais perdidos utilizando Reconhecimento De Imagem**

Dissertação apresentada à Escola  
Politécnica da Universidade de São  
Paulo para obtenção do título de  
Engenheira e Engenheiro Eletricista  
com Ênfase em Computação

Área de Concentração:  
Engenharia de Software

Orientador: Prof. Dr. Reginaldo Arakaki

**São Paulo  
2020**

## RESUMO

O projeto tem duas frentes principais: a primeira, focada na melhoria da situação atual de adoção de animais no país para os dois elementos envolvidos, facilitando a obtenção de informação para o lado do adotante e o avanço na conexão para o lado do doador, buscando aumentar a efetividade e o alcance de abrigos de animais. Para realização deste objetivo, foi idealizado um sistema em que os doadores pudessem exibir os animais disponíveis, centralizando as informações de diversos abrigos e permitindo que os adotantes tivessem um canal sem atrito para busca e análise de possíveis futuros mascotes. A outra frente visa utilizar reconhecimento de imagem para analisar a semelhança entre fotos de animais encontrados na rua, providas pelos usuários da plataforma, e uma foto principal, provida pelo dono de um animal que possivelmente está perdido. Nesta frente, pretendemos também utilizar de outras características disponibilizadas pelos usuários que cadastraram esses cachorros encontrados e perdidos, para fazer uma busca mais precisa a fim de encontrar o animal correto rapidamente. Como plataforma de trabalho, foi escolhido o formato de aplicativo, visando a facilidade de conexão com os usuários.

Palavras-Chave: Adoção de animais. Abrigo de animais. Animais perdidos. Análise de imagem. Comparação de imagem. Object Detection. Foreground Extraction. Software. Mobile. Android. React Native. Python. Flask. SQLAlchemy.

## **ABSTRACT**

The project has two main verticals: the first one is focused on developing the current situation for pet adoption in Brazil for its two main stakeholders, producing a smoother information obtaining process for those interested in adopting and improving shelter's communication and connection with them, providing better reach and efficiency. For that goal, this system was idealized, in which donors could exhibit available pets and possible adopters had a seamless channel to search for potential pets. The second vertical intends to use image analysis tools to determine resemblance between user-inputed photos of pets they lost and user-inputed photos of lost pets found in the streets. Here, it is also intended to use other physical characteristics provided in the pets' registrations to make a more precise search engine, improving efficiency in finding the correct pet. The platform selected for the project was the app format, intending to provide better user adoptance.

**Keywords:** Pet adoption. Pet shelters. Lost pets. Image analysis. Image comparison. Object Detection. Foreground Extraction. Software. Mobile. Android. React Native. Python. Flask. SQLAlchemy.

# SUMÁRIO

<b>1 INTRODUÇÃO</b>	<b>7</b>
1.1 OBJETIVO	7
1.2 MOTIVAÇÃO	8
1.3 JUSTIFICATIVA	9
1.4 ORGANIZAÇÃO DO TRABALHO	10
<b>2 ASPECTOS CONCEITUAIS</b>	<b>12</b>
2.1 DIAGRAMA DE OSTERWALDER	12
2.2 ADOÇÃO DE PETS	13
2.3 NORMAS E PADRONIZAÇÃO	14
2.3.1 Arquitetura de Sistemas de Software	14
2.3.2 Qualidade de Sistemas de Software	15
2.4 PLATAFORMAS CLOUD	15
2.5 ALGORITMOS DE TRATAMENTO E COMPARAÇÃO DE IMAGENS	16
2.6 PRIVACIDADE DE DADOS	18
<b>3 TECNOLOGIAS UTILIZADAS</b>	<b>19</b>
3.1 PYTHON	19
3.2 REACT NATIVE	20
3.3 FLASK SQLALCHEMY	20
3.4 RECONHECIMENTO E COMPARAÇÃO DE IMAGENS	20
3.4.1 Vize.ai	21
3.4.1.1 Reconhecimento de Imagem	22
3.4.2 Algoritmo próprio	26
3.4.2.1 YOLO Object Detection	26
3.4.2.2 OpenCV GrabCut: Foreground Segmentation e Extraction	28
3.4.2.3 Comparação por cores	30

3.5 S3	31
3.6 GOOGLE MAPS SDK FOR ANDROID	31
3.7 API DE SEARCH E FORWARD GEOCODING DA LOCATIONIQ	31
3.8 TECNOLOGIAS UTILIZADAS PARA TESTES E DESENVOLVIMENTO	32
3.8.1 Android Studio	32
3.8.2 Postman	32
3.8.3 MarvelApp	33
3.9 EC2	33
<b>4 METODOLOGIA DE TRABALHO</b>	<b>35</b>
4.1 REPOSITÓRIO GIT	35
4.2 METODOLOGIA ÁGIL	37
4.3 JIRA	39
4.4 PESQUISA DE USUÁRIOS	41
4.4.1 Stakeholders	41
4.4.2 Papéis e variáveis de perfil	42
4.4.3 Necessidades	43
4.4.4 Instrumentos	44
4.4.5 Amostra	45
4.4.6 Consentimento	46
4.4.7 Resultados	47
4.5 DISCIPLINAS ACELERADORAS	50
<b>5 ESPECIFICAÇÃO DE REQUISITOS DO SISTEMA</b>	<b>51</b>
5.1 ADOÇÃO DE ANIMAIS	51
5.1.1 Fluxo de cadastro de animal para adoção	52
5.1.2 Fluxo de busca por animal para adoção	52
5.2 ENCONTRAR ANIMAIS PERDIDOS	53
5.2.1 Fluxo de cadastro de animal encontrado	53

5.2.2 Fluxo de cadastro de animal perdido	54
5.3 REQUISITOS	55
5.3.1 Requisitos Funcionais	55
5.3.1.1 Prioridade alta	56
5.3.1.2 Prioridade média	57
5.3.1.3 Prioridade baixa	58
5.3.1.4 Diagramas	59
5.3.2 Requisitos Não Funcionais	61
5.3.2.1 Segurança	63
5.3.2.2 Usabilidade	64
5.3.2.3 Disponibilidade	65
5.4 PROTÓTIPOS	66
<b>6 PROJETO E IMPLEMENTAÇÃO</b>	<b>71</b>
6.1 PET PERDIDO	71
6.1.1 Usuário	71
6.1.1.1 Página Inicial	71
6.1.1.2 Cadastro de novo usuário	72
6.1.1.3 Tela inicial de usuário	74
6.1.2 Cadastro de animal	75
6.1.3 Match de animais	83
6.1.3.1 Match por características	83
6.1.3.2 Match por imagem (Vize)	85
6.1.3.3 Match por imagem (algoritmo próprio)	87
6.1.4 Histórico	89
6.2 PET PARA ADOÇÃO	92
6.2.1 Usuário comum	92
6.2.1.1 Pesquisa por animais para adoção	93

6.2.2 Abrigos	94
6.2.2.1 Tela de login do abrigo	94
6.2.2.2 Cadastro de novo abrigo	94
6.2.2.3 Tela inicial do abrigo	96
6.2.2.4 Visualização e atualização do perfil	96
6.2.2.5 Cadastro de animal para adoção	97
6.2.2.6 Histórico	99
6.3 SEGURANÇA	100
6.3.1 Habilitação de portas no servidor e segurança	101
6.3.2 Certbot	102
6.3.3 Nginx	102
6.3.4 Log	103
6.3.5 Armazenamento de senhas com hash	104
6.4 GEOLOCALIZAÇÃO	104
6.5 SPRINTS REALIZADAS	107
6.6 DIAGRAMA ARQUITETURAL DO SISTEMA	111
<b>7 TESTES E AVALIAÇÃO</b>	<b>113</b>
7.1 ALGORITMO DE ANÁLISE DE IMAGEM	113
7.1.1 Preparação das imagens	113
7.1.1.1 Object Detection	114
7.1.1.2 Foreground Extraction	115
7.1.1.3 Resultados	118
7.1.2 Comparação das imagens	119
<b>8 CONSIDERAÇÕES FINAIS</b>	<b>124</b>
8.1 CONCLUSÕES DO PROJETO DE FORMATURA	124
8.2 CONTRIBUIÇÕES	126
8.3 PERSPECTIVAS DE CONTINUIDADE	126



**REFERÊNCIAS** **129**

**APÊNDICE A - Termo de Consentimento** **137**

## 1 INTRODUÇÃO

Neste capítulo, são introduzidos os primeiros conceitos deste trabalho, como seu objetivo, motivação e justificativa.

### 1.1 OBJETIVO

O objetivo do projeto é a construção de um sistema computacional com arquitetura distribuída envolvendo recursos de *cloud*, base de dados e conexão à internet, acessados por *smartphones*. Essa aplicação está vinculada a serviços de conveniência de animais de estimação, com foco em adoção e na busca por animais perdidos.

Esse sistema, batizado de PetFinder, foi idealizado como uma junção de duas plataformas distintas:

1. Agregador de informações para identificação de animais perdidos.
2. Motor de pesquisa de animais em abrigos para adoção.

Cada plataforma tem seu foco principal, mas dada a intersecção de parte dos requisitos foi possível juntá-las numa única plataforma com duas verticais. Em relação à vertical de identificação de animais perdidos, o foco é a análise de imagens de animais submetidas por usuários, com paralela classificação baseada nas características descritas durante o cadastro e posterior tentativa de correspondência com animais encontrados na rua ou perdidos, já cadastrados na plataforma, visando aumentar a efetividade de identificação do animal perdido. Isso demonstra a intersecção entre as duas verticais: a necessidade de cadastro dos animais e unificação de uma base de dados para cada propósito. Para a vertical de busca para adoção, os abrigos devem cadastrar seus animais disponíveis no sistema do PetFinder, facilitando então a busca de usuários que querem adotar um novo companheiro, dada a centralização de dados de vários abrigos e simplificação do processo de busca.

Dado que este trabalho foi dividido em duas plataformas distintas, houve a necessidade de especificação de ambas as plataformas e um planejamento que permitisse desenvolvimento paralelo das 3 grandes tarefas: a busca por adoção, o match de animais perdidos, e a análise de imagem. Todo o planejamento de implementação bem como as sprints realizadas durante o desenvolvimento da plataforma estão documentados nos capítulos seguintes.

## 1.2 MOTIVAÇÃO

A decisão do tema se baseou em necessidades e fraquezas identificadas em relação aos métodos atuais presentes no Brasil para lidar com o abandono, a perda e a adoção de animais de estimação.

Analisando a vertical de animais perdidos por seus donos, é possível ver que não existe atualmente um conjunto de procedimentos protocolados que sejam reconhecidos e eficazes a serem seguidos pelos donos. Além de opções tradicionais pouco abrangentes (como colar cartazes nos bairros ou divulgar em redes sociais), as soluções atuais esbarram em problemas debilitantes, como uma base de usuários pouco consolidada ou necessidade de pagamento sem a garantia de encontrar o animal.

Em uma cidade como São Paulo, onde estima-se que um bairro tenha, em média, 120 mil habitantes, é vital que a informação de que um animal foi perdido consiga chegar ao maior número de pessoas possível da região de interesse, ou seja, o bairro onde o animal foi visto pela última vez. (SEADE, 2020)

Por outro lado, nas verticais de animais abandonados e animais para adoção, o Instituto Pet Brasil, entidade que publica estudos sobre o setor de produtos e serviços para animais de estimação, divulgou em 2018 dados que mostram uma grande discrepância entre o número de animais de estimação sob tutela de abrigos e ONGs (170 mil) e em situações vulneráveis (3.9 milhões), descritos como animais perdidos pelos donos, animais abandonados ou sob os cuidados de famílias abaixo da linha de pobreza (onde o risco de abandono se torna muito alto) e animais de rua (VELASCO, 2020).

Algumas das causas às quais essa disparidade pode ser atribuída são a quantidade e a capacidade de ONGs e abrigos voltados para o acolhimento de animais em situação vulnerável: das 370 organizações mapeadas pelo estudo, 33% não conseguem abrigar mais do que 100 animais, e apenas 19% têm capacidade de abrigo acima de 500 animais.

Partindo dessa premissa, foi idealizada a vertical deste projeto que propõe auxílio em encontrar um animal para adoção. A proposta é que, por facilitar a procura pelos animais e dar mais exposição dos animais vivendo nos abrigos, seja possível então aumentar o número de pessoas adotantes e, no futuro, diminuir a quantidade de animais de rua no país.

### 1.3 JUSTIFICATIVA

Com os dados apresentados acima, pode-se ver potenciais claros de utilização de tecnologia para ajudar em ambas as verticais mencionadas.

Olhando para a vertical de animais abandonados e disponíveis em abrigos de adoção, a tecnologia poderia ajudar na busca por um animal por parte de um possível adotante, sem ter todo o incômodo de visitar pessoalmente cada abrigo para ver os animais. Além disso, a ideia de ter um aplicativo que une as informações e animais de diversos abrigos permite uma exposição maior dos animais em cada um, pois pode virar um ponto de referência. Assim, foi julgado que o potencial mais claro estaria no auxílio do problema de capacidade, ao desenvolver uma aplicação que sirva como plataforma de divulgação e aumente a efetividade de adoção dos animais de estimação que estão sob cuidado das entidades, liberando espaço das mesmas para resgate de animais em situação vulnerável.

Já na vertical de animais perdidos, a tentativa de atenuação da disparidade mencionada previamente seria adaptar essa aplicação para que seja possível a existência de um espaço integrado onde usuários consigam cadastrar animais encontrados em situação vulnerável, que acreditam terem sido perdidos por seus

donos, e, do outro lado, os donos também possam cadastrar e procurar por seus animais perdidos.

Também é possível conferir a relevância deste assunto por haver algumas empresas que implementam parte do que está sendo proposto, como a Finding Rover<sup>1</sup>, empresa estrangeira não atuante no Brasil, que auxilia a busca por animais perdidos por Reconhecimento Facial dos animais. Eles dizem ter 98% de acurácia e utilizar um algoritmo que analisa 138 pontos no rosto do animal para identificá-lo. Este projeto faz uma comparação alternativa, por cores ao invés de reconhecimento facial, que não tem resultado tão preciso mas leva muito pouco tempo para realizar grandes quantidades de comparações.

#### 1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho foi organizado em oito capítulos. O primeiro capítulo, a qual este tópico pertence, corresponde à introdução do problema, discussão sobre motivação e um resumo sobre a ideia da solução.

O segundo capítulo aborda os aspectos conceituais que foram utilizados e discutidos durante o desenvolvimento deste projeto, descritos na seção com uma visão teórica e explicativa de como se conecta ao trabalho.

O terceiro capítulo apresenta as tecnologias usadas para desenvolvimento, teste e documentação de todo o projeto.

O quarto capítulo discursa sobre a metodologia aplicada durante todo o projeto, dando embasamento para as decisões posteriormente feitas e citadas em outras seções.

O quinto capítulo apresenta a especificação do sistema, com a descrição mais detalhada da ideia, a listagem de requisitos funcionais e não funcionais, e a exposição dos protótipos desenvolvidos para a plataforma projetada.

O sexto capítulo discute a implementação do sistema, citando as decisões tomadas, as abordagens seguidas e os resultados obtidos.

---

<sup>1</sup> Site do Finding Rover: <https://findingrover.com/>

O sétimo capítulo discursa sobre os testes feitos com os resultados obtidos e previamente apresentados no sexto capítulo, além das análises feitas a partir destes testes.

O oitavo e último capítulo consiste nas considerações finais do projeto, fazendo uma comparação entre as intenções iniciais e os resultados obtidos, a partir da qual são realizadas conclusões.

## 2 ASPECTOS CONCEITUAIS

Para conseguir modelar a solução para o problema identificado, foram utilizados como auxílio diversos conceitos, materiais e ferramentas sem os quais não seria possível ter tido o resultado apresentado. Todos esses auxiliares utilizados serão referenciados em seguida, bem como seus papéis no desenvolvimento do tema.

### 2.1 DIAGRAMA DE OSTERWALDER

Como primeiro artifício de modelagem, fizemos a descrição do Diagrama de Osterwalder representativo da ideia do PetFinder para melhor entendimento da proposta de valor e quais seriam as alavancas para alcançá-las.

O perfil de cliente é dividido entre a pessoa física que utiliza o aplicativo para buscar informações sobre adoção ou localização de animais, e as entidades dos abrigos e organizações que se responsabilizam por animais e procuram novos lares para eles.

As maiores dores identificadas se encontram na dificuldade de divulgação do processo de adoção de animais em maior escala, e conseqüente baixa procura e na falta de conhecimento para auxílio de animais encontrados na rua. Para que essas dores sejam convertidas em ganhos, espera-se que os clientes físicos sejam ativos nos seus cadastros, busquem opções de adoção acima de outras alternativas como a compra, e façam procedimentos de acordo com regulamentações para adoção consciente. Do lado das entidades, espera-se que tenham perfis atualizados e velocidade de resposta, trabalhem para organizar eventos de adoção e estejam de acordo com regulamentações governamentais para o tipo de trabalho que realizam.

A proposta de valor então se baseia na aproximação desses dois perfis que não possuem um canal de comunicação direto e amplo, e exposição das entidades aos interessados nos programas. Para isso, o sistema depende um banco de dados e de imagens extenso, resultando em comparações adequadas

dos animais submetidos por usuários, plataforma de contato entre os dois perfis de clientes, facilidade para cadastrar e buscar animais, tudo visando sempre a criação de ganhos e alívio de dores dos clientes.

Figura 1: Diagrama de Osterwalder



Fonte: elaborado pelos autores, utilizando o modelo de <https://analistamodelosdenegocios.com.br/canvas-da-proposta-de-valor/>

## 2.2 ADOÇÃO DE PETS

O processo de adoção varia de acordo com o abrigo em questão, mas todos têm suas restrições, como por exemplo as listadas pelo Centro de Controle de Zoonoses de São Paulo:

O adotante precisa ser maior de 21 anos, apresentar RG, CPF, comprovante de residência recente [e] assinar um termo se comprometendo a cuidar do animalzinho, que agora passa a ser de responsabilidade dele. No CCZ de São Paulo, é preciso pagar uma taxa de R\$ 16,20, pois o trâmite inclui a carteirinha do RGA (Registro Geral do Animal), uma plaqueta e um microchip de identificação. (<https://www.terra.com.br/vida-e-estilo/mulher/saiba-o-que-e-preciso-para-ad>



otar-um-animal-de-estimacao,07b96ee9f9e27310VgnCLD100000bbcceb0aR  
CRD.html)

Considerando então que o processo de adoção é regulamentado, além de ser uma decisão importantíssima tomada por ambas as partes do acordo, o intuito do projeto é facilitar o contato entre abrigos e possíveis tutores, promovendo a facilitação de pesquisa e de comunicação para que os mesmos possam focar nas responsabilidades legais juntos.

Essa facilidade ocorre na medida em que o aplicativo permite que o usuário veja os animais disponíveis para adoção e possa contatar o abrigo sem necessariamente fazer uma visita. Dessa maneira o usuário se sente mais compelido a procurar em diversos abrigos, e estes, por sua vez, tem um painel muito maior disponível, mostrando seus animais para muitas pessoas que não iriam fisicamente ao abrigo.

## 2.3 NORMAS E PADRONIZAÇÃO

Visando fornecer um conjunto de requisitos bem implementados, que o produto final atenda às necessidades e expectativas dos usuários, e que tudo esteja em conformidade com as leis e regulações aplicáveis, uma série de normas foram levadas em consideração durante a implementação do projeto.

### 2.3.1 Arquitetura de Sistemas de Software

Para definição de padrões para a arquitetura do sistema desenvolvido temos a ISO 10746 como auxílio para consulta.

Esta norma fornece uma estrutura bem desenvolvida para a estruturação de especificações para sistemas distribuídos em larga escala.

A estrutura para especificação de sistemas fornecida pela ISO 10746 possui quatro elementos fundamentais: uma abordagem de modelagem de objetos para especificação de sistemas; a especificação de um sistema em termos de especificações de ponto de vista separadas, mas

inter-relacionadas; a definição de uma infraestrutura de sistema que forneça transparências de distribuição para aplicativos do sistema; uma estrutura para avaliar a conformidade do sistema (ORGANIZAÇÃO INTERNACIONAL DE NORMALIZAÇÃO, 2009).

### 2.3.2 Qualidade de Sistemas de Software

Para definição de padrões para a qualidade do sistema desenvolvido, é possível utilizar duas padronizações ISO como auxílio para consulta, a ISO 12207 e a ISO 25010.

A ISO 12207 visa estabelecer uma estrutura comum para os processos de ciclo de vida e de desenvolvimento de softwares, procurando ajudar as organizações a compreenderem todos os componentes presentes na aquisição, desenvolvimento e fornecimento de software (ORGANIZAÇÃO INTERNACIONAL DE NORMALIZAÇÃO, 2008).

Na esfera profissional, ela é utilizada como base para empresas firmarem contratos e executarem projetos de forma mais eficaz. No contexto deste projeto de formatura, a norma garante com que exista uma padronização de linguagem e framework.

A ISO 25010, por sua vez, é direcionada para a qualidade de produtos de software e contém as características de qualidade que todos os softwares devem ter, de forma a alcançar um nível muito alto de qualidade no software que será entregue. Ela compreende 8 características de qualidade, são elas: Adequação funcional, Eficiência de desempenho, compatibilidade, usabilidade, confiabilidade, segurança, manutenção e portabilidade (ORGANIZAÇÃO INTERNACIONAL DE NORMALIZAÇÃO, 2011).

O projeto foi continuamente monitorado em relação a esses aspectos para garantir a qualidade do que foi entregue.

## 2.4 PLATAFORMAS CLOUD

Plataformas *cloud* disponibilizam uma suíte de serviços ao cliente, possibilitando que o mesmo desenvolva sites, aplicativos e softwares sem a necessidade de manter uma infraestrutura para isso, conseguindo controlar suas aplicações e demandas de infraestrutura de forma integrada, com controle total sobre o consumo desses serviços.

A plataforma *cloud* também traz maiores níveis de segurança e disponibilidade, dois requisitos relevantes para o projeto desenvolvido. Isso, aliado à facilidade de uso, tornou a plataforma *cloud* em uma escolha óbvia para a solução de hospedagem.

A decisão do projeto foi hospedar a aplicação em uma plataforma AWS (*Amazon Web Services*). Essa decisão foi tomada por se tratar de uma plataforma amplamente reconhecida e utilizada, se destacando entre startups, que tem um uso similar ao projeto aqui desenvolvido. Nos capítulos seguintes, será descrito com maior especificidade quais elementos do projeto utilizam a plataforma da AWS.

## 2.5 ALGORITMOS DE TRATAMENTO E COMPARAÇÃO DE IMAGENS

O projeto foi idealizado com a base de utilizar algoritmos de análise de imagens para prover os *outputs* necessários para a comparação do animais cadastrados pelos usuários na plataforma, de maneira que tornasse o processo de busca de animais perdidos mais eficiente como um todo para as partes envolvidas.

No projeto em questão, de início foi utilizado o acelerador Vize como ferramenta de análise e reconhecimento de imagens, para permitir o desenvolvimento de outras funcionalidades em paralelo, sem ter limitações por não ter o algoritmo de comparação de imagem completo. Em seguida, iniciou-se o desenvolvimento de um algoritmo próprio para a funcionalidade de comparação, fazendo uso de bibliotecas pré-existentes do Python e modelos públicos já treinados. O grupo se deparou com três diferentes métodos para realizar a análise de semelhanças de imagens: identificar pontos específicos do rostos dos

animais, comparação da imagem por formas e comparação por cor. Dentre essas opções optou-se pela comparação por cor, por ter um resultado mais rápido, o que era considerado um requisito muito relevante dado que a comparação vai ser feita entre grandes quantidades, e também por ter um resultado consideravelmente preciso (apesar de não ser o melhor dentre as três opções). Em vista disso, foram utilizados conceitos e técnicas de tratamento de imagens por meio de inteligência artificial, como *object detection*, *foreground extraction* e a conversão de imagens para histogramas.

A técnica de *object detection* consiste na prática de detectar instâncias de objetos de determinadas classes dentro de uma imagem. Neste projeto, a identificação desses objetos ocorre através de uma predição realizada pela máquina de qual a probabilidade do objeto identificado por uma caixa pertencer a determinada classe. O início desse processo é marcado pela divisão da imagens em células e, depois de mensuradas as probabilidades de existirem objetos dentro delas, aquelas que possuem baixas chances são descartadas enquanto as que permanecem são agrupadas por um processo denominado *non-max-suppression*, demarcando a área correspondente ao objeto inteiro. No fim do processo, têm-se as coordenadas da imagem que contêm os objetivos identificados - essas coordenadas foram utilizadas para recortar a imagem quando um animal for encontrado; assim, ao final é obtida a imagem recortada isolando apenas o cachorro da foto.

A etapa de *foreground extraction*, por sua vez, realiza o processo responsável por separar a imagem no que ele considera o conteúdo “principal” e o ambiente de fundo. Quando a distinção está completa, o algoritmo remove o conteúdo de fundo, substituindo seus pixels pela cor preta.

Esses dois processos são feitos durante o cadastro do animal, para ser possível guardar a imagem resultante no banco e utilizá-la no processo de comparação com os outros animais cadastrados. Essa comparação é feita por meio da conversão de imagens em histogramas e posterior comparação entre eles por meio do método que confere a intersecção entre os dois histogramas. Os valores obtidos são normalizados dentre seus limites e depois são ponderados

com o match feito por características escritas dos animais, resultando enfim no valor final de match entre os dois animais sendo comparados.

## 2.6 PRIVACIDADE DE DADOS

A proteção das informações dos usuários da aplicação é de suma importância para que o sistema seja utilizável, uma vez que a privacidade de dados é um tema de profunda atenção e necessidade dos usuários.

As medidas de segurança implementadas são determinadas com base na LGPD, a Lei Geral de Proteção de Dados, que versa sobre as novas normas virtuais no que tange a proteção de informações hospedadas em aplicações online e que entrará em vigor a partir de agosto de 2020.

A LGPD se aplica a qualquer atividade que envolva utilização de dados pessoais com o objetivo de proteger os direitos fundamentais de liberdade e de privacidade. A legislação prevê que é necessário um consentimento por escrito ou por outros meios que demonstrem a vontade do titular, sendo que em caso de qualquer alteração, o titular deve ser notificado e o mesmo poderá revogar o consentimento de acordo com sua vontade, em um processo gratuito e facilitado. (Lee Brock Camargo Advogados, 2020)

Foram feitas implementações de segurança para evitar acessos indevidos ao banco de dados, para proteger as informações dos usuários, e para outras finalidades, que serão explicadas em capítulos seguintes.

### 3 TECNOLOGIAS UTILIZADAS

O projeto pode ser dividido em 6 grandes partes que serviram como blocos de organização para o desenvolvimento:

1. Backend: Flask Python
2. Frontend: React Native
3. Base de dados: Flask SQLAlchemy
4. Algoritmo de análise e comparação de imagem
5. Storage: S3 - armazenamento das imagens
6. Server: EC2

Abaixo serão esclarecidas as ações transcorridas em cada uma dessas partes no projeto, ressaltando as extensões de tecnologias auxiliares utilizadas e o desenvolvimento como um todo nessas seis partes principais.

#### 3.1 PYTHON

Todos os integrantes do grupo já haviam tido contato prévio com a linguagem Python - portanto, dada a facilidade adquirida durante esse contato anterior, foi decidido que essa seria a linguagem utilizada no backend deste projeto. Por se tratar de uma linguagem gratuita com muitos usuários e uma biblioteca extensa disponível, utilizar o Python facilitou o desenvolvimento de todas as diferentes partes do sistema: backend, análise de imagem, comunicação com a AWS, segurança, entre outros.

O backend foi escrito em Python utilizando o Flask, que é um *framework web* que é encarregado da comunicação entre o frontend e o backend, através de comandos de POST e GET. (Flask.palletsprojects.com, 2020).

Também é necessário guardar as imagens cadastradas para posteriores matches e exibições. Para tal, foi integrado na aplicação o serviço S3 que faz

parte da Cloud da Amazon (AWS) através da biblioteca do *boto3*. Esta tecnologia é detalhada no item 3.4.

O backend também é responsável pela segurança das informações, tanto no armazenamento dos dados quanto para garantir uma comunicação segura entre o backend e o frontend da aplicação.

### 3.2 REACT NATIVE

Para o desenvolvimento do frontend do aplicativo, foi escolhido o React Native, por recomendações de colegas de trabalho. Como nenhum dos integrantes do grupo possuía nenhuma experiência prévia com desenvolvimento mobile, foi seguida a orientação dada por colegas já com conhecimento nesta área.

O React Native é um *framework* baseado no React, ambos criados e mantidos pelo Facebook, e utiliza a linguagem Javascript mas renderiza as páginas com componentes nativos, o que é um grande atrativo deste *framework* e permitiu sua disseminação no mundo técnico.

### 3.3 FLASK SQLALCHEMY

Para guardar as informações de todos os cadastros, foi utilizado o Flask SQLAlchemy, que é uma extensão do Flask, utilizado no backend, para uma fácil utilização do SQLAlchemy, que permite a criação de um banco de dados relacional.

### 3.4 RECONHECIMENTO E COMPARAÇÃO DE IMAGENS

Neste item, são descritos os dois modos utilizados durante o desenvolvimento do projeto para comparar as imagens cadastradas e fazer o match de animal perdido. O Vize, apresentado primeiro, foi utilizado apenas como acelerador no primeiro semestre de 2020, para permitir o desenvolvimento em

paralelo de outras funcionalidades, mas foi posteriormente abandonado quando foi finalizada a implementação do algoritmo de análise e comparação de imagem específico para este projeto.

### 3.4.1 **Vize.ai**

Ao utilizar um acelerador de comparação de imagem, foi possível desenvolver as outras funcionalidades do aplicativo sem depender de ter o algoritmo de análise e comparação de imagem próprio completo. Assim, o desenvolvimento geral da plataforma (tanto pelo lado das *features*, como o cadastro de animais e a procura por eles, quanto pelo lado de UI, por exemplo) pôde ser feito paralelamente com a solução de análise de imagem.

Assim, durante todo o primeiro semestre, foi utilizado o acelerador Vize.ai enquanto eram desenvolvidas as funcionalidades de cadastro de animais perdidos e encontrados, cadastro e login de usuários, além do match completo (por imagem e características) dos animais cadastrados. A partir dessa etapa, com um maior conhecimento de desenvolvimento de aplicativos, foi possível dividir entre os quatro integrantes a continuação do desenvolvimento de funcionalidades (agora focadas em adoção e abrigos) e a criação do algoritmo próprio de análise das imagens dos animais. Então, com o algoritmo próprio finalizado, o acelerador foi retirado do projeto e seguiu-se o desenvolvimento das outras funcionalidades.

Em relação ao Vize.ai, este foi um acelerador recomendado para este projeto por um mestrando, assistente do professor orientador. O plano gratuito desta plataforma tem as funções de Reconhecimento de Imagem, *Generic Tagging*, Similaridade de Imagens e Similaridade de Produtos - para este projeto, os serviços utilizados foram Reconhecimento e Similaridade de Imagens, pois são os mais adequados para as funcionalidades previstas.

A estrutura do Vize é feita por coleções, que seria como um banco de imagens de fotos. O plano gratuito permite apenas uma coleção - o que não é ideal, pois a intenção inicial era separar a base de dados de animais perdidos e



encontrados. Porém, como a utilização deste acelerador é apenas para permitir a paralelização do desenvolvimento, não houve necessidade de procurar alternativas apenas por causa dessa questão - o que é buscado em um acelerador, neste caso, não é sua precisão de resultados e idealidade de armazenamento, mas sim a facilidade de utilização, para não causar atrito com as partes permanentes do desenvolvimento.

Toda a adequação do Vize para o projeto é feita pela API, desde a adição de imagens até a procura e comparação de cada uma. Isso é um dos motivos para ter sido escolhido como o acelerador para este projeto - há diversos métodos, acessados facilmente por *requests* do tipo GET e POST.

Outro motivo é o resultado retornado pelos variados métodos - são dados suficientes dentro da necessidade para o desenvolvimento completo da interface. O dado recebido na função de Reconhecimento de Imagem era o principal para validar a adequação do acelerador, e é compatível com as necessidades deste projeto, pois não é binário (como “match” ou “não match”). Neste método de busca, é possível configurar quantos resultados devem ser retornados, e eles são apresentados com o que é chamado de “distância entre imagens”, ou seja, a similaridade (ou, no caso, a falta de similaridade) entre elas. Assim, é possível criar uma tela na plataforma com diversos resultados e ordená-los por proximidade à foto buscada, como era a intenção inicial do projeto.

Abaixo é mostrado um fluxo completo, desde o cadastro até a busca, usando esta função de Reconhecimento de Imagem, para exemplificar a comprovar a conformidade deste acelerador com este projeto.

#### 3.4.1.1 Reconhecimento de Imagem

O serviço de Reconhecimento de Imagem é utilizado para dar match entre uma busca (no cadastro de animal perdido ou encontrado) e o banco de dados (ou coleção) em que estão armazenados os possíveis correspondentes.

Primeiro, deve-se popular a coleção do Vize. Para isso, é feito um *request* para a API de *insert*, com os *headers* contendo o *token* de autorização e o ID da

coleção em que se está inserindo. O *body* do request deve conter os IDs e a URL de cada imagem a ser inserida.

Code Snippet 1: *Body* do request para popular a coleção do Vize.

```

{"fields_to_return" : [ "_id" ], "records" : [
  { "_id" : "1",
    "_url" :
    "https://img.elo7.com.br/product/original/2926731/kit-peitoral-e-guia-para-golden-retr
iever-brinde-cinto-rottweiler.jpg"},
    { "_id" : "2",
    "_url" :
    "https://www.racasdecachorro.com.br/wp-content/uploads/2018/09/sh_pinscher-miniatura_5
70795091.jpg"},
    { "_id" : "3",
    "_url" : "https://doggiedesigner.com/wp-content/uploads/2018/09/Doxie-Pin-3.jpg"},
    { "_id" : "4",
    "_url" : "https://img.olx.com.br/images/19/192027013550052.jpg"},
    { "_id" : "5",
    "_url" : "https://labradorclub.org.nz/wp-content/uploads/2018/11/Jack-274x300.jpg"},
    { "_id" : "6",
    "_url" :
    "https://i.pining.com/originals/b0/38/88/b038888b8c9deecf2fdac223fab4d33.jpg"},
    { "_id" : "7",
    "_url" :
    "https://sites.google.com/site/lostmalesiberianhusky/_/rsrc/1304100260798/home/King%2C
%20my%20lost%20Siberian%20Husky%20pdfn%20jpeg_Page_07.jpg?height=1316&width=1618"},
    { "_id" : "8",
    "_url" :
    "https://www.pets4homes.co.uk/images/classifieds/2014/12/29/850421/large/husky-looking
-for-a-new-home-54a1da0a007b2.jpg"}]]}

```

Fonte: elaborado pelos autores.

Code Snippet 2: Retorno do request a fim de popular a coleção do Vize.

```

{"status": { "code": 210, "text": "records inserted" },
"statistics": { "OperationTime": 4269 },
"answer_records": [{"_id": "1"}, {"_id": "2"}, {"_id": "3"}, {"_id": "4"}, {"_id":
"5"}, {"_id": "6"}, {"_id": "7"}, {"_id": "8"}],
"skipped_records": [],
"answer_count": 8}

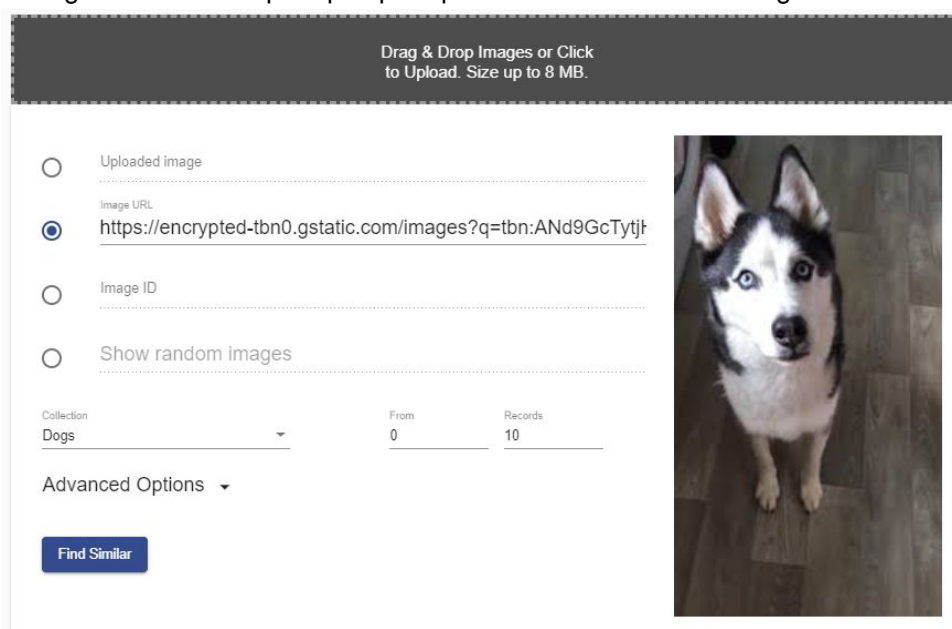
```

Fonte: <https://vize.ai/>

Com a coleção popularizada, pode-se fazer a primeira busca. Para fins de melhor exemplificar o funcionamento, foram inseridas algumas fotos de raças variadas e algumas fotos da raça Husky Siberiano, que será utilizada também na busca.

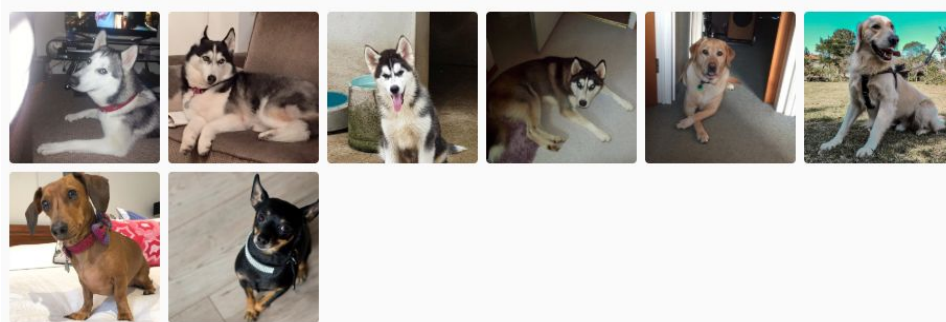
A busca, apesar de também poder ser feita por um *request*, neste exemplo será feita diretamente no site do Vize, a fim de exibir os resultados diretamente por imagem ao invés de por URL, para facilitar a observação de corretude do resultado.

Figura 2: Interface para pesquisa por Reconhecimento de Imagem no Vize.



Fonte: <https://vize.ai/>

Figura 3: Retorno visual da pesquisa, ordenada por similaridade



Fonte: <https://vize.ai/>

Code Snippet 3: Retorno em código da pesquisa.

```

    {"status": {"code": 200, "text": "OK"},
    "statistics": {"OperationTime": 329},
    "answer_records": [
      {"_id": "8",
      "_url":
      "https://www.pets4homes.co.uk/images/classifieds/2014/12/29/850421/large/husky-looking
      -for-a-new-home-54a1da0a007b2.jpg"},
      {"_id": "6",
      "_url":
      "https://i.pining.com/originals/b0/38/88/b038888b8c9deeacf2fdae223fab4d33.jpg"},
      {"_id": "4",
      "_url": "https://img.olx.com.br/images/19/192027013550052.jpg"},
      {"_id": "7",
      "_url":
      "https://sites.google.com/site/lostmalesiberianhusky/_/rsrc/1304100260798/home/King%2C
      %20my%20lost%20Siberian%20Husky%20pdfn%20jpeg_Page_07.jpg?height=1316&width=1618"},
      {"_id": "5",
      "_url": "https://labradorclub.org.nz/wp-content/uploads/2018/11/Jack-274x300.jpg"},
      {"_id": "1",
      "_url":
      "https://img.elo7.com.br/product/original/2926731/kit-peitoral-e-guia-para-golden-retr
      iever-brinde-cinto-rottweiler.jpg"},
      {"_id": "3",
      "_url": "https://doggiedesigner.com/wp-content/uploads/2018/09/Doxie-Pin-3.jpg"},
      {"_id": "2",
      "_url":
      "https://www.racasdecachorro.com.br/wp-content/uploads/2018/09/sh_pinscher-miniatura_5
      70795091.jpg"}],
    "answer_distances":
    [0.3677506,0.38800547,0.399719,0.40319747,0.5878337,0.60501754,0.7154529,0.75068265],
    "answer_count": 8}

```

Fonte: <https://vize.ai/>

Na figura 2, é possível ver qual imagem está sendo utilizada na função de reconhecimento. Na figura 3, é possível ver a ordenação dos resultados, observando-se que as fotos de cães da raça Husky Siberiano tiveram distâncias menores, ou seja, foram calculadas maiores similaridades.

No terceiro *snippet* de código, é possível ver no item “*answer\_distances*” as distâncias das imagens apresentadas. O item “*answer\_records*” também é dado por ranking e é retornado com as URLs, o que permite o desenvolvimento de uma tela, na plataforma deste projeto, que apresente os resultados parecidos também ranqueados e com suas imagens diretamente apresentadas.

### 3.4.2 Algoritmo próprio

Foi desenvolvido um algoritmo específico para esta plataforma, ou seja, projetado para identificar o cachorro numa imagem e permitir a comparação com outras imagens de cachorros. O algoritmo completo foi dividido em três etapas (explicadas nos itens abaixo) e, para cada etapa, foram utilizadas bibliotecas do Python que auxiliaram na identificação e tratamento das imagens. As bibliotecas, modelos e algoritmos utilizados são explicados nos tópicos abaixo.

#### 3.4.2.1 YOLO *Object Detection*

Na primeira fase, a imagem fornecida pelo usuário é passada pelo algoritmo de *Object Detection*, que procura o cachorro na foto. Se é encontrado, a imagem é recortada para deixar apenas o animal, retirando o excesso de ambiente. Caso não encontre nenhum animal ou mais de um, é emitido um erro.

Figura 4: Imagem original utilizada para exemplificar o processo de Object Detection



Fonte:

[https://img.theculturetrip.com/768x432/wp-content/uploads/2017/09/24276978571\\_51a7f5128c\\_k.jpg](https://img.theculturetrip.com/768x432/wp-content/uploads/2017/09/24276978571_51a7f5128c_k.jpg)

Figura 5: Imagem após o procedimento de *Object Detection*.



Fonte: Alteração feita pelos autores. Original em [https://img.theculturetrip.com/768x432/wp-content/uploads/2017/09/24276978571\\_51a7f5128c\\_k.jpg](https://img.theculturetrip.com/768x432/wp-content/uploads/2017/09/24276978571_51a7f5128c_k.jpg)

Para isso, foi utilizado o algoritmo YOLO, treinado com a base de dados COCO (que consiste de 80 *labels*, apesar de nós só utilizarmos os 10 de animais e darmos maior importância para 1 - o de cachorro).

Assim, com a combinação do YOLO e do *dataset* COCO, foi possível utilizar um modelo já treinado (pelo time Darknet) e importado para o projeto pela biblioteca OpenCV.

A escolha do algoritmo YOLO foi dada por dois fatores. Primeiro, pelas prioridades alinhadas com a deste projeto, pois este algoritmo utiliza a Estratégia de Detector de Um Estágio (tradução livre de *One-Stage Detector Strategy*), que prioriza a performance em relação à acurácia (o contrário da Estratégia de Detector de Dois Estágios). Como este projeto foi imaginado para utilização do público, foi dada a preferência para uma análise mais rápida das imagens a serem cadastradas, para não atingir negativamente a experiência de usuário.

Em segundo, este modelo e o *dataset* são facilmente obtidos e utilizados, o que conta como grande vantagem para concluir sua utilização neste projeto. A

importação do modelo e dos dados é feita pela biblioteca OpenCV do Python, sendo que apenas dois arquivos de dados são necessários: os *labels* e os pesos.

#### 3.4.2.2 OpenCV GrabCut: *Foreground Segmentation e Extraction*

Na próxima etapa, agora que já se possui a imagem cortada do cachorro, deve ser feita a segmentação do que é animal e o que é ambiente na imagem. Com isso determinado, faz-se a remoção do ambiente (deixando apenas a cor preta), isolando apenas o animal na imagem.

Para isso, foi utilizado o algoritmo GrabCut, que faz iterativamente o seguinte processo:

- “Passo 1: Estima a distribuição de cor do fundo e do primeiro plano via Modelo de Misturas Gaussianas (GMM)
- Passo 2: Constrói um campo aleatório de Markov em cima dos labels de cada pixel (ou seja, se é primeiro plano ou de fundo)
- Passo 3: Aplica uma otimização de corte gráfico para chegar à segmentação final.” (Tradução livre. Fonte: <https://www.pyimagesearch.com/2020/07/27/opencv-grabcut-foreground-segmentation-and-extraction/>)

Assim, quanto mais iterações são feitas, mais preciso é o resultado - em contrapartida, mais demorado é o processo. Na implementação utilizada para este projeto, foi determinado que o número de iterações seria cinco, o que permite uma acurácia aceitável sem exigir muito tempo do usuário (como é mostrado no tópico 7.1.1.2).

Assim como o YOLO, este algoritmo foi escolhido para este projeto pela possibilidade de controlar sua duração (pelo número de iterações) e pela facilidade de uso. Também foi utilizada a biblioteca OpenCV para importar o modelo do algoritmo GrabCut e utilizá-lo nas imagens fornecidas.

Figura 6: Resultado de 5 iterações de GrabCut feitas na Figura 4.



Fonte: Alteração feita pelos autores. Original em [https://img.theculturetrip.com/768x432/wp-content/uploads/2017/09/24276978571\\_51a7f5128c\\_k.jpg](https://img.theculturetrip.com/768x432/wp-content/uploads/2017/09/24276978571_51a7f5128c_k.jpg)

#### 3.4.2.3 Comparação por cores

Por fim, tendo padronizado que todas as imagens a serem comparadas serão as de resultado do GrabCut (ou seja, com fundo preto e cachorro isolado), é possível fazer uma comparação de cores pela intersecção dos histogramas de cada imagem.

Para isso, foi utilizada novamente a biblioteca OpenCV para obter as imagens e então convertê-las em histogramas, que são representações gráficas dos pixels de uma imagem em função de sua intensidade. Em seguida, esses histogramas são comparados por um método também da biblioteca OpenCV, que obtém a intersecção dos histogramas apresentados. Assim, quanto maior o resultado, mais similares são as imagens.

A escolha deste método para comparação de imagens foi feita por combinar bem a rapidez dos resultados e a importância das cores. Havia outros métodos que faziam uma comparação mais direcionada a raças do que a cores, inclusive o Reconhecimento Facial, mas descartamos essas alternativas por terem uma performance muito inferior, chegando a demorar minutos para



comparar apenas duas imagens. Com o método escolhido, apesar de não ter a comparação de raças, é obtida uma acurácia aceitável dado que uma comparação de cores é bem significativa, ainda mais com uma base de dados de cachorros (que raças iguais podem até ter cores diferentes).

Ainda assim, a precisão deste método também é afetada pela posição dos animais nas fotos, dado que é feita a comparação da intersecção. Por isso, é possível que duas imagens em que os cachorros estão na mesma posição, mas têm cor bem diferente, tenham uma intersecção maior do que imagens de cachorros com cores similares em posições diferentes.

Apesar de não ser o algoritmo com maior precisão encontrada, ele se adequou bem às prioridades deste projeto, dando maior importância ao tempo gasto do que à precisão absoluta.

### 3.5 S3

O *Amazon Simple Storage Service*, conhecido como Amazon S3, é um serviço de armazenamento de dados na nuvem. O S3 faz uso do armazenamento de dados como objetos em contêineres denominados *buckets*. É possível controlar os acessos a esses repositórios, decidindo quem pode criar, excluir e listar objetos nele. Também é possível escolher a região geográfica em que o Amazon S3 armazenará o *bucket* e seu conteúdo e visualizar os logs de acesso do *bucket* e seus objetos.

Foi decidido pelo uso desta plataforma por ter fácil acesso durante o desenvolvimento por todos os membros do grupo. O S3 foi utilizado para armazenamento de imagens do projeto.

### 3.6 GOOGLE MAPS SDK FOR ANDROID

Foi utilizado o SDK de mapa do Google para Android, para poder exibir um mapa e obter as coordenadas selecionadas, tanto no cadastro de usuário e

abrigo quanto nos de animais. As coordenadas salvas são utilizadas no match dos animais perdidos e na busca de adoção.

### 3.7 API DE *SEARCH E FORWARD GEOCODING* DA LOCATIONIQ

Com as coordenadas obtidas pelo Google Maps SDK, é possível fazer um request para a API de Geocoding da LocationIQ para converter as coordenadas em um endereço legível, como rua e cidade. Também é possível fazer o processo contrário: fornecer o endereço legível e procurar pelas coordenadas.

Foi utilizada a versão gratuita desta API, que permite até 5 mil requests por dia e até 2 requests por segundo.

### 3.8 TECNOLOGIAS UTILIZADAS PARA TESTES E DESENVOLVIMENTO

#### 3.8.1 **Android Studio**

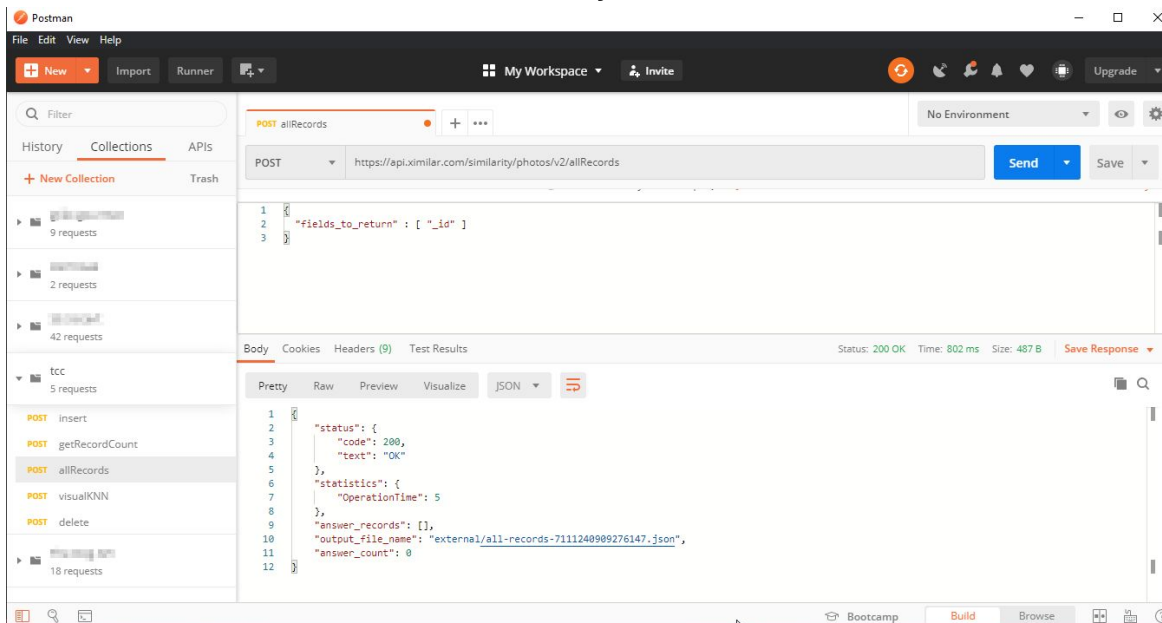
Android Studio é uma plataforma gratuita muito utilizada para desenvolvimento de aplicativos. Nele, é possível criar um emulador de Android e assim testar o aplicativo que estava sendo desenvolvido no computador. (Android Developers, 2020)

Foi utilizado neste projeto para auxiliar no desenvolvimento, ao permitir que fosse criado um emulador de celular Android em que era possível rodar o aplicativo sendo desenvolvido.

#### 3.8.2 **Postman**

Postman é utilizado no desenvolvimento de APIs, permitindo o teste de *requests* de GET e POST. (Postman.com, 2020) Neste projeto, foi utilizado durante a integração com o Vize, para testar as diversas funções que a documentação do Vize listava.

Figura 7: Tela do Postman, mostrando o *request* feito para a API *allRecords* do Vize. É possível visualizar o *body* e o *response*. À esquerda, é possível ver a lista de *requests* já testados, cujos nomes são funções do Vize.



Fonte: elaborado pelos autores, utilizando Postman (<https://www.postman.com/>)

### 3.8.3 MarvelApp

MarvelApp é uma plataforma online que permite a criação de *wireframes*, *mockups* e design tanto de sites quanto de aplicativos. Por meio dela, é possível planejar o design de telas de um aplicativo e, a partir dele, criar um protótipo interativo capaz de gerar uma navegação simplificada entre essas telas estáticas.

Essa plataforma foi utilizada para a criação da versão 2 do protótipo, apresentada no item 5.4.

### 3.9 EC2

EC2 é uma plataforma de *cloud computing* da Amazon Services (AWS). Ela permite que os usuários aluguem computadores virtuais para processamento, permitindo a criação de aplicações escaláveis. É permitido escolher em qual *datacenter* da AWS deseja-se criar a instância, assim como todas as regras que envolvem a máquina. Assim, é o usuário que tem 100% controle do que deseja

fazer. Por estar dentro do ambiente da AWS, é facilitado o uso de outras aplicações da plataforma, como o S3, para armazenamento de dados.

O EC2 foi utilizado neste projeto para permitir a utilização do aplicativo sem ter necessidade de rodar o backend e o frontend em modo local, ou seja, no seu próprio computador. Com o EC2, é possível se conectar virtualmente.

## 4 METODOLOGIA DE TRABALHO

Para que o desenvolvimento de um projeto ocorra da forma esperada, é necessário um planejamento prévio que forneça diretrizes aos desenvolvedores para garantir a qualidade do mesmo.

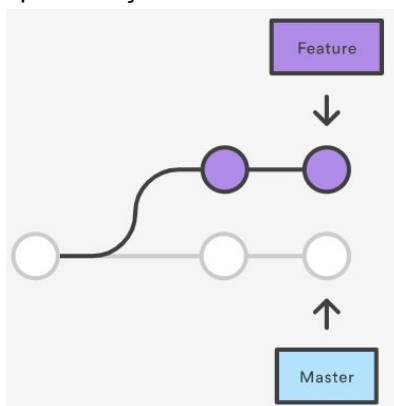
Para tanto, este capítulo abordará as metodologias e ferramentas utilizadas para alavancar esse planejamento, assim como o desenvolvimento do mesmo.

É importante notar que apesar do planejamento servir como base, o projeto foi atualizado ao longo do caminho de acordo com as necessidades identificadas no projeto e tomando as ações que o serviriam da melhor forma.

### 4.1 REPOSITÓRIO GIT

Dado que o grupo tem quatro alunos, era necessário se pensar em um modelo de organização do projeto em que os quatro pudessem trabalhar ao mesmo tempo, sem interferir e prejudicar o progresso do colega, principalmente considerando o trabalho era majoritariamente remoto, devido à quarentena. Para isso, foi decidido que seria utilizado um repositório git para manter o progresso e facilitar o desenvolvimento deste projeto.

A principal vantagem ao utilizar um repositório git é a capacidade das branches. Foi criada a branch *master*, que contém uma versão funcional do projeto e, a cada início de uma nova tarefa, por qualquer um dos integrantes, é criada uma nova branch de *feature*, que se separa da *master* para poder desenvolver a nova funcionalidade sem afetar a versão já funcional.

Figura 8: Representação das branches *master* e *feature*

Fonte: <https://www.atlassian.com/git/tutorials/why-git>

Assim, cada um dos integrantes podia criar sua própria branch, a partir da *master*, e trabalhar na sua tarefa separadamente dos colegas até que a *feature* estivesse completa e, então, realizar o *merge* com a *master*, finalizando o processo de desenvolvimento daquela *feature*. A partir disso, a *master* agora tem uma nova versão funcional da plataforma.

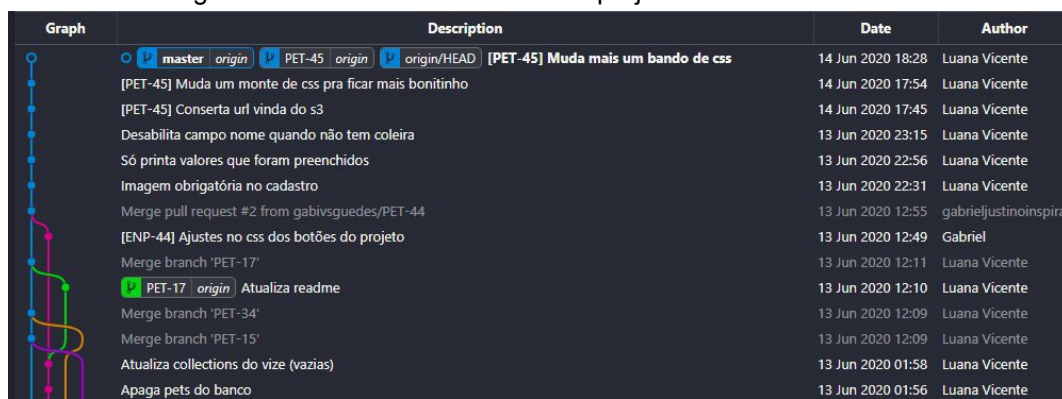
Para ilustrar nossa utilização desse repositório, as figuras abaixo mostram, em dado ponto do nosso desenvolvimento, as diversas branches ativas no momento, com os integrantes que estavam trabalhando nelas e suas ligações com a *master*.

Figura 9: Branches ativas no repositório no dia 14/06/2020.

Branch Name	Last Updated	Updated By	Commits	Actions
PET-45	Updated 4 hours ago	luanavicante	0   0	New pull request
PET-17	Updated yesterday	luanavicante	11   0	New pull request
PET-34	Updated 2 days ago	gabrieljustinoinspira	19   0	#1 Merged
PET-15	Updated 2 days ago	gabrieljustinoinspira	24   0	New pull request
Match-caracteristicas-2	Updated 7 days ago	thiagoyork	35   2	New pull request

Fonte: <https://github.com/gabivsguedes/petfinder/branches/active> (repositório privado).

Figura 10: Gráfico das branches do projeto no dia 14/06/2020.



Fonte: elaborado pelos autores, utilizando a extensão do VSCode Git Graph (<https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph>)

Além de facilitar o desenvolvimento por causa das branches, criar um repositório git também proporciona outra vantagem: o histórico. Como é possível ver na figura 10, o git mantém um histórico de cada commit, com uma mensagem descrevendo o que foi alterado, a data e o autor. Isso facilita muito na correção de bugs e ajuda a prevenir erros de se tornarem permanentes.

## 4.2 METODOLOGIA ÁGIL

As Metodologias Ágeis são amplamente utilizadas atualmente, pois tiram o enfoque de um projeto da documentação e trazem para o desenvolvimento iterativo, fornecendo resultados graduais, ao contrário de esperar-se meses para ter uma primeira versão do produto final. Dado o contato prévio com essa metodologia por parte dos integrantes e a convicção de que seria o melhor modo de organizar o progresso deste projeto, foi decidido que seria utilizada a metodologia Scrum.

O Scrum é uma das metodologias ágeis mais presentes no mercado e tem um modelo iterativo adequado para o objetivo deste projeto. No Scrum, o desenvolvimento dos projetos é dividido em sprints, que são períodos de tempo, normalmente variando de uma semana a um mês, em que se implementa uma série de funcionalidades discutidas previamente. A intenção, como dito nos

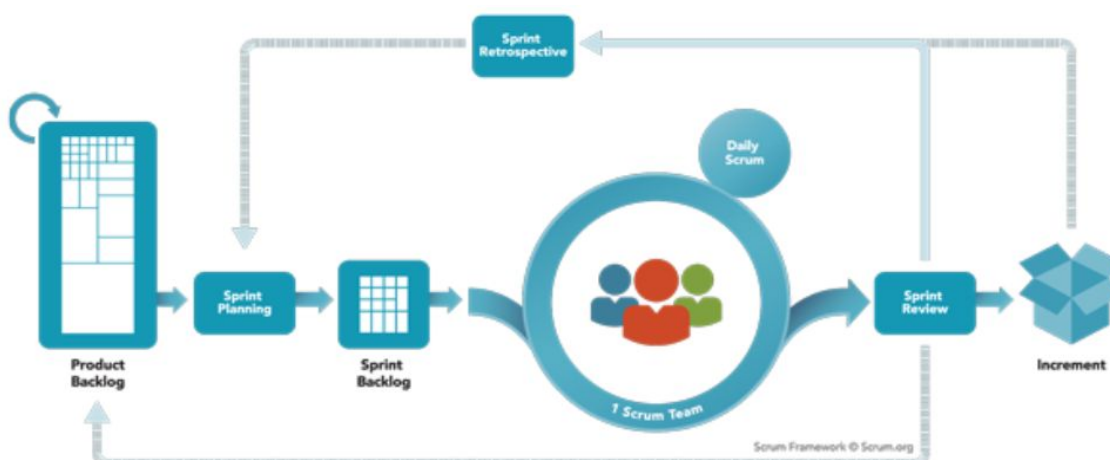
princípios das metodologias ágeis, é de entregar um produto funcional, mesmo que não inteiramente completo, a cada fim de sprint.

O ciclo principal do Scrum segue um modelo de organização e tem os seguintes elementos, já com a descrição voltada ao projeto aqui apresentado:

- **Product Backlog:** uma lista de todas as funcionalidades desejadas para a plataforma finalizada.
- **Sprint Planning:** uma reunião em que se junta os integrantes e, se necessário, o orientador, para definir quais são as próximas funcionalidades mais importantes e adequadas a serem implementadas.
- **Sprint Backlog:** lista de funcionalidades que foi concordado que serão implementadas na sprint atual.
- **Desenvolvimento:** período determinado em que é feita a implementação das funcionalidades selecionadas.
- **Daily Scrum:** reunião diária para conferir o andamento da implementação. Neste projeto, apenas em períodos críticos houve comunicação diária, e se limitava a atualizações por mensagem, não chegando a ser uma reunião oficial.
- **Sprint Review:** ao fim do período de *sprint*, é feita uma reunião para conferir se todas as funcionalidades foram implementadas com sucesso e apresentá-las aos outros integrantes por meio de uma demonstração. Ocasionalmente, o orientador também participa dessa reunião.
- **Increment:** é o fruto do trabalho durante esta *sprint*, as novas funcionalidades acrescentadas ao produto.
- **Sprint Retrospective:** uma reunião para analisar dificuldades enfrentadas durante a *sprint* passada e possíveis melhorias para as próximas *sprints*. Essa reunião acabou sendo unida à de Sprint Review para este projeto.



Figura 11: Ciclo do Scrum.



Fonte: <https://www.scrum.org/resources/what-is-scrum>

O projeto foi dividido em três versões principais, desenvolvidas com o apoio do Scrum.

- A primeira versão do projeto, também utilizada para a disciplina de Laboratório de Software, continha as funcionalidades da vertical de pet perdido, com a utilização do Vize para as funcionalidades de comparação de imagens.
- A segunda versão trouxe a adição da vertical de adoção de pets e de características de segurança reforçadas, além da introdução do algoritmo proprietário desenvolvido para a análise de imagens, substituindo o Vize.
- A terceira versão focou em ajustes nas funcionalidades de ambas verticais e no redesenho da aparência geral do app, com um visual mais familiar para usuários, buscando tornar o app mais intuitivo e atrativo.

### 4.3 JIRA

Seguindo a ideia da metodologia ágil e Scrum, foi decidido utilizar o Jira para organizar o backlog e as sprints. O Jira é uma ferramenta da Atlassian feita

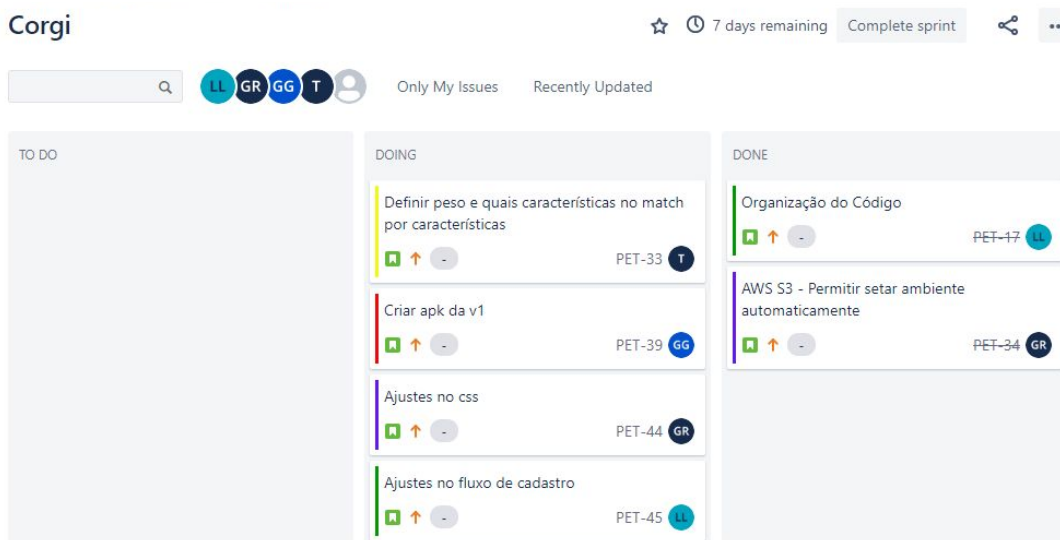
exatamente para este tipo de projeto: de desenvolvimento de software, com entregas parciais e um time de pessoas trabalhando no mesmo projeto.

Nessa ferramenta, são criadas *issues* - que são basicamente descrições de tarefas, que podem abranger desde novas features, correção de bugs, deveres de boas práticas etc. Cada issue é atribuída a um dos integrantes e pode ter quatro status:

- **Backlog:** não está prevista para ser implementada nesta *sprint*.
- **To Do:** ainda falta ser começada, mas está prevista para esta *sprint*.
- **Doing:** já foi começada e ainda não foi finalizada.
- **Done:** já foi finalizada e foi feito o *merge* na *master*.

No Jira também é possível criar sprints, com a duração podendo ser configurada livremente, além de poder gerar relatórios nas sprints passadas. Neste projeto, cada sprint criada foi nomeada a partir de uma raça de cachorro, para combinar com o tema.

Figura 12: Sprint *Corgi*, no dia 14/06/2020.



Fonte: elaborado pelos autores, utilizando o Jira (<https://www.atlassian.com/software/jira>)

## 4.4 PESQUISA DE USUÁRIOS

Para obter dados mais precisos sobre os principais perfis de usuários que farão uso do sistema produzido foi realizada uma pesquisa com foco na frente do projeto que lida com a procura de animais perdidos. Essa pesquisa foi realizada para a matéria de Interação Humano-Computador.

### 4.4.1 Stakeholders

Os stakeholders do projeto que foram identificados são as pessoas que perderam ou encontraram animais, pessoas que desejam adotar algum animal e ONGs de animais. As pessoas que perderam seus animais poderão entrar no aplicativo para sinalizar o ocorrido através do upload de fotos do animal, que serão comparadas no sistema a fim de encontrar o animal perdido com tais características no banco de dados. Tendo isso em vista, esses usuários são os maiores interessados na iniciativa do projeto, pois desejam encontrar seus animais de estimação.

Já as pessoas que encontraram animais possuem interesse, mas este partirá de uma ação voluntária de entrar no sistema para enviar fotos, descrições, localização, de modo que o animal seja identificado. Estas pessoas também são classificadas como usuárias.

Por fim, pessoas que desejam adotar um animal e as ONGs de animais podem, respectivamente, procurar e cadastrar animais para adoção, além da possibilidade de ambos demonstrarem interesse em ajudar a iniciativa implementada paralelamente, cadastrando animais perdidos. Dado o papel descrito, as ONGs ainda podem funcionar como usuárias (que publicam dados de animais perdidos) e não-usuárias, tendo o papel de abrigo temporário do animal, externo ao sistema.

#### 4.4.2 Papéis e variáveis de perfil

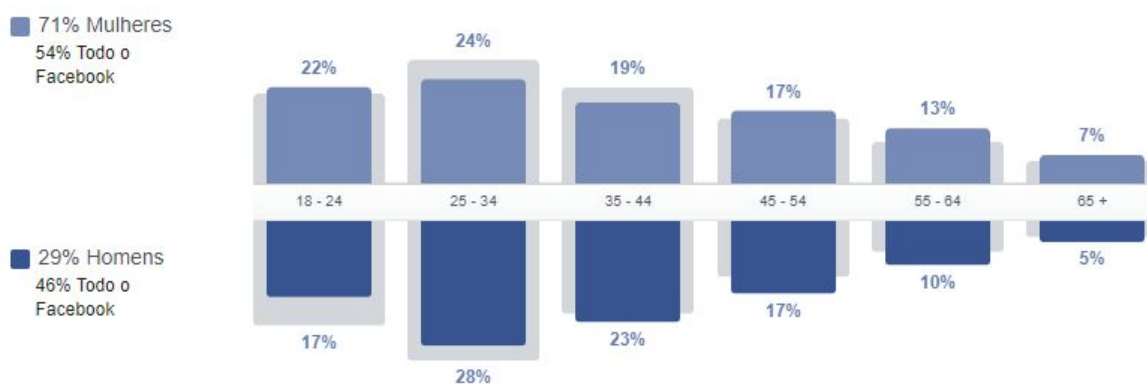
É importante que o design e a linguagem do sistema corresponda aos perfis de usuários que o utilizará, adaptando-se ao máximo ao conhecimento acumulado da pessoa. Portanto, é necessário descobrir as características mais recorrentes do público alvo, como idade, gênero, escolaridade, tipo de moradia (apartamento/casa) e nível socioeconômico.

Tendo um panorama geral sobre idade, escolaridade e nível sócio econômico, é possível tomar decisões acerca da linguagem usada na comunicação do aplicativo, se é melhor um tom mais divertido ou mais sério. Também é possível prever o quanto os usuários já estão adaptados às tecnologias envolvidas e quais hábitos rotineiros eles têm em comum.

Unindo-se as variáveis e os resultados da pesquisa, é possível até reconhecer padrões, como por exemplo o quão frequente é a perda de animais domésticos para pessoas que moram em casa versus àquelas que moram em apartamento. Ou até em qual nível socioeconômico é mais comum encontrar animais perdidos.

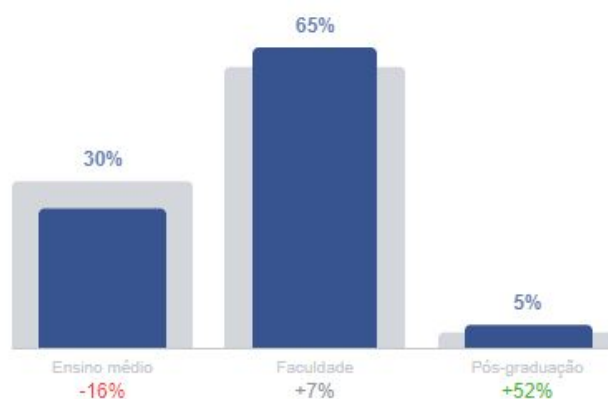
A fim de entender melhor o perfil da população e validar essas informações com as que serão coletadas através da pesquisa, utilizou-se a ferramenta *Audience Insights* do *Facebook Ads*, que auxilia com variáveis sociodemográficas e outras, sobre um determinado segmento da população. Com isso, buscou-se por grupos de resgate de animais, onde se encontram possíveis usuários do sistema, e obteve-se alguns dados sobre esse grupo:

Figura 13: Gênero e idade dos perfis.



Fonte: *Facebook Audience Insights* ([https://www.facebook.com/ads/audience\\_insights?ref=fbiq\\_ai](https://www.facebook.com/ads/audience_insights?ref=fbiq_ai))

Figura 14: Nível educacional dos perfis.



Fonte: *Facebook Audience Insights* ([https://www.facebook.com/ads/audience\\_insights?ref=fbiq\\_ai](https://www.facebook.com/ads/audience_insights?ref=fbiq_ai))

#### 4.4.3 Necessidades

Os objetivos da pesquisa são, primeiramente, validar as dores previstas pela equipe, para então tentar solucionar questões em aberto no planejamento e absorver novas ideias para o projeto. Sendo assim, a pesquisa como um todo deve fornecer meios para os seguintes itens:

- Garantir que a dor dos usuários é real e relevante;

- Entender os detalhes de como acontecem as perdas de animais atualmente:
  - Como ocorrem?
  - Quando ocorrem?
  - Em quais ambientes?
  - Quais os planos de ação mais comuns?
  - Quais são os resultados finais?
  - Impressões gerais de quem perde seu pet;
- Entender como é o comportamento médio das pessoas diante de um animal visto na rua:
  - O quanto elas reparam nesses animais?
  - Como elas diferenciam animais de rua de animais perdidos?
  - Que tipo de ação elas estão dispostas a tomar?
- Conhecer outros serviços que atuam no combate do mesmo problema;
- Entender a preferência dos usuários em relação à aplicativos mobile e redes sociais.

#### 4.4.4 Instrumentos

Os instrumentos escolhidos para realizar o estudo dos usuários foram entrevista e questionário. Optou-se pela entrevista como método qualitativo e atitudinal para obter-se uma compreensão mais aprofundada dos usuários, de modo a entender as dores destes que podem ser sanadas através do design do sistema a ser desenvolvido.

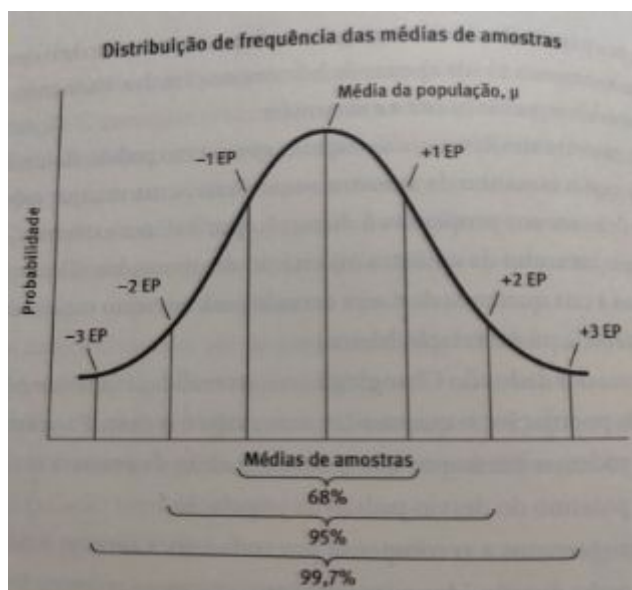
#### 4.4.5 Amostra

O tamanho da amostra é bastante relevante principalmente para a pesquisa quantitativa, que será feita através do questionário. Para encontrar o

tamanho de amostra adequadamente, existem alguns passos e escolhas feitos pelo grupo:

1. **Margem de erro e nível de confiança desejado:** antes de iniciar a coleta dos dados, precisamos saber quanto podemos confiar nos mesmos e qual a margem de erro dos resultados. Com isso, optou-se por uma margem de erro de 10 pontos percentuais, com um nível de confiança de 95%.
2. **Calcular o erro padrão dado o nível de confiança:** para um nível de confiança de 95%, sabe-se pela distribuição da frequência da média das amostras (advinda do teorema do limite central, que prevê que as médias das amostras formam uma distribuição normal em volta da média da população) que a média das amostras deve se encontrar entre 2 erros padrões, como pode-se perceber pela figura abaixo:

Figura 15: Distribuição de frequência das médias das amostras



Fonte: Estatística: O que é, Para que Serve, Como Funciona (WHEELAN, 2016)

Com isso,  $2EP = 10\%$ , que é a margem de erro escolhida, o que resulta em  $EP = 0,05$ .

3. **Calcular o tamanho da amostra (n):** sabe-se que a fórmula de cálculo do erro padrão é

$$EP = \frac{\sigma}{\sqrt{n}}$$

Nesse caso, não temos o desvio padrão da população, porém podemos usar como aproximação o desvio padrão da amostra, que para uma pesquisa com múltiplas questões, considera-se a proporção da amostra para cada pergunta como 50%, então esse desvio resulta próximo a 0,5. Com isso, o tamanho da amostra resulta  $n = 100$ .

Portanto, deveriam ser coletadas 100 respostas ao questionário, com uma margem de erro de 10 pontos percentuais e nível de confiança de 95%. Entretanto, algo que deve ser levado em conta é o viés da amostra. Dado o método de coleta, que foi online, a amostragem não será aleatória, pois o questionário será enviado pelos integrantes do grupo. Com isso, a amostra provavelmente terá um viés de seleção, não permitindo que façamos inferências a respeito da população total, mas cumprirá a função de esclarecer, quantitativamente, hipóteses formuladas a respeito dos usuários, colaborando com o intuito do projeto.

Devido à limitação de tempo por ter que apresentar os resultados na matéria de IHC, a amostra obtida foi menor do que a planejada. Para entrevistas, obtivemos 11 respostas e para o questionário, 64.

#### 4.4.6 Consentimento

O termo de consentimento foi desenvolvido em aula, adaptado com o andamento do projeto e foi utilizado tanto no questionário como na entrevista. Ele pode ser visto no apêndice A presente neste documento.

#### 4.4.7 Resultados

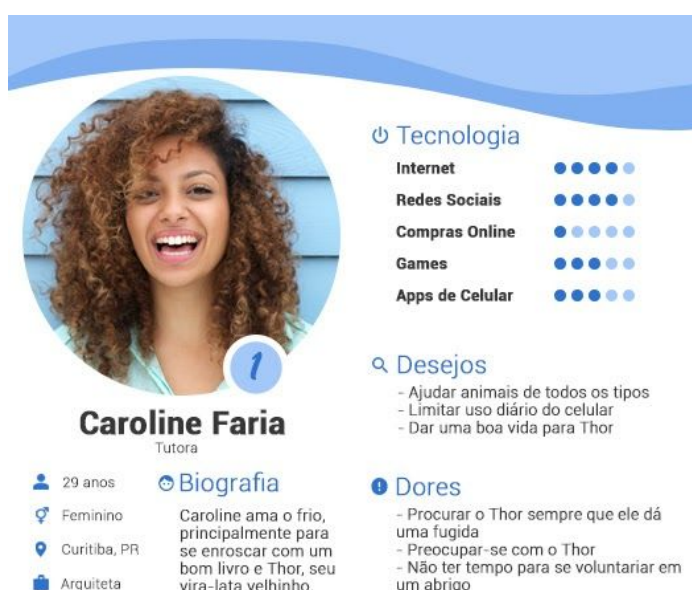


Como mencionado na seção da amostra, obteve-se 64 respostas ao questionário enviado. Tal questionário foi composto por três seções: uma que pedia dados pessoais como nome, idade, gênero e se já possuiu ou possui animais; outra mais voltada aos tutores de animais, se já perderam algum, quantas vezes isso aconteceu, o que foi feito, dentre outros. Por último, a seção dos resgatadores aborda questões sobre já ter visto animais na rua e suas respectivas reações a essa situação.

Já na entrevista, feitas com 11 indivíduos, o formulário preparado teve quatro seções: uma coletando dados pessoais como gênero, idade e quantos pets costuma ter, outra sobre a perda de animais (como perdeu, se conseguiu encontrá-lo, dentre outros), uma seção sobre encontro de animais (se consegue diferenciar um animal perdido de um de rua), e a última seção foi mais relacionada ao aplicativo em si, se a pessoa baixaria ou preferiria uma nova funcionalidade em redes sociais já existentes.

A partir dos resultados obtidos por esses instrumentos e também com o apoio das informações obtidas dos grupos de Facebook, definimos duas personas e suas respectivas jornadas.

Figura 16: Persona de tutora - Caroline Faria.



Fonte: elaborado pelos autores.

Figura 17: Jornada de tutora - Caroline Faria



Fonte: elaborado pelos autores.

Figura 18: Persona de resgatadora - Aparecida Santos



Fonte: elaborado pelos autores.

Figura 19: Jornada de resgatadora - Aparecida Santos



Fonte: elaborado pelos autores.

## 4.5 DISCIPLINAS ACELERADORAS

No primeiro semestre, o projeto foi desenvolvido em conjunto com duas matérias:

1. Laboratório de Software II, em que foram desenvolvidos os requisitos do sistema, o Diagrama de Osterwalder e uma implementação inicial de metade do projeto, focado nos animais perdidos.
2. Interação Humano-Computador, onde foram realizadas as pesquisas de usuários, e desenvolvidas as personas e jornadas de usuário.

## 5 ESPECIFICAÇÃO DE REQUISITOS DO SISTEMA

Como foi dito, o sistema a ser desenvolvido neste projeto tem duas frentes principais, a de adoção de animais e a da busca por animais perdidos. Portanto, cada uma dessas frentes foi planejada como fluxo de negócio individual. A seguir é mostrado o entendimento do problema apresentado em cada frente e o plano para tratamento pela plataforma aqui sendo desenvolvida.

### 5.1 ADOÇÃO DE ANIMAIS

A ideia de montar um banco de dados único e de fácil acesso ao usuário é incentivar a adoção de animais. Para tal, é ideal a adesão de diversos abrigos e ONGs de animais. Como o projeto abrangeria diversos estabelecimentos, a interface de uso será construída de forma que seu uso seja muito intuitivo, não necessitando assim que os utilizadores sejam extensamente treinados em seu manuseio. Porém, deveriam ser aceitos apenas estabelecimentos que sejam comprovadamente abrigos de animais, precisando ser pré-aprovados antes de se realizar sua adesão, para evitar qualquer tipo de estelionato que possa ocorrer. Essa conferência não foi implementada na versão atual do projeto, mas, caso haja liberação para o público, é algo necessário de se desenvolver.

Do lado dos estabelecimentos, há uma opção para adicionar animais com foto e seus atributos, como idade, pelo, tamanho, entre outras características. Estes dados podem ser alterados no futuro - o usuário mantém acesso completo aos animais que existem no seu banco de dados. Caso um animal seja adotado, sua situação deve ser alterada, mostrando assim casos de sucesso do aplicativo.

Do lado do usuário que deseja adotar, ele idealmente terá um perfil de características para o animal que quer adotar. Estes dados serão utilizados para classificação dos animais através do perfil criado pelo abrigo. Esta lógica visa encontrar uma parceria melhor e aumentar a chance da adoção ser realizada e mantida.

### **5.1.1 Fluxo de cadastro de animal para adoção**

Quando um usuário deseja adotar um animal, deve-se garantir que o processo seja o mais sem atrito possível uma vez que os abrigos necessitam capturar todas as oportunidades que aparecem. Para tanto, foi elaborada uma lista com as principais características desejadas para um usuário na hora de adotar um animal que serão comparadas em cada um dos animais disponíveis para adoção na plataforma.

A partir do momento em que o abrigo tem acesso à plataforma, ele poderá realizar o cadastro de animais, preenchendo as características e subindo os mesmos na sua plataforma. Uma outra opção que pode ser adicionada é cadastro em massa de animais, onde o abrigo poderá alavancar a sua própria base de dados e preencher uma tabela fornecida pelo aplicativo com as características dos seus animais e posteriormente subir essa mesma tabela, criando um cadastro para cada instância preenchida na tabela. Para fins de projeto, será implementada somente a primeira opção de cadastro. Após enviadas, as instâncias dos abrigos serão cadastradas na base de dados do PetFinder, onde posteriormente poderão ser editadas pelo próprio abrigo, encontradas em buscas de adoção e encontradas em buscas de animais perdidos.

### **5.1.2 Fluxo de busca por animal para adoção**

Caso um usuário acesse a plataforma com a intenção de adotar um animal, ele terá que selecionar a opção de busca por adoção na tela inicial, e será então redirecionado para uma página onde terá visibilidade dos animais disponíveis para adoção na base de dados e poderá escolher dentre algumas opções para buscar animais.

Primeiramente, o usuário poderá filtrar o animal por características, como raça, porte, cor e outros dados cadastrados pelo abrigo. O usuário poderá também filtrar por localização do abrigo onde o animal se encontra, uma vez que

muitas vezes o usuário pode não estar disposto a percorrer longas distâncias pelo animal escolhido.

Após escolhido um animal, o usuário deverá entrar em contato com o abrigo de que ele se interessa pelo animal, para decidirem horários e lugares para cumprir as responsabilidades legais que envolvem a adoção e levar o animal para casa.

## 5.2 ENCONTRAR ANIMAIS PERDIDOS

Um outro espectro de valor do sistema será a possibilidade de cadastro e pesquisa por animais perdidos. A ideia principal é aproveitar a plataforma de análise de imagens para ajudar usuários que perderam seus animais de estimação, utilizando a rede criada pelo PetFinder para possivelmente localizar onde o animal poderia ter sido encontrado por outros usuários do aplicativo.

O sistema é intencionado com o seguinte funcionamento: usuários que encontram animais perdidos na rua poderão fazer *upload* de fotos e informações sobre eles, como raça, porte, localização onde foi visto, cor e outras características visíveis. Assim, quando um usuário perde seu animal de estimação, ele pode fazer *upload* de foto do seu animal e última localização conhecida, e o sistema retorna as correspondências de animais em seu registro, com porcentagem de match e outras informações pertinentes.

Para o usuário comum, existirão duas opções de interação, sendo elas “Perdi meu pet” e “Encontrei um pet perdido”. Em ambas as interações, o usuário deve preencher o máximo de informações do animal que conseguir, e o sistema retornará as correspondências e as opções de contato com o registrador da instância correspondente.

### 5.2.1 Fluxo de cadastro de animal encontrado

Quando um usuário encontrar um animal, o mesmo deverá logar em sua conta e escolher a opção de registrar pets perdidos, onde ele deverá escolher

entre registrar um pet que ele mesmo perdeu, ou registrar um pet perdido que ele encontrou.

Após escolher a opção de registrar um pet que o usuário encontrou perdido, ele deverá colocar o máximo de informações possível que ajude a descrever o animal encontrado como tipo de animal, raça, cor, se tinha ou não coleira, endereço em que foi visto, foto, e outras informações relevantes para quem estiver procurando. Como pode ser difícil identificar muitas informações de um animal perdido, esse preenchimento não será totalmente obrigatório.

Uma vez completa e enviada a descrição do animal encontrado, o usuário receberá uma confirmação das informações que preencheu, uma opção de editar o seu preenchimento e resultados de instâncias correspondentes ao animal preenchido na base de dados. Os resultados serão apresentados de acordo com a correspondência entre as instâncias, ou seja, a foto e a descrição do animal cadastrado será conferida com todos os animais presentes na base, tanto por uma conferência da análise de imagem quanto pelas características dos animais.

Se o usuário achar que um dos resultados é o animal que ele encontrou, ele poderá avisar a pessoa responsável pelo cadastro da correspondência. Caso ele não encontre nenhum que pareça ser correspondente, ele poderá voltar para a tela inicial, onde poderá acompanhar o status e rever matches quando desejar.

### **5.2.2 Fluxo de cadastro de animal perdido**

Caso um usuário tenha perdido seu animal, o mesmo deverá logar em sua conta e escolher a opção de registrar pets perdidos, onde ele deverá escolher entre registrar um pet que ele mesmo perdeu, ou registrar um pet perdido que ele encontrou.

Após escolher a opção de registrar um pet que ele perdeu, o usuário será redirecionado para uma tela onde deverá preencher o máximo de características possíveis que ajudem a identificar o animal.

Uma vez completa e enviada a descrição do animal perdido, o usuário receberá uma confirmação das informações que preencheu, uma opção de editar

o seu preenchimento e resultados de instâncias correspondentes na base de dados ao animal preenchido. Os resultados serão apresentados de acordo com a correspondência entre as instâncias, ou seja, a foto e a descrição do animal cadastrado será conferida com todos os animais presentes na base, por uma conferência da análise de imagem e pelas características preenchidas dos animais.

Se o usuário achar que um dos resultados é o animal que ele perdeu, ele poderá avisar a pessoa responsável pelo cadastro da correspondência de acordo com as opções de privacidade escolhidas por ambos os usuários e esperar pela resposta. Caso ele não encontre nenhum que pareça ser correspondente, ele poderá voltar para a tela inicial, onde poderá acompanhar o status do animal que cadastrou.

### 5.3 REQUISITOS

Depois de planejado cada uma das verticais do projeto, foi feito um desenho do funcionamento das duas frentes de trabalho, a partir do qual foi possível levantar os requisitos funcionais e não funcionais necessários para o projeto completo.

Em seguida, os requisitos (tanto funcionais quanto não funcionais) foram elencados de acordo com a sua relevância e necessidade para a confecção do sistema e houve a elaboração de diagramas, a fim de aprofundar o planejamento de como abordar esses requisitos durante o desenvolvimento.

#### 5.3.1 Requisitos Funcionais

Os requisitos funcionais listados abaixo podem estar ligados a uma ou ambas frentes de projeto. Foram omitidos requisitos comuns a quase todos os projetos desse tipo, como cadastro e edição de usuário.

A lista de requisitos foi separada em três níveis de prioridade: alta, média e baixa; dentro de cada nível, não há ordenação entre os itens.



### 5.3.1.1 Prioridade alta

Esta prioridade traz os requisitos imprescindíveis para o objetivo da plataforma.

- **Interface de usuário e interface do abrigo:** um abrigo deve ter acesso a algumas *features*, enquanto um usuário comum deve ter acesso a outras diferenciadas, por exemplo a busca por animal para adoção.
- **Buscar animais para adoção:** mecanismo de busca com filtros para o usuário encontrar o tipo de animal que procura, de acordo com as características previamente decididas.
- **Buscar animal perdido:** usuário tem opção de subir fotos de animais perdidos ou listar características e como retorno a plataforma mostra os animais que batem com a descrição listada ou extraída da foto.
  - **Tempo total de processamento da imagem durante o cadastro:** o cadastro de animais deve demorar no máximo 15 segundos.
- **Cadastrar animal encontrado na rua:** registro de animais encontrados na rua por usuário, para funcionar em conjunto com a busca por animal perdido.
- **Match por análise de imagem:** a busca por animal perdido ou para adotar é feita pela comparação dos animais cadastrados, quando eles têm imagens salvas.
  - **Tempo total de comparação das imagens durante o match:** o tempo de demora para match tem um *soft limit* de 1 minuto e um *hard limit* de 3 minutos. Este tempo mais elevado é justificável por usuários que devem ter perdido seu cachorro e estão ansiosos para encontrá-los - eles não

devem desistir da procura por uma demora de 3 minutos. Ainda assim, é preciso ser transparente sobre quanto tempo irá demorar a pesquisa completa.

- **Precisão do algoritmo de análise de imagem no match:** a precisão do match por imagem não deve ser obrigatoriamente tão elevada pois é balanceada pelas outras características. Assim, estima-se um mínimo de precisão de 70%.
- **Match por características:** a busca por animal perdido ou para adotar leva em conta todas as características inseridas no cadastro, não só a foto. O match final é calculado com um peso entre o match das imagens e o das características.
- **Histórico de cadastros:** o usuário pode acessar um histórico de todos os animais cadastrados por ele, para editar ou ver os *matches* de cada animal.

#### 5.3.1.2 Prioridade média

Esta prioridade foca em itens que deixariam a plataforma muito mais adequada, mas não impedem o funcionamento essencial.

- **Escolha entre cadastro simples e completo:** no cadastro de animal encontrado na rua, para levar em consideração a possibilidade de o usuário estar na rua e com pressa, deve ser apresentada uma opção de cadastro rápido, apenas com as informações principais. Ainda assim, deve ter a opção de cadastro completo, pois sempre é melhor ter mais dados para calcular o *match* com mais precisão.
- **Registrar animais em grande quantidade (para abrigos):** registro de animais disponíveis para adoção em abrigos via upload de base de dados.

- **Configurações de privacidade para usuários:** usuário escolhe como e com quem deseja compartilhar suas informações de contato, como e-mail ou chat interno.
- **Entrar em contato com outros usuários:** possibilidade de abrigos e usuários comuns entrarem em contato com outros usuários da plataforma, de acordo com suas preferências de privacidade.
- **Compartilhar em redes sociais:** botões para compartilhamento de informações (animais para adoção, adoções realizadas, etc) em diversas redes sociais.
- **Geolocalização:** o match entre o animal perdido cadastrado e os do BD leva em conta a distância entre as localizações de cada um, além de isso ser considerado na busca de adoção.
- **Notificações:** quando houver um cadastro novo de um animal que dê match razoavelmente alto com um já cadastrado previamente, deve-se avisar por notificação o usuário.

### 5.3.1.3 Prioridade baixa

Nesta prioridade, são listados os requisitos que seriam interessantes de se implementar, mas não são necessários.

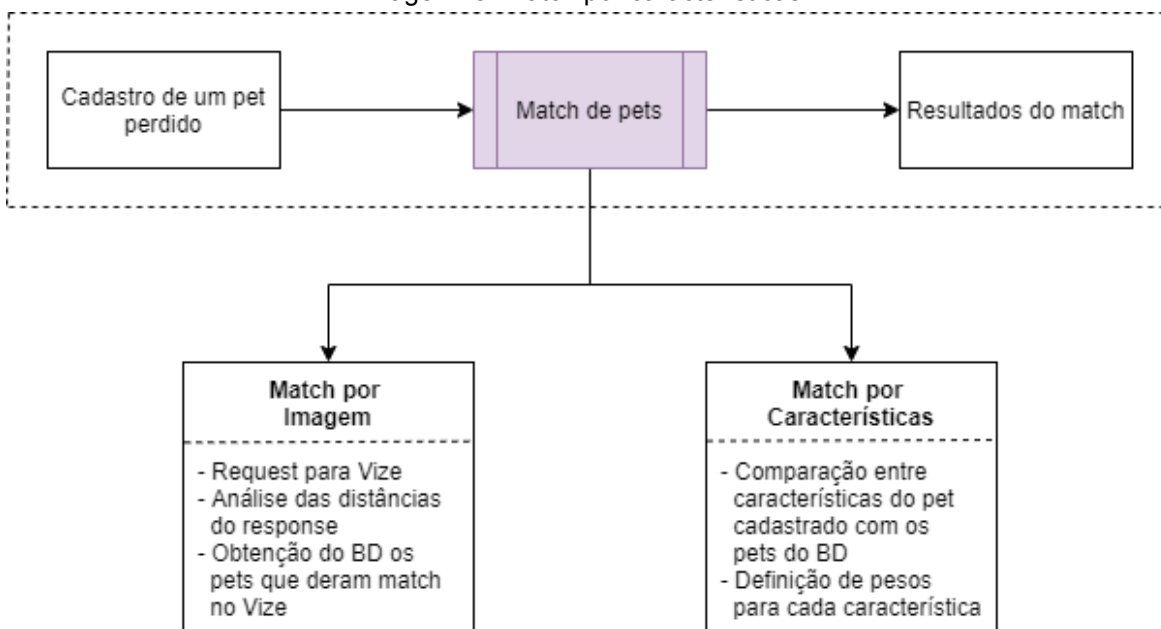
- **Abrigo colocar aviso de usuário quando já devolveu/maltratou:** possibilidade de levantamento de uma *flag* de usuário pelo abrigo, caso o usuário tenha devolvido ou maltratado algum animal que adotou previamente.
- **Banir usuário:** feature para excluir um usuário da possibilidade da utilização dos serviços do app.
- **Classificação dos animais por análise de imagem:** feature interna para análise de imagens e extração de características não apresentadas durante o cadastro mas possivelmente obtidas pela imagem, como cor, raça e porte.

- **Listagem de abrigos e instituições parceiras:** lista com informações essenciais de ONGs, abrigos e instituições parceiras.
- **Avaliar experiência:** feature para usuário avaliar a plataforma, após utilizar algum dos serviços.
- **Colocar *timestamp* nos eventos de carga e de busca:** colocar *flag* de horário/data nos *uploads* de usuários/abrigos.
- **Favoritar animais:** é possível favoritar animais, tanto para adoção quanto originados pelo match. No perfil do usuário deve ter acesso à lista de favoritos.
- **Chat interno:** ter um chat interno entre usuários seria útil para facilitar a comunicação quando houvesse um match ou uma procura por adoção, e diminuiria a carga de privacidade necessária pois não estaria compartilhando informações entre usuários.
- **Dashboard de casos:** os usuários de abrigos e usuários administradores devem ter um dashboard com o número de casos de adoção feitos pelo aplicativo, entre outros dados, para poder analisar o sucesso de utilização da plataforma.

#### 5.3.1.4 Diagramas

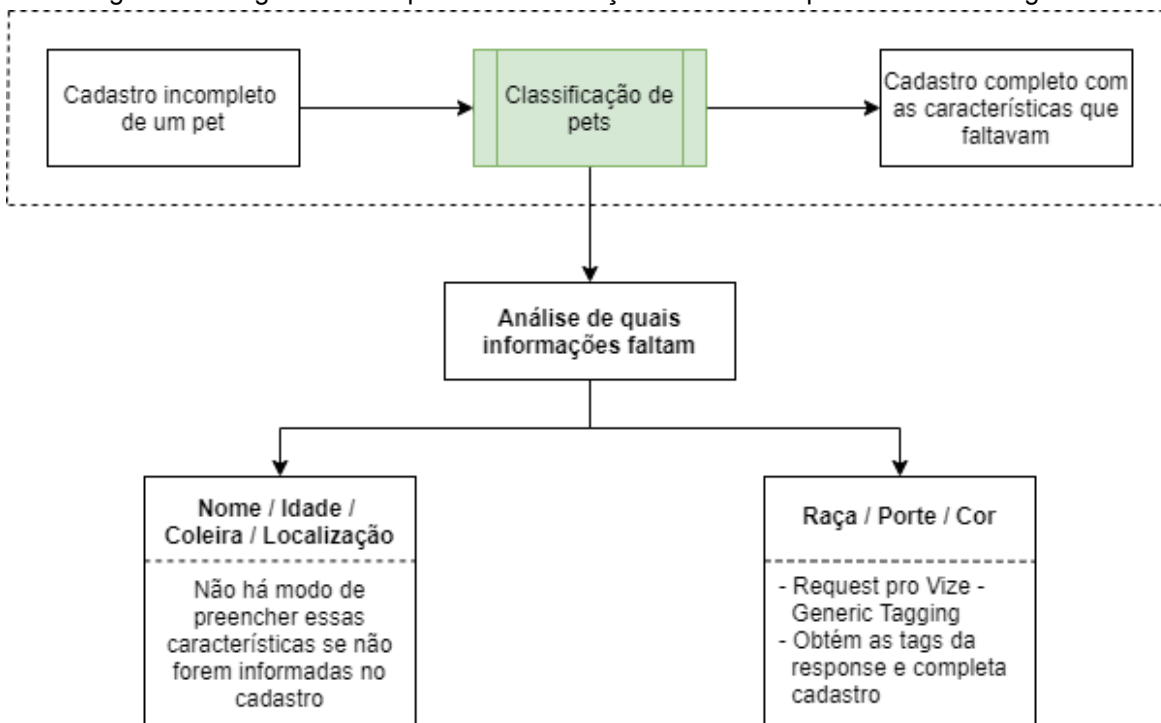
Dentre os requisitos funcionais levantados, alguns foram selecionados para a criação de diagramas. A seleção foi feita considerando-se a prioridade atribuída aos requisitos e a complexidade da abordagem.

Figura 20: Diagrama do requisito de “Match de pets”, que engloba o “Match por análise de imagem” e “Match por características”



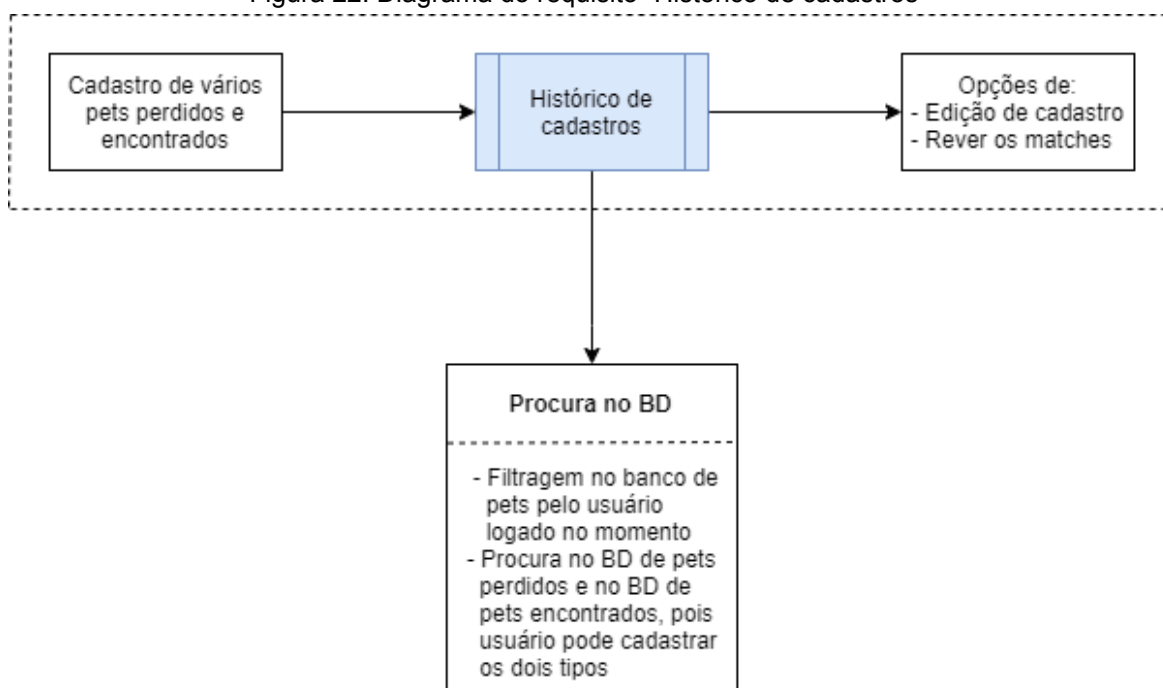
Fonte: elaborado pelos autores.

Figura 21: Diagrama do requisito “Classificação dos animais por análise de imagem”



Fonte: elaborado pelos autores.

Figura 22: Diagrama do requisito “Histórico de cadastros”



Fonte: elaborado pelos autores.

### 5.3.2 Requisitos Não Funcionais

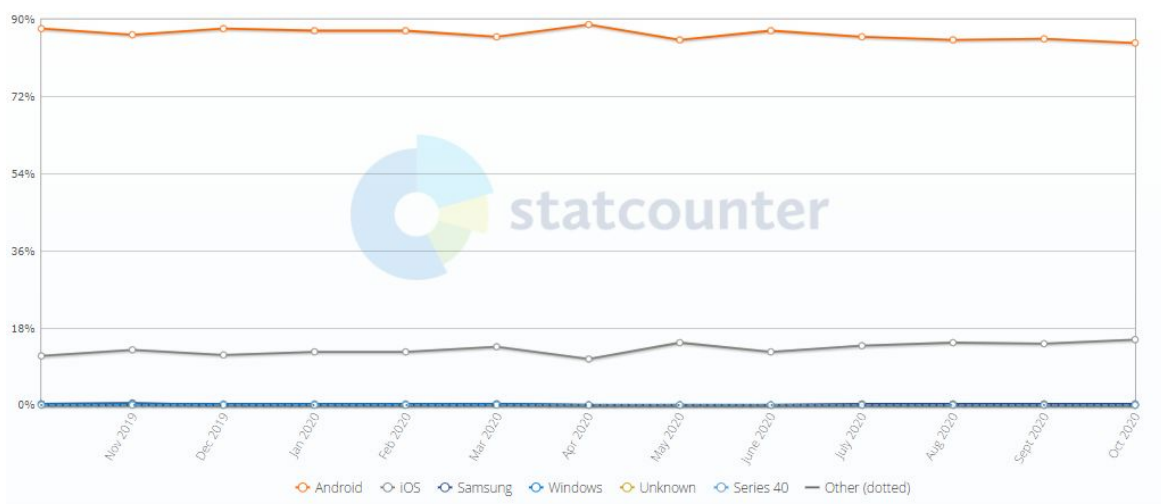
O grupo encontrou diversos requisitos não funcionais aplicáveis ao sistema, cada um apresentando diferentes graus de prioridade para serem implementados. Estes requisitos são listados a seguir:

- **Portabilidade:** este projeto pretende ser implementado apenas para Android, dado que quase 85% dos celulares do Brasil utilizam este OS, ao invés de iOS, de acordo com Global Stats. Numa continuidade pós projeto de formatura, visando uma adesão ainda maior, seria indicada a criação de um ambiente web, possibilitando, por exemplo, que abrigos possam utilizar o sistema sem que haja necessidade do uso de seus aparelhos pessoais.

Figura 23: Gráfico exibindo a porcentagem de sistema Android e iOS nos celulares do Brasil, sendo que Android está por volta de 85% e iOS está com 15%.

### Mobile Operating System Market Share Brazil

Oct 2019 - Oct 2020



Fonte: <https://gs.statcounter.com/os-market-share/mobile/brazil>

- **Tempo Real:** o tempo de pesquisa para animais de adoção deve ser relativamente baixo, não passando de um minuto - para o usuário não perder o interesse - em 90% das vezes. Enquanto a pesquisa por animais perdidos pode ser ligeiramente mais demorada, como se trata de uma pesquisa mais extensa e completa. O tempo de espera envolve diversos elementos:
  - **Tempo de comunicação entre frontend e backend:** pode variar com a internet do usuário.
  - **Tempo de processamento do algoritmo de comparação de imagem:** quanto maior o banco de comparação, maior é o tempo de processamento, por isso pode ser necessário colocar tags para diminuir o número de animais a serem comparados.
  - **Tempo de match por características:** do mesmo modo que o algoritmo de análise de imagem, deve-se comparar com todos do banco de comparação.

- **Log - Registros Estatísticos:** deve-se manter registro das adoções concluídas com sucesso e dos animais encontrados. Por dois motivos:
  - É possível saber a efetividade e assertividade do nosso produto, e também ajudar a realizar calibrações em nossas classificações.
- **Acessos Múltiplos:** o sistema deve manter conexão e responder em tempo hábil várias requisições. A medição de quantos acessos múltiplos devem ser permitidos deve ser feita com base na região em que a plataforma será utilizada (cidade de São Paulo, estado, país).

Além dos requisitos citados acima, foram selecionados e aprofundados três requisitos não funcionais por serem de alta prioridade para o desenvolvimento efetivo do sistema: a segurança, a usabilidade e a portabilidade.

#### 5.3.2.1 Segurança

Para cumprir este requisito optou-se por um sistema de controle de permissões para assegurar que acessos não-autorizados ocorram, por meio de autenticação e autorização de usuários.

Com o propósito de prevenir que ataques ao sistema ocorram, poderia-se utilizar recursos de rastreabilidade, a fim de checar quaisquer atividades suspeitas realizadas pelos usuários. Por fim, para que se mantenha a confidencialidade dos dados do usuário, algoritmos de criptografia podem ser utilizados.

Por meio dos atributos mencionados, se torna possível, por exemplo, assegurar que, ao subir uma imagem de um animal encontrado, a localização do usuário será mantida confidencial.

Para cumprir estas necessidades, serão implementadas:

- Uso de servidor AWS (EC2) com firewalls configurados limitando o tráfego externo no servidor.



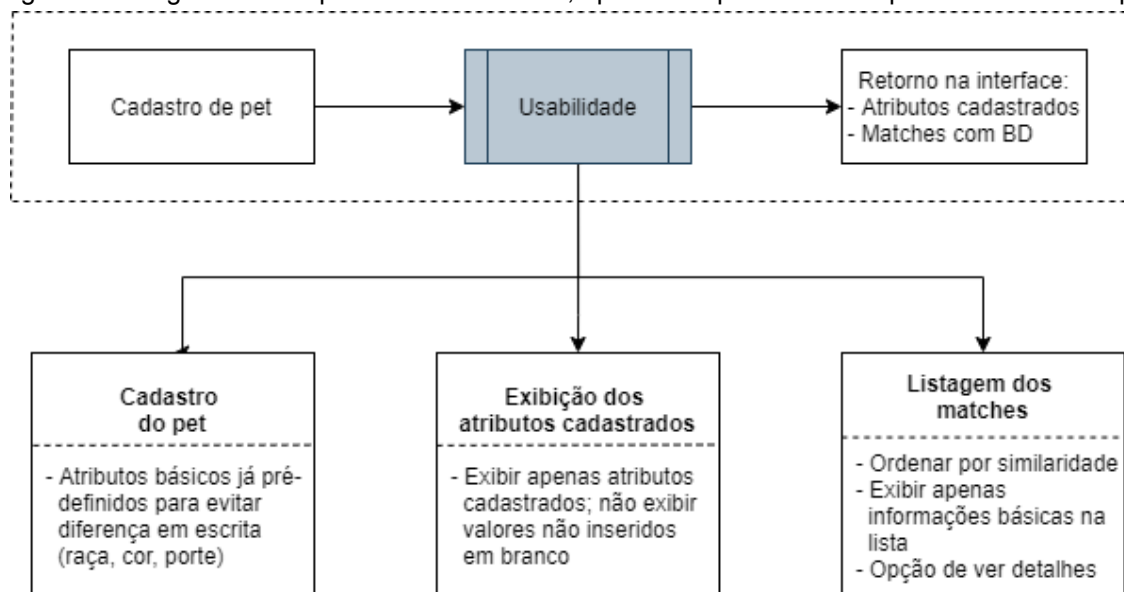
- Uso de S3 private que poderá ser acessado pela EC2 através de um perfil criado na conta (role). Assim, não haverá nenhuma senha no código. Caso o servidor seja invadido, as informações estarão seguras no S3 e o invasor não terá acesso a nenhuma informação dos usuários.
- Uso de comunicação segura através de comunicação HTTPS e não HTTP, em conjunto com uma chave privada e um certificado digital para autenticar o usuário.
- Somente permitir o cadastro de abrigos que tenham sua existência confirmada.
- Possuir canal de denúncia de usuários, caso algum apresente comportamento indevido e não respeitando as regras de uso do aplicativo.

#### 5.3.2.2 Usabilidade

Este requisito foi considerado pelo grupo de extrema importância. Por meio dele, é possível assegurar que os usuários utilizarão o aplicativo sem grande esforço. Para que isso seja possível, devem ser utilizadas interfaces que funcionem de maneira instrutiva e clara, fazendo uso de *affordances*, que tornam a jornada de um usuário simplificada e intuitiva.

Vale ressaltar que outros artifícios podem ser utilizados para gerar uma satisfação maior das pessoas que farão uso do sistema projetado, como, por exemplo, o uso de busca por palavras-chave.

Figura 24: Diagrama do requisito de usabilidade, aplicado especificamente para o cadastro de pet.



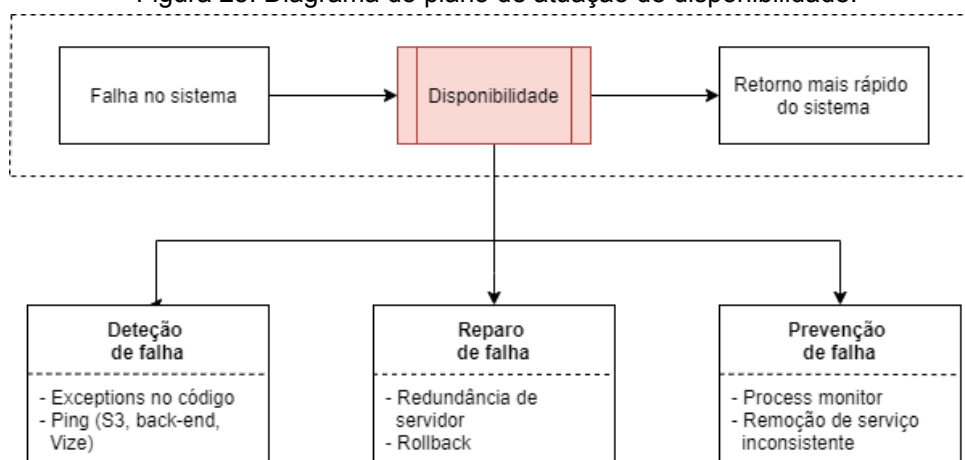
Fonte: elaborado pelos autores.

### 5.3.2.3 Disponibilidade

Por não se tratar de um sistema crítico, não é necessário que ele tenha uma alta disponibilidade garantida. Apesar de não ser um requisito extremamente necessário, a disponibilidade ainda foi levada em consideração e afetou a escolha do host do sistema: a AWS, que tem 2 grandes benefícios pelo lado de disponibilidade:

- Caso ocorra algum problema com o servidor, é fácil e rápido substituir o que está com defeito por um novo.
- Utilização de *Load Balancers*, que monitoram a utilização da aplicação e podem subir ou derrubar servidores conforme o fluxo de usuários demanda.

Figura 25: Diagrama do plano de atuação de disponibilidade.



Fonte: elaborado pelos autores.

## 5.4 PROTÓTIPOS

Com a ideia inicial de como seria a plataforma, foi realizada uma prototipagem básica das telas presentes nos principais fluxos do projeto.

Figuras 26, 27 e 28: Protótipos da tela de login, da tela inicial para usuários comuns e da tela inicial para abrigos.



Fonte: elaborado pelos autores.

Figuras 29 e 30: Protótipo do cadastro de animais por usuários de abrigos, com a funcionalidade de cadastro em massa.



Fonte: elaborado pelos autores.

Figuras 31, 32, 33 e 34: Protótipo do fluxo de cadastro de animal perdido ou animal encontrado.



Fonte: elaborado pelos autores.

Depois, com todos os requisitos funcionais levantados e com a ideia mais concreta de como seria o projeto final, foi desenvolvido um protótipo mais robusto, por meio do MarvelApp (vide item 3.8.3). Este protótipo foi desenvolvido para a disciplina de Interação Humano-Computador, durante a etapa de planejamento de testes de usabilidade, portanto as funcionalidades exibidas neste protótipo não

englobam a frente de adoção, com a participação de abrigos, visada neste trabalho.

Este protótipo, ao contrário da primeira versão mais simplificada, não consiste de apenas imagens, mas consegue simular a utilização de um aplicativo real, com o toque em certas partes da tela direcionando para outras telas. É possível testá-lo neste link: <https://marvelapp.com/242j64ag/screen/70418459>.

Figuras 35 e 36: Protótipos do requisito funcional *Escolha entre cadastro simples e completo*, durante o cadastro de animal encontrado. Foi suposto que o cadastro de animal perdido sempre será completo.

Marvel 13:00

← Cadastro de animal encontrado

Raça  
Selecione uma raça

Último local visto

Cor  
Selecione uma cor

Estava com coleira?

+

INFORMAÇÕES AVANÇADAS

FINALIZAR CADASTRO

Página Inicial Notificações Mensagens Perfil

Marvel 13:00

← Cadastro de animal encontrado

INFORMAÇÕES AVANÇADAS

Nome

Porte  
Selecione um porte

Gênero  
Selecione um gênero

Rabo  
Selecione um tamanho de rabo

Pelo  
Selecione um tipo de pelo

Castração  
Parece ser castrado?

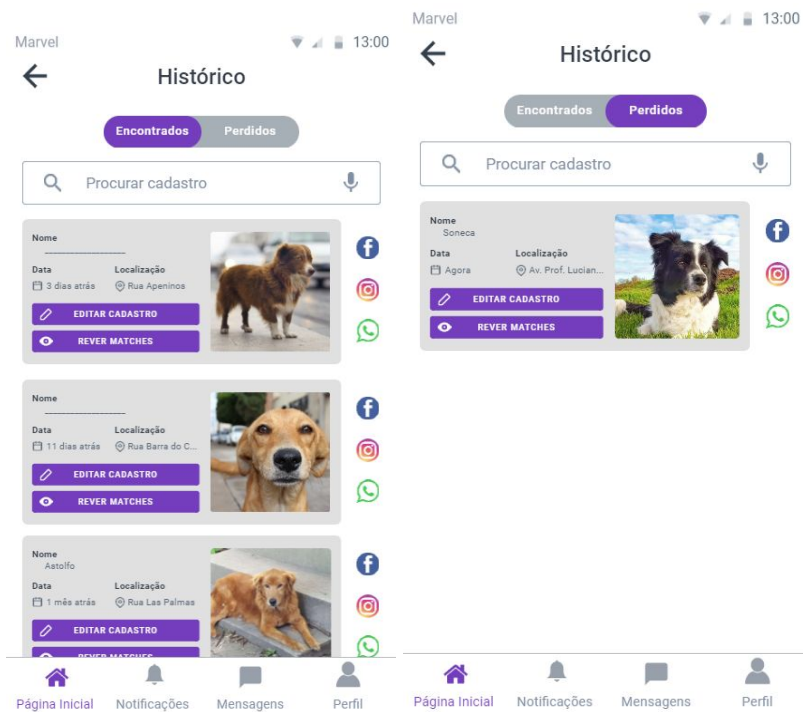
Idade que aparenta

FINALIZAR CADASTRO

Página Inicial Notificações Mensagens Perfil

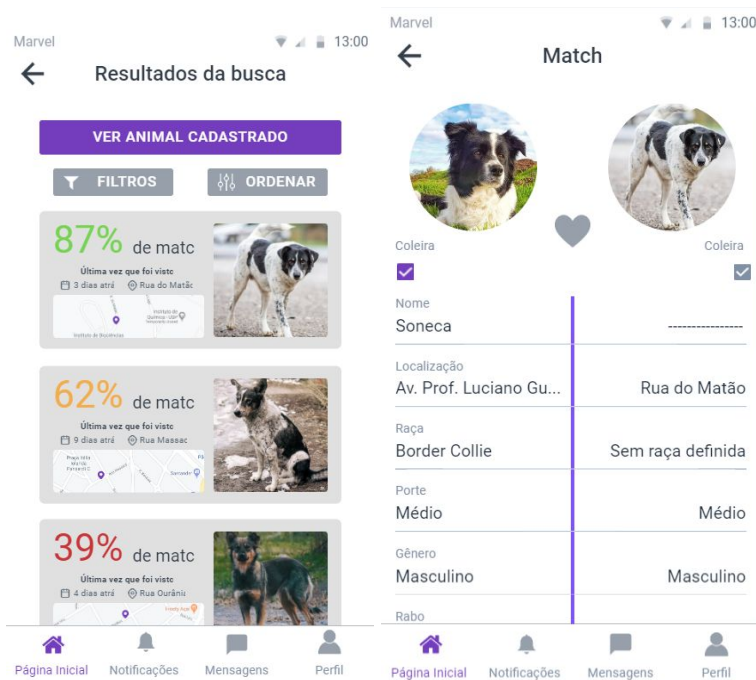
Fonte: elaborado pelos autores.

Figuras 37 e 38: Protótipos da tela de histórico, com os cadastros divididos entre animais encontrados e perdidos.



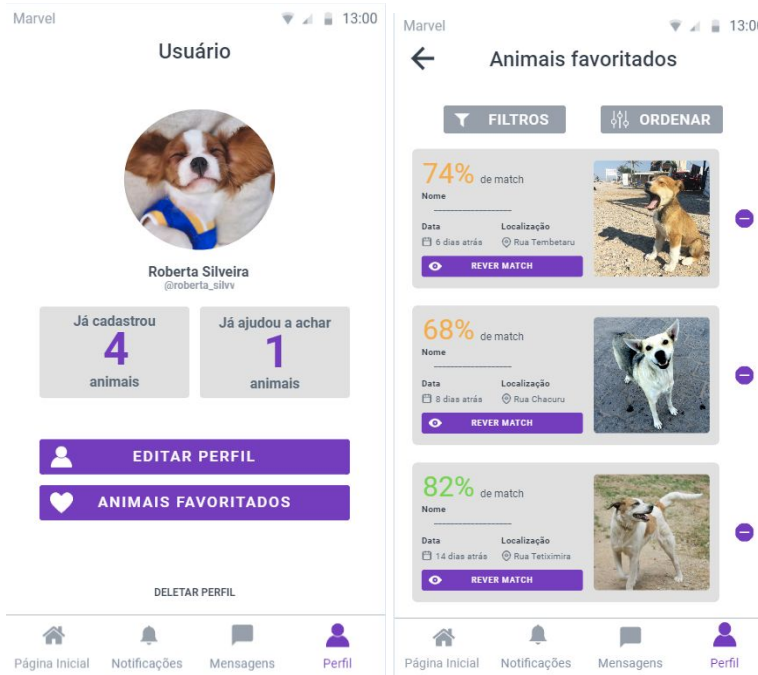
Fonte: elaborado pelos autores.

Figuras 39 e 40: Protótipos da tela de análise dos resultados de match e da de visualização de detalhes de um único match.



Fonte: elaborado pelos autores.

Figuras 41 e 42: Protótipos da tela de perfil e da de animais favoritos, acessada pelo perfil.



Fonte: elaborado pelos autores.

## 6 PROJETO E IMPLEMENTAÇÃO

Além da decisão de fazer a implementação do projeto em sprints, seguindo a metodologia Scrum, também foi decidido a organização de fluxos, ou grupos de funcionalidades, a serem implementados separadamente. A ideia inicial é ter três agrupamentos de *features*: de pet perdido, pet para adoção e análise de imagem.

Como foi dito no item 4.5, no primeiro semestre letivo de 2020 os integrantes do grupo participaram de duas matérias que agiram como aceleradoras para este projeto. Dentre elas, a matéria Laboratório de Software II tinha como método de avaliação a entrega de um aplicativo funcional ao fim do semestre, portanto foi escolhido um dos dois fluxos adequados para a implementação durante a disciplina. A parte de análise de imagem não casava com a intenção da matéria, então a decisão ficou entre o agrupamento de pet perdido e pet para adoção, concluindo-se como o fluxo de pet perdido.

Nos tópicos a seguir, é descrita a implementação completa deste projeto, mais especificamente sobre a implementação do resultado final, não das versões parciais.

### 6.1 PET PERDIDO

#### 6.1.1 Usuário

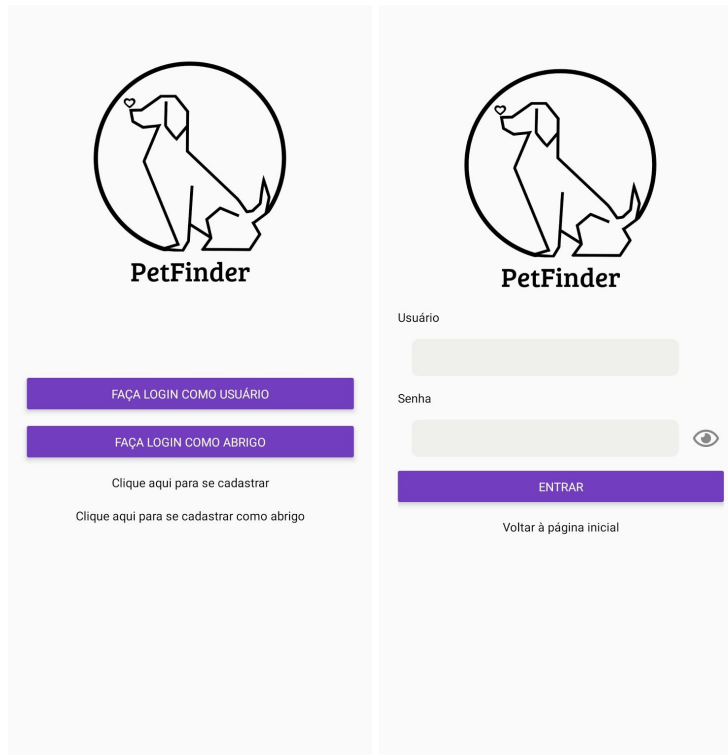
Para criar um projeto completo com as funcionalidades previstas para o fluxo de pet perdido, deve-se implementar também a parte de usuários, com cadastro e login. Portanto, foi desenvolvido o cadastro para o usuário com suas informações básicas e sua interface principal, com as opções mais importantes dos fluxos necessários, como match de animais perdidos e procura por animais para adoção.

##### 6.1.1.1 Página Inicial



Na tela inicial do aplicativo, o usuário tem a escolha entre realizar o login como usuário comum ou abrigo ou, caso ainda não tenha feito seu cadastro, pode seguir para a tela de cadastro, tanto de usuário comum quanto de abrigo.

Figuras 43 e 44: Tela inicial do aplicativo e tela de login de usuário.



Fonte: elaborado pelos autores

Na parte do login, o backend recebe os dados de usuário e senha inseridos e verifica na base se estão corretos. Caso contrário, retorna uma mensagem com o motivo de erro ao usuário.

#### 6.1.1.2 Cadastro de novo usuário

O usuário realiza o seu cadastro escolhendo o seu nome de usuário, que deve ser único no sistema, além disso, existem campos não-obrigatórios que possibilitam que o usuário não compartilhe dados que não se sinta confortável.

Figura 45: Cadastro de novo usuário

← Cadastre-se como usuário

Username

Nome

Sobrenome

Email

Telefone

Idade

Endereço

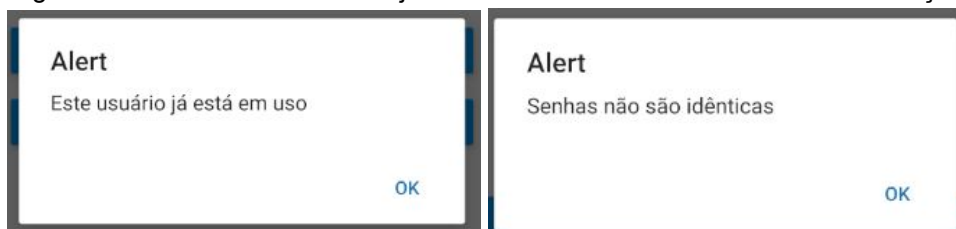
Senha

Confirmar senha

SUBMIT

Fonte: elaborado pelos autores

Figuras 46 e 47: Erro de usuário já existente e senhas diferentes na confirmação



Fonte: elaborado pelos autores.

Code Snippet 4: Atributos de usuário comum no banco de dados.

```
class Users(db.Model):
    username = db.Column(db.String(80), unique=True, nullable=False, primary_key=True)
    name = db.Column(db.String(80), nullable=False, unique=False)
    surname = db.Column(db.String(80), nullable=True, unique=False)
    email = db.Column(db.String(80), nullable=False, unique=False)
    password = db.Column(db.String(80), nullable=False, unique=False)
    age = db.Column(db.Integer, nullable=True, unique=False)
    phone = db.Column(db.Integer, nullable=True, unique=False)
    latitude = db.Column(db.String(80), nullable=True, unique=False)
    longitude = db.Column(db.String(80), nullable=True, unique=False)
    lost_pets = db.relationship('PetLost', backref='user', lazy=True)
    found_pets = db.relationship('PetFound', backref='user', lazy=True)
```

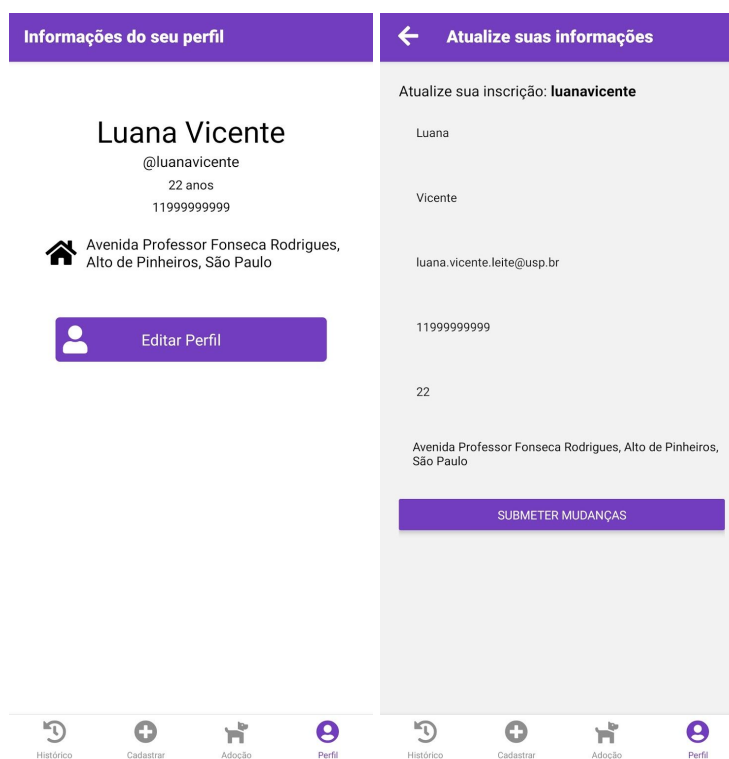
Fonte: elaborado pelos autores

### 6.1.1.3 Tela inicial de usuário

Finalmente, o usuário logado tem acesso à sua interface, que contém um menu inferior com quatro opções:

- **Histórico:** onde há uma listagem dos animais perdidos e encontrados que este usuário já cadastrou;
- **Cadastro:** leva a uma tela que permite o cadastro de animal perdido e encontrado;
- **Adoção:** permite a busca por filtros dos animais de adoção cadastrados pelos abrigos;
- **Perfil:** exhibe as características do perfil do usuário e permite a atualização das informações.

Figuras 48 e 49: Tela do perfil e edição do usuário. É possível ver o menu inferior com as 4 opções explicadas anteriormente.

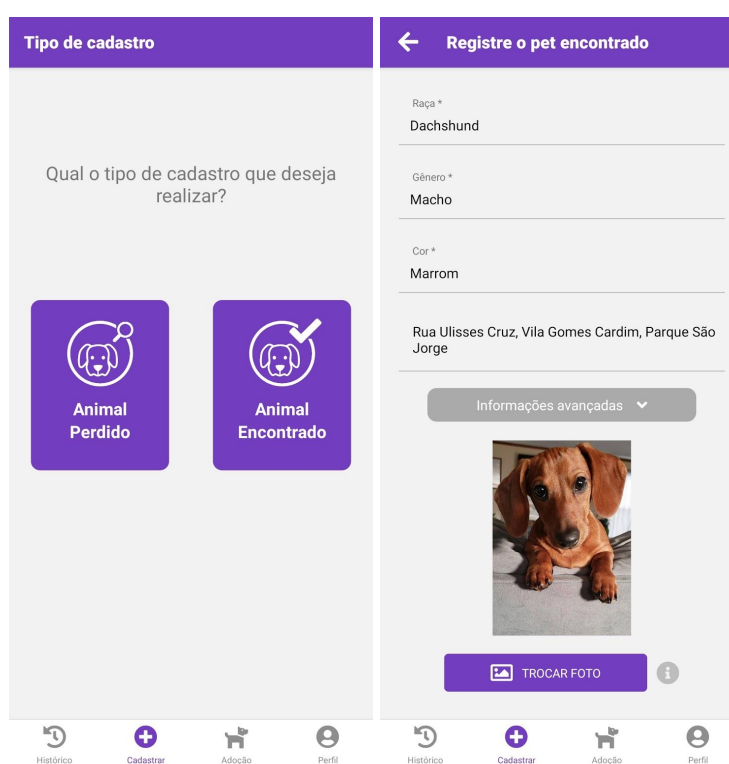


Fonte: elaborado pelos autores

## 6.1.2 Cadastro de animal

Para o cadastro de animais perdidos e encontrados, foi feita uma única tela em que o primeiro item é uma seleção entre “Perdido” e “Encontrado”. Em seguida, há diversos itens que caracterizam o cachorro a ser cadastrado, finalizando com o *upload* de uma imagem tirada na hora ou originada da galeria.

Figuras 50 e 51: Telas iniciais do fluxo de cadastro de animal.

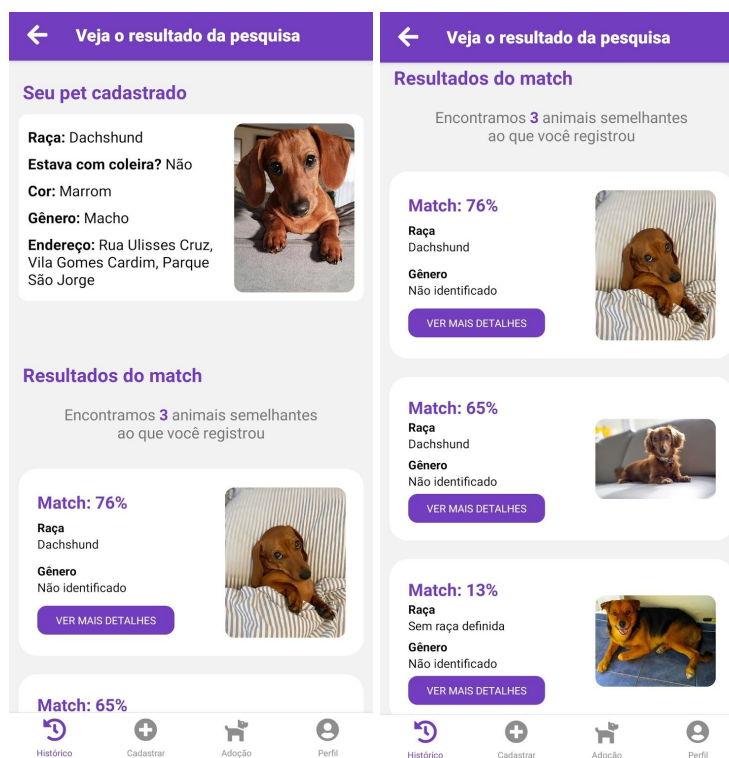


Fonte: elaborado pelos autores.

Para facilitar a comparação entre as características de diversos animais cadastrados, foi decidido que a maioria dos campos seria do tipo *dropdown*, a fim de evitar a escrita variada de informações que teoricamente iriam ter o mesmo significado. Por exemplo, a raça dachshund é comumente se referida, no Brasil, como “salsicha” - é possível que, ao fazer o cadastro de um cachorro dessa raça, o tutor insira o apelido ao invés do nome oficial, o que dificultaria a comparação com cadastros que contenham o nome oficial.

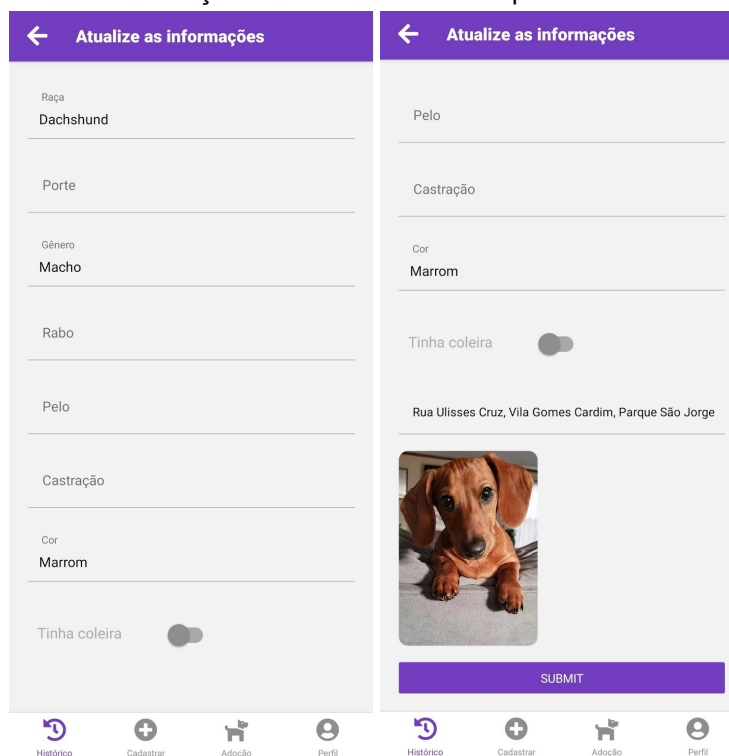
O fluxo do cadastro termina na tela de resultados, que apresenta os *matches* com o BD. Também é possível selecionar um dos animais que deram *match* para ver mais informações sobre ele, dado que na tela de resultados só é apresentada a porcentagem de *match* e algumas informações básicas. Na tela de detalhes do animal, também é possível direcionar para uma tela de visualização do perfil que cadastrou aquele animal, para permitir o contato caso necessário.

Figuras 52 e 53: Tela de resultados.

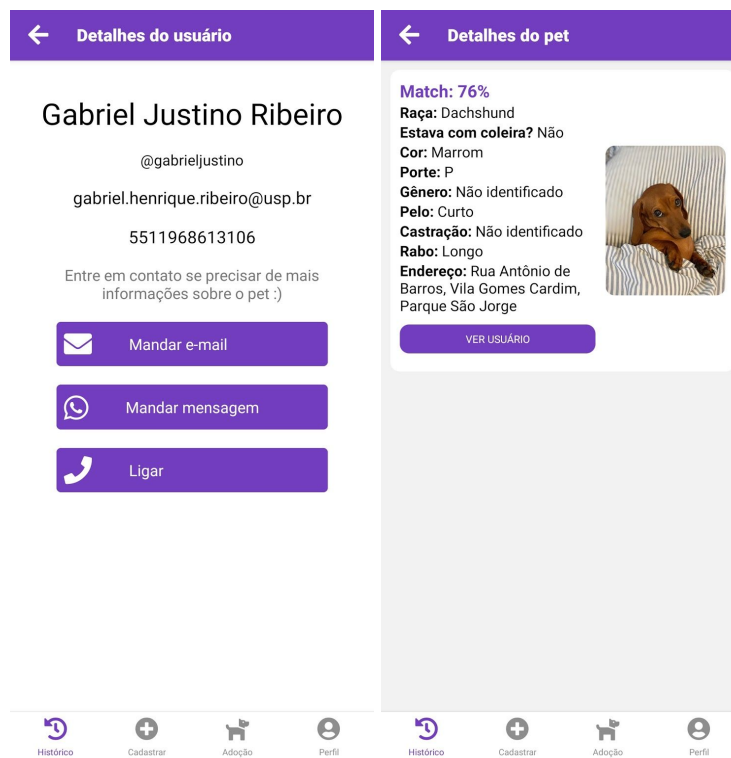


Fonte: elaborado pelos autores.

Figura 54 e 55: Tela de edição dos dados do animal que acabou de ser cadastrado.



Fonte: elaborado pelos autores.

Figura 56 e 57: Detalhes de um dos animais que deram *match* com o cadastrado e seu respectivo usuário.

Fonte: elaborado pelos autores.



```

net.setInput(blob)
layerOutputs = net.forward(ln)
boxes = []
confidences = []
classIDs = []

for output in layerOutputs:
    for detection in output:
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]

        if confidence > CONFIDENCE:
            box = detection[0:4] * np.array([W, H, W, H])
            (centerX, centerY, width, height) = box.astype("int")
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))
            boxes.append([x, y, int(width), int(height)])
            confidences.append(float(confidence))
            classIDs.append(classID)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, CONFIDENCE,
                        THRESHOLD)

if len(idxs) > 0:

    indexes_animals = []
    for i in idxs.flatten():
        if LABELS[classIDs[i]] in ANIMAL_LABELS:
            indexes_animals.append(i)
    if len(indexes_animals) == 0:
        return jsonify({'response': f'Erro: não foi reconhecido nenhum animal na
imagem.', 'code': 417})
    if len(indexes_animals) > 1:
        return jsonify({'response': f'Erro: há mais de um animal na imagem.',
'code': 417})

    for i in indexes_animals:
        (x, y) = (boxes[i][0], boxes[i][1])
        (w, h) = (boxes[i][2], boxes[i][3])

        x = x if x >= 0 else 0
        y = y if y >= 0 else 0
        image_cropped = image[y:(y+h),x:(x+w)]
        return image_cropped

    return jsonify({'response': f'Erro: não foi reconhecido nenhum animal na imagem.',
'code': 417})

```

Fonte: elaborado pelos autores.



Code Snippet 6: Função de preenchimento do fundo da imagem pela cor preta.

```
def getPetImageBlackBackground(image):
    mask = np.zeros(image.shape[:2], dtype="uint8")

    rect = (10, 10, image.shape[1]-20, image.shape[0]-20)

    fgModel = np.zeros((1, 65), dtype="float")
    bgModel = np.zeros((1, 65), dtype="float")

    (mask, bgModel, fgModel) = cv2.grabCut(image, mask, rect, bgModel,
        fgModel, iterCount=ITERATIONS, mode=cv2.GC_INIT_WITH_RECT)

    values = (
        ("Definite Background", cv2.GC_BGD),
        ("Probable Background", cv2.GC_PR_BGD),
        ("Definite Foreground", cv2.GC_FGD),
        ("Probable Foreground", cv2.GC_PR_FGD),
    )

    for (name, value) in values:
        valueMask = (mask == value).astype("uint8") * 255

    outputMask = np.where((mask == cv2.GC_BGD) | (mask == cv2.GC_PR_BGD),
        0, 1)

    outputMask = (outputMask * 255).astype("uint8")

    output = cv2.bitwise_and(image, image, mask=outputMask)

    return output
```

Fonte: elaborado pelos autores.

Sobre o banco de dados, a função simplesmente pega os dados enviados pelo frontend, inclusive a URL obtida pelo S3, e os insere no banco correspondente à seleção de “Perdido” ou “Encontrado”. Como é possível ver no Code Snippet 7, há uma tabela para animais (uma para perdido e outra para encontrado) e outra para as imagens (também separada entre perdido e encontrado); nele também é possível ver a composição das tabelas de animais encontrados, observando que a de imagem é ligada à de animais por uma *foreign key*. As de animais perdidos foram omitidas pois são análogas.

Code Snippet 7: Atributos de animais encontrados, análogos aos de animais perdidos, e de armazenamento das urls das imagens no banco de dados.

```
class PetFound(db.Model):
```

```

id = db.Column(db.String(80), unique=True, nullable=False, primary_key=True)
username = db.Column(db.String(80), db.ForeignKey(
    Users.username), nullable=False)
breed = db.Column(db.String(80), nullable=False, unique=False)
name = db.Column(db.String(80), nullable=True, unique=False)
hasLeash = db.Column(db.Boolean, nullable=False, unique=False)
latitude = db.Column(db.String(80), nullable=False, unique=False)
longitude = db.Column(db.String(80), nullable=False, unique=False)
color = db.Column(db.String(80), nullable=False, unique=False)
size = db.Column(db.String(80), nullable=True, unique=False)
gender = db.Column(db.String(80), nullable=False, unique=False)
hair_size = db.Column(db.String(80), nullable=True, unique=False)
castrated = db.Column(db.String(80), nullable=True, unique=False)
tail = db.Column(db.String(80), nullable=True, unique=False)
active = db.Column(db.Boolean, nullable=False)
images = db.relationship('UrlImagesFound', backref='petfound', lazy=True)

class UrlImagesFound(db.Model):
    id = db.Column(db.String(80), unique=True,
        nullable=False, primary_key=True)
    url_original = db.Column(db.String(200), nullable=False, unique=True)
    url_comparar = db.Column(db.String(200), nullable=False, unique=True)
    pet_id = db.Column(db.String(80), db.ForeignKey(PetFound.id), nullable=False)

```

Fonte: elaborado pelos autores.

### 6.1.3 Match de animais

Quando entra na tela de resultados, é feita uma chamada para a função do backend em que faz o match do animal em questão com os outros animais do banco de dados oposto a ele. Por exemplo, se o animal cadastrado é do tipo perdido, o match é feito com os animais do banco de encontrados. Este match também pode ser realizado através da página de histórico, explicada mais para frente.

Até o momento, o match é feito em duas etapas: o match por características e o match por imagens. Cada uma dessas partes tem um peso no match final.

#### 6.1.3.1 Match por características

O match por características apresenta duas funcionalidades: além de ser responsável por gerar uma porcentagem de compatibilidade entre os dados dos animais, também realiza uma pré-seleção dos animais que passarão pela etapa de match de imagens. É importante notar, porém, que o reconhecimento de imagem agrega um valor muito grande para a correspondência uma vez que animais não possuem nenhuma característica única (e.g. CPF para pessoas), e portanto existem inúmeras combinações onde uma correspondência alta entre dois animais (até mesmo de 100%) não necessariamente significa que eles são o mesmo animal. Assim, foi decidido também um peso de 30% para o match de características, deixando os outros 70% para o match proveniente da análise de imagem.

Em seguida, foram definidos pesos para cada característica de maneira que quando um cadastro novo de animal perdido ou encontrado é feito, o animal em questão tem todas suas características comparadas com as daqueles do banco oposto, como é feito no match de imagens.

A pré-seleção foi implementada de maneira a identificar diferenças extremas nas características dos animais comparados e desclassificá-los. Caso passem pela etapa de seleção por apresentarem semelhança, serão somados coeficientes de correspondência de acordo com critérios exibidos na Tabela 1.

Tabela 1: Pesos atribuídos às características.

Característica	Iguais	Parcialmente iguais	Diferentes	Observações
<b>Coleira</b>	0%	-	0%	A coleira é utilizada somente como complemento ao nome, caso o animal possua coleira o nome será considerado no match
<b>Nome</b>	5%	-	0%	O nome é considerado somente se o animal possuir coleira
<b>Endereço</b>	-	Até 1 km → 15% Até 3 km → 10% Até 5 km → 5%	De 5 km a 50 km → 0% Mais que 50 km → Descarta match	-
<b>Cor</b>	15%	5%	Descarta match	São considerados parcialmente iguais pares entre as cores bege, marrom e avermelhado e também entre quaisquer animais bicolores e tricolores

<b>Porte</b>	5%	0%	Descarta match	São considerados parcialmente iguais os pares: mini e pequeno, pequeno e médio, médio e grande, grande e GG
<b>Gênero</b>	15%	5%	Descarta match	São considerados parcialmente iguais pares entre quaisquer animais sem gênero especificado
<b>Pelo</b>	5%	-	0%	-
<b>Castração</b>	Macho → 15% Fêmeas → 0%	5%	Machos → Descarta match Outros → 0%	São considerados parcialmente iguais quaisquer pares entre animais com gênero não especificados ou sem identificação de castração
<b>Rabo</b>	10%	-		-
<b>Raça</b>	15%	5%	Descarta match	São considerados parcialmente iguais pares entre animais sem raça definida

Fonte: elaborado pelos autores.

Vale notar que a correspondência entre os animais é mais certa quando ambos os animais comparados têm todas as suas características preenchidas. Nos casos em que o usuário não conseguir identificar algumas delas, como o tamanho do pelo ou se o animal é castrado, o valor do match é diminuído, de acordo com a porcentagem definida do atributo não preenchido, mas nunca é desconsiderado do match na pré-seleção.

Também, para que o match seja mais eficaz, sempre que possível foi adicionado uma lista *dropdown* para a escolha da característica, padronizando as entradas do usuário para evitar problemas relacionados a erros de soletração ou diferenças de definição.

### 6.1.3.2 Match por imagem (Vize)

Como foi dito previamente, no primeiro semestre de desenvolvimento foi utilizado o Vize como acelerador para realizar o match por imagem, para permitir o desenvolvimento das outras funcionalidades paralelamente. A partir de Julho/2020, foi desenvolvido o algoritmo próprio para este projeto e o Vize foi abandonado; ainda assim, é relevante descrever sua utilização neste item, pois teve grande participação no desenvolvimento.

Quando era utilizado o Vize, o match por imagem era feito em duas partes principais. Como as coleções do Vize só guardam as informações de imagem, sem guardar as outras características do cadastro, primeiro é feito o match do Vize e depois é feita a correspondência entre os resultados com os seus respectivos animais no banco de dados.

A primeira parte, do match pelo Vize, é simples. Apenas se envia a URL a ser buscada e qual é a coleção em que se deve buscar para a API *visualKNN* do Vize. A *response* contém muitas informações, mas as utilizadas são apenas a URL de cada imagem de match, para fazer a correspondência com o banco de dados, e o *answer\_distances*, que dá a distância, ou porcentagem, de match de cada um dos resultados.

Code Snippet 8: Função que faz a busca por imagem no Vize.

```
def vizeSearchImage(url, type):
    if type == 'Perdido':
        headers = headersFound
    else:
        headers = headersLost

    body_json = {
        "query_record": {
            "_url": url
        },
        "fields_to_return": ["_id", "_url"]
    }
    body = json.dumps(body_json)
    insert = requests.post(
        url=vize['url'] + '/visualKNN', headers=headers, data=body)
    print('VIZE VISUAL KNN FROM ', type, ': ', insert.json())
    return insert.json()
```

Fonte: elaborado pelos autores.

Depois, vem a segunda parte: a correspondência das imagens retornadas pelo Vize com os animais cadastrados no BD. Essa parte utiliza as URLs vindas do Vize, por isso a URL é determinada como a *primary key* nas tabelas *UrllImagesFound* e *UrllImagesLost*.

Code Snippet 9: Correspondência entre *results*, que são os resultados de *match* vindos do Vize, e os animais no banco de dados.

```

data = request.get_json()
results = vizeSearchImage(data['url'], data['tipo'])
matches = []
pets_url = {}
for result in results['answer_records']:
    if data['tipo'] == 'Perdido':
        urls = UrlImagesFound.query.filter_by(url=result['_url'])
        print(urls)
        if urls.count() == 0:
            return jsonify({'response': 'Pet não encontrado', 'code': 404})
        pets = PetFound.query.filter_by(id=urls.first().pet_id)
        if pets.count() == 0:
            return jsonify({'response': 'Pet não encontrado', 'code': 404})
        match = return_pet_json(pets.first(), urls.first())
        pets_url[urls.first().url]=pets.first()
    if data['tipo'] == 'Encontrado':
        urls = UrlImagesLost.query.filter_by(url=result['_url'])
        if urls.count() == 0:
            return jsonify({'response': 'Pet não encontrado', 'code': 404})
        pets = PetLost.query.filter_by(id=urls.first().pet_id)
        if pets.count() == 0:
            return jsonify({'response': 'Pet não encontrado', 'code': 404})
        match = return_pet_json(pets.first(), urls.first())
        pets_url[urls.first().url]=pets.first()

```

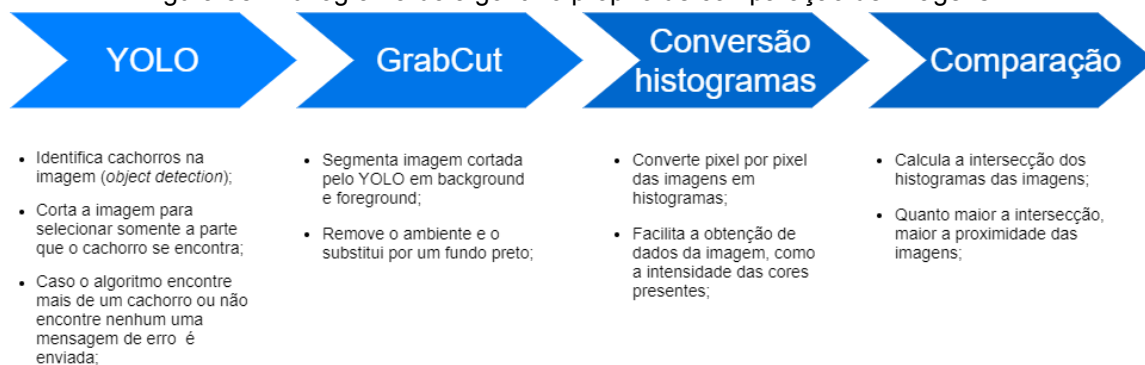
Fonte: elaborado pelos autores.

Com a finalização desse processo, é iniciada a parte de match por características, que altera a porcentagem de match, ou distância, de acordo com a comparação entre os atributos do animal cadastrado e os resultados do Vize. Essa ordem foi alterada posteriormente, permitindo uma pré-seleção dos objetos do banco, pelas suas características, para diminuir o tanto de comparações de imagem a serem feitas (que é o que custa mais tempo neste processo). Isso será mais detalhado abaixo.

### 6.1.3.3 Match por imagem (algoritmo próprio)

Durante a segunda parte do desenvolvimento, como já foi mencionado previamente, foi criado um novo algoritmo de comparação para obter valores normalizados que serão utilizados no momento de realizar matches entre animais perdidos e encontrados.

Figura 58: Fluxograma do algoritmo próprio de comparação de imagens



Fonte: elaborado pelos autores.

Como foi dito no item 6.1.3.1, primeiramente é feita uma pré-seleção do banco, considerando as características dos animais e eliminando cadastros que tenham diferenças muito extremas. Depois desta seleção feita, os animais selecionados são passados para o algoritmo de análise de imagem, que inicia a comparação entre as imagens executando a função *getColorComparisonResults*. Esta função converte as imagens para o formato de histogramas e em seguida calcula a intersecção dos histogramas, utilizando o método *compareHist()* da biblioteca OpenCV juntamente com a flag *HISTCMP\_INTERSECT*. Após a obtenção dos valores de intersecção, eles são ordenados em ordem crescente e, depois de normalizados, o algoritmo retorna à função inicial, que por fim retorna os resultados e o valor do match para o frontend.

Code Snippet 10: Função de controle do processo de comparação entre imagens.

```

def getColorComparisonResults(image_needle, images_haystack): #url, array de urls
    image = {
        "url": image_needle,
        "data": getImageFromUrl(image_needle)
    }
    imagesToCompare = []
    for image_url in images_haystack:
        imagesToCompare.append({
            "url": image_url,
            "data": getImageFromUrl(image_url)
        })
    imagesToCompareHist = convert_images_to_histogram(imagesToCompare)
    imageHist = convert_images_to_histogram([image])
    results = color_compare_images(imageHist, imagesToCompareHist)
  
```

```

if len(results) > 0:
    normalized_results = normalize_results(results)
    return normalized_results
return []

```

Fonte: elaborado pelos autores.

Code Snippet 11: Função de conversão de imagens em histogramas.

```

def convert_images_to_histogram(images):
    converted_images = []

    for image in images:
        image_color = cv2.cvtColor(image["data"], cv2.COLOR_BGR2RGB)
        hist = cv2.calcHist([image_color], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
        converted_images.append({
            "url": image["url"],
            "data": image["data"],
            "hist": cv2.normalize(hist, hist).flatten()
        })

    return converted_images

```

Fonte: elaborado pelos autores.

Code Snippet 12: Função de comparação dos histogramas.

```

def color_compare_images(selected_image_hist, hist_images):
    comparison_results = []

    for image in hist_images:
        comparison_results.append({
            "url": image["url"],
            "intersec": cv2.compareHist(selected_image_hist[0]["hist"],
            image["hist"], cv2.HISTCMP_INTERSECT)
        })

    comparison_results = [(elem["intersec"], elem["url"]) for elem in
    comparison_results]
    return comparison_results

```

Fonte: elaborado pelos autores.

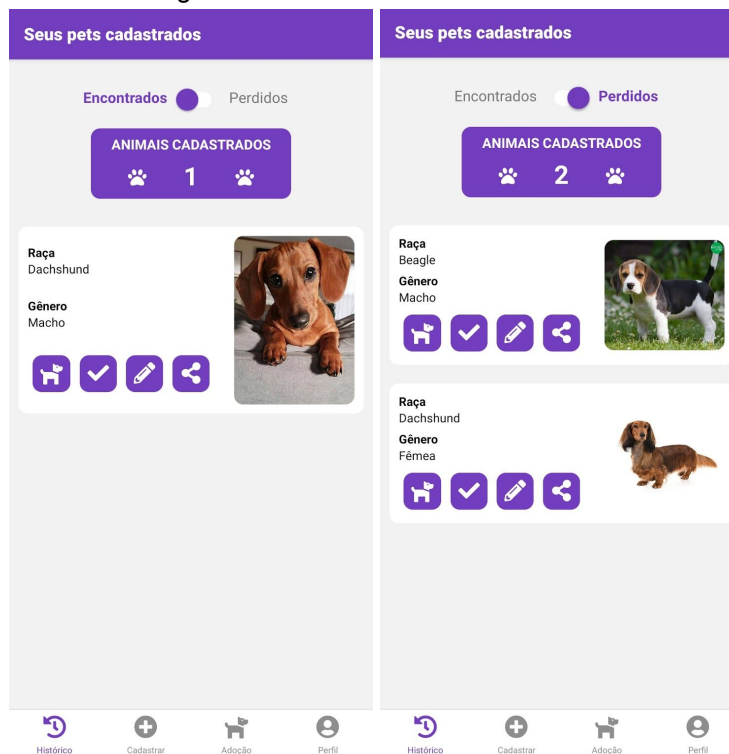
#### 6.1.4 Histórico

A terceira opção para os usuários, na tela inicial, é listar todos os animais cadastrados por ele - o que chamamos de “histórico”. Na página de histórico há a possibilidade de verificar cadastros de animais encontrados e perdidos.



Para cada animal, foram implementadas 4 funcionalidades. O usuário é capaz de visualizar os matches, inativar o cadastro, editar dados do animal e compartilhar nas redes sociais.

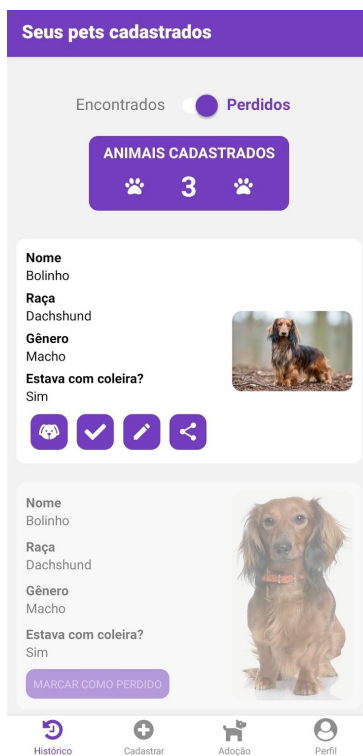
Figuras 59 e 60: Telas de histórico.



Fonte: elaborado pelos autores.

A função de inativar o cadastro foi implementada a fim de impedir que surjam novos matches para um animal que já foi encontrado pelo seu dono. Ao ser inativado o cadastro do animal é redirecionado para o fim da listagem e possui uma aparência opaca para gerar melhor identificação visual, mesmo assim o usuário ainda é capaz de reativar o cadastro ao pressionar o botão presente no *card*.

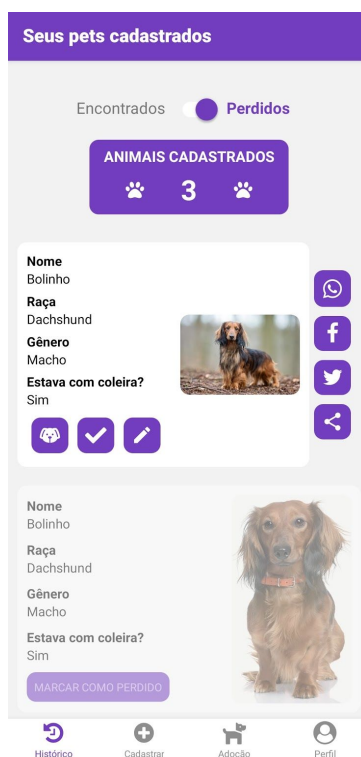
Figura 61: Telas de histórico demonstrando um cadastro inativo.



Fonte: elaborado pelos autores.

O compartilhamento em redes sociais não possui o funcionamento ideal. Ao clicar no botão, será oferecida ao usuário a opção de compartilhamento em redes sociais consideradas relevantes pelo o grupo. Ao selecionar a opção desejada, o usuário é redirecionado para o aplicativo correspondente onde pode compartilhar uma mensagem com quem desejar. Porém este compartilhamento ainda não é capaz de enviar a imagem, apenas os dados do animal; numa possível continuidade do projeto, essa funcionalidade seria desenvolvida.

Figura 62: Telas de histórico com os botão de compartilhamento com redes sociais.



Fonte: elaborado pelos autores.

Em relação ao código, a função que retorna esses animais é simples: apenas faz uma *query* pelo usuário nas tabelas de animais.

## 6.2 PET PARA ADOÇÃO

### 6.2.1 Usuário comum

O usuário comum já tem algumas de suas funções básicas: o login, a tela de cadastro de usuário e a tela de perfil do usuário, com todas as opções de interação com o aplicativo. Além destas e das funções específicas da jornada de pet perdido, temos as funções específicas da jornada de pet para adoção.

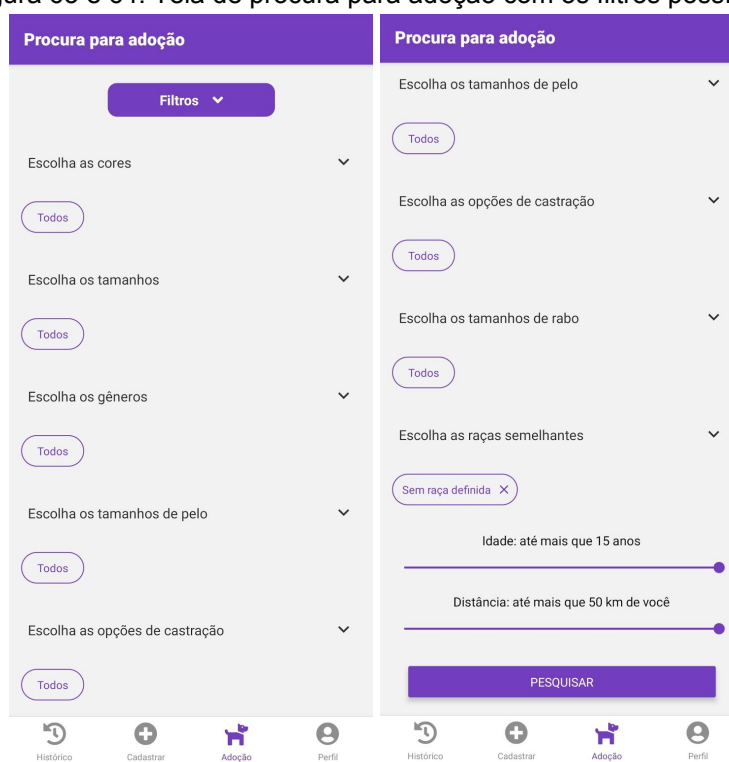
Na jornada de pet para adoção, o usuário comum teve a adição de duas novas funcionalidades: a adição da opção de busca de pet para adoção e a adição de um meio de comunicação com abrigos, buscando facilitar o processo de adoção.

### 6.2.1.1 Pesquisa por animais para adoção

A tela de pesquisa por animais para adoção permite ao usuário selecionar todas as características com as quais deseja realizar a busca. Por padrão, se nada específico for selecionado, é considerado que todas as características disponíveis são aceitas.

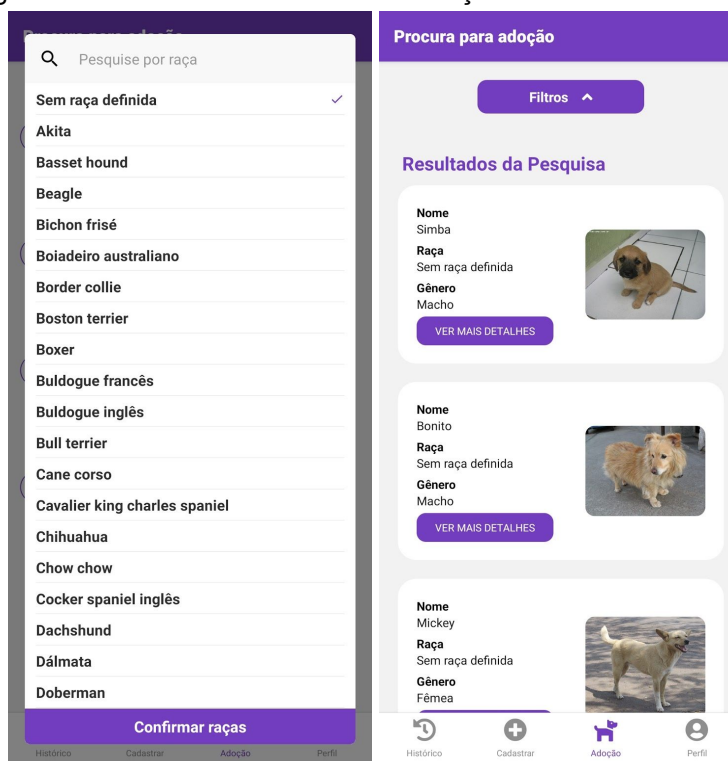
As opções disponíveis para filtragem são todas as obtidas durante o cadastro do animal, como raça, cor, porte e gênero, entre outras.

Figura 63 e 64: Tela de procura para adoção com os filtros possíveis.



Fonte: elaborado pelos autores.

Figura 65 e 66: Escolha de um filtro de raça e os resultados obtidos.



Fonte: elaborado pelos autores.

Ao receber o pedido de pesquisa, o backend realiza uma procura por tipo descrito na seção acima e depois realiza uma classificação por pontos. Assim, os animais que tiverem mais pontos serão os primeiros na visualização do usuário, sendo cada ponto atribuído por característica do animal que se encontra na pesquisa do usuário.

## 6.2.2 Abrigos

Assim como o usuário comum tem uma interface específica para ele, o abrigo também deve ter uma que se adeque às suas necessidades. A motivação por trás da criação da interface é que os abrigos tenham um meio para atualizar os animais que estejam disponíveis para adoção, e garantir que os usuários consigam identificar que está fazendo a oferta de adoção, e possibilitar a comunicação entre as duas partes.

As funções dos abrigos é análoga àquelas dos usuários, sendo que as principais são: cadastro de abrigos, tela de login, tela inicial com as funcionalidades disponíveis para o abrigo, visualização e alteração de dados do cadastro do abrigo, cadastro de animais para doação e alteração do status desses animais.

#### 6.2.2.1 Tela de login do abrigo

O padrão de abertura do aplicativo é a tela de login de usuário, que possui um botão de direcionamento para o login como abrigo. Nessa tela, o usuário pode tanto realizar o login em sua conta como cadastrar o seu abrigo, caso esse seja o primeiro acesso.

No caso do login, o backend recebe os dados de usuário e senha inseridos e verifica na base de abrigos se estão corretos. Caso contrário, retorna uma mensagem com o motivo de erro ao abrigo, assim como é feito no login de usuários comuns.

#### 6.2.2.2 Cadastro de novo abrigo

O abrigo realiza o seu cadastro escolhendo o seu nome de usuário, que deve ser único no sistema. Caso este já exista, será retornada uma mensagem de erro. Além disso, o programa pede para o abrigo adicionar os seguintes dados: nome, email, telefone, endereço, cidade e uma senha para login (a senha é adicionada em um processo de input duplo para verificação).

Após concluído o cadastro, o abrigo é adicionado ao banco de dados da classe “Shelters”, análogo à classe “Users”.

Figura 67: Tela para cadastro de abrigo

← Cadastre-se como abrigo

Username

Nome do abrigo

Email

Telefone

Endereço

Senha

Confirmar senha

SUBMIT

Fonte: elaborado pelos autores.

Code Snippet 13: Atributos de abrigo no banco de dados.

```
class Shelters(db.Model):
    username = db.Column(db.String(80), unique=True,
                        nullable=False, primary_key=True)
    shelter_name = db.Column(db.String(80), nullable=False, unique=False)
    email = db.Column(db.String(80), nullable=False, unique=False)
    password = db.Column(db.String(80), nullable=False, unique=False)
    phone = db.Column(db.Integer, nullable=False, unique=False)
    latitude = db.Column(db.String(80), nullable=True, unique=False)
    longitude = db.Column(db.String(80), nullable=True, unique=False)
    adoption_pets = db.relationship('PetAdoption', backref='shelter', lazy=True)
```

Fonte: elaborado pelos autores.

### 6.2.2.3 Tela inicial do abrigo

Após cadastrar e realizar o login, o abrigo tem acesso à sua tela inicial, onde pode escolher entre as seguintes opções:

- Listar suas informações cadastradas (botão “Perfil”)
  - Aqui o abrigo também poderá alterar essas informações.
- Cadastrar pets para adoção (botão “Cadastrar”)

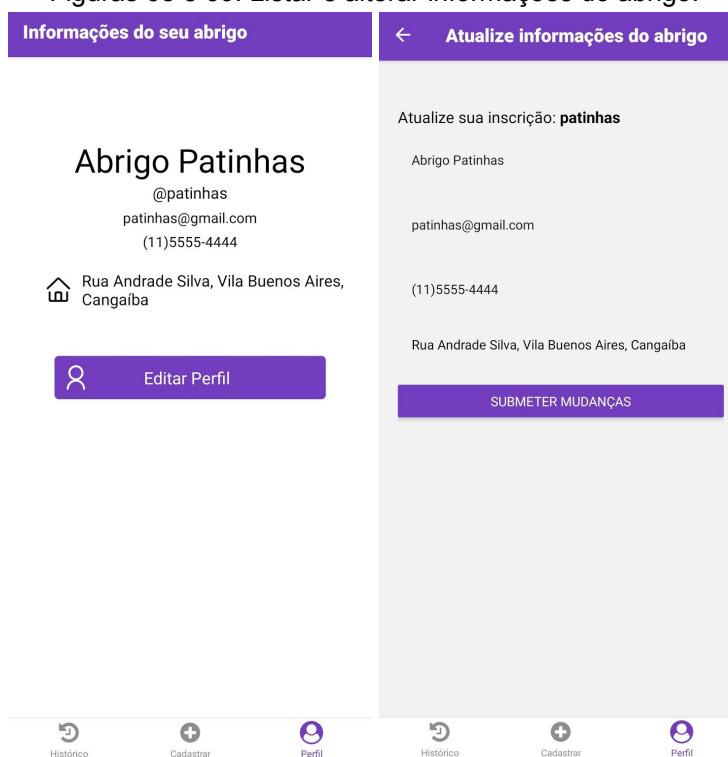
- Listar os pets cadastrados em sua base (botão “Histórico”)
  - Aqui o abrigo também pode alterar o cadastro de seus pets;

#### 6.2.2.4 Visualização e atualização do perfil

A tela de listagem de informações do abrigo apresenta todas as informações que foram colocadas pelo abrigo na hora do cadastro e permite a alteração das mesmas caso tenha ocorrido algum erro ou caso elas tenham mudado.

Para mudar as informações, o usuário deve simplesmente tocar no retângulo com sua descrição. As regras para aceitação das informações continuam as mesmas que aquelas apresentadas no cadastro do abrigo.

Figuras 68 e 69: Listar e alterar informações do abrigo.



Fonte: elaborado pelos autores.

#### 6.2.2.5 Cadastro de animal para adoção



As telas de cadastro para adoção são quase idênticas às telas dos animais perdidos e encontrados, as duas únicas diferenças sendo que os de adoção não possuem tipo (Encontrado/Perdido), nem estado da coleira. Também permite na atualização informar caso o animal tenha sido adotado.

Para que isso seja possível, foram criadas as classes *PetAdoption* e *UrllImagesAdoption* no banco de dados para adequar o cadastro dos animais às funções pré-existentes.

Figura 70: Atualização de cadastro de animal para adoção.

Fonte: elaborado pelos autores.

Code Snippet 14: Atributos de animal para adoção no banco de dados e de armazenamento das urls das imagens no banco de dados.

```
class PetAdoption(db.Model):
    id = db.Column(db.String(80), unique=True, nullable=False, primary_key=True)
    username = db.Column(db.String(80), db.ForeignKey(
        Shelters.username), nullable=False)
    breed = db.Column(db.String(80), nullable=False, unique=False)
    name = db.Column(db.String(80), nullable=False, unique=False)
    age = db.Column(db.Integer, nullable=False, unique=False)
    latitude = db.Column(db.String(80), nullable=False, unique=False)
    longitude = db.Column(db.String(80), nullable=False, unique=False)
    color = db.Column(db.String(80), nullable=False, unique=False)
```

```

size = db.Column(db.String(80), nullable=False, unique=False)
gender = db.Column(db.String(80), nullable=False, unique=False)
hair_size = db.Column(db.String(80), nullable=False, unique=False)
castrated = db.Column(db.String(80), nullable=False, unique=False)
tail = db.Column(db.String(80), nullable=False, unique=False)
adopted = db.Column(db.Boolean, nullable=False, unique=False)
images = db.relationship('UrlImagesAdoption', backref='petadoption', lazy=True)

class UrlImagesAdoption(db.Model):
    id = db.Column(db.String(80), unique=True, nullable=False, primary_key=True)
    url_original = db.Column(db.String(200), nullable=False, unique=True)
    pet_id = db.Column(db.String(80), db.ForeignKey(PetAdoption.id), nullable=False)

```

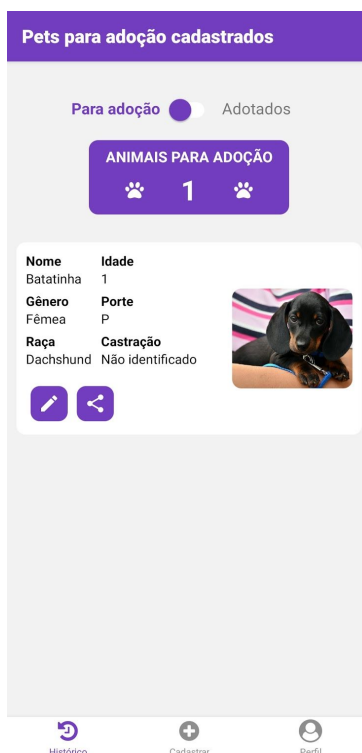
Fonte: elaborado pelos autores.

### 6.2.2.6 Histórico

Ao selecionar essa opção, o abrigo consegue ver os animais que ele cadastrou no seu sistema e o respectivo status dele (se já foi ou não adotado).

Caso deseje, o usuário pode também alterar as informações dos animais cadastrados por ele ao clicar no botão de edição existente no card do animal.

Figura 71: Tela de listagem dos pets cadastrados para adoção.



Fonte: elaborado pelos autores.

### 6.3 SEGURANÇA

A segurança de um sistema digital compartilhado por múltiplos usuários é um tópico de extrema importância para garantir que os dados dos usuários estejam disponíveis apenas de acordo com as especificações que eles concordaram e se sentem confortáveis.

Como já versado no capítulo dos aspectos conceituais, algumas das medidas de segurança foram implementadas pensando na complacência com a Lei Geral de Proteção de Dados, que está em vigor desde Agosto de 2018. Outras medidas, no entanto, foram implementadas para garantir que o sistema como um todo esteja protegido a possíveis ataques externos.

Após começar a implementação, percebeu-se que simplesmente rodar o backend diretamente na URL e garantir o HTTPS através do flask não seria o melhor modo de operação. Assim, decidiu-se implementar em conjunto com o uso do servidor, o uso de um reverse proxy - o nginx - garantindo assim que todos os acessos não sejam feitos diretamente ao nosso backend, eles teriam que passar por uma camada extra que realiza o chaveamento entre os pedidos. Em conjunto com este novo sistema, é possível então conectar o certificado digital diretamente no servidor, sendo este uso independente da aplicação que roda em Python - aumentando sua resiliência - e garantindo que qualquer acesso ao servidor seja feito através de um canal seguro. Acredita-se que esta arquitetura seja mais segura em comparação com a inicialmente desenhada e foi portanto desenvolvida.

Por fim, foram feitas implementações para que, no caso de mau funcionamento, fosse identificável se o erro foi causado pelo sistema ou por agentes externos.

Todas as implementações realizadas para garantir a segurança do sistema serão descritas abaixo, assim como suas características e funcionalidades:

- Habilitado somente porta 80 (HTTP) do servidor e 443 (HTTPS) - evitando que acessos indevidos sejam realizados.

- Certificado digital emitido pelo certbot para um domínio adquirido (www.petfinder.digital) - garantindo que a conexão realizada pelo usuário é segura e da fonte correta.
- Utilização do nginx como reverse proxy, garantindo o funcionamento dos caminhos como o servidor só possui a porta 80 e 443 aberta.
- Implementação de log com timestamp e detalhes do pedido realizado
- Armazenamento de informações sensíveis com hash no banco de dados

### 6.3.1 Habilitação de portas no servidor e segurança

Foram selecionadas somente algumas portas para restringir o tráfego que chega no servidor e assim garantir uma maior segurança.

Estas foram:

- Aceitar qualquer conexão para a porta 80 - HTTP
- Aceitar qualquer conexão para a porta 443 - HTTPS
- Somente os IPs dos membros do grupo na porta 22 - SSH - impedindo assim que qualquer pessoa possa acessar diretamente o servidor

Figura 72: Portas habilitadas para acesso no servidor

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
HTTP	TCP	80	0.0.0.0/0
HTTP	TCP	80	:::0
SSH	TCP	22	189.100.213.109/32
HTTPS	TCP	443	0.0.0.0/0
HTTPS	TCP	443	:::0

Fonte: elaborado pelos autores.

Juntamente a isto, foi criada uma chave de segurança .ppk necessária para logar no servidor, e somente os membros do grupo possuem ela. Dificultando portanto a invasão por terceiros da máquina.

### 6.3.2 Certbot

Para poder utilizar a conexão HTTPS no servidor, foi utilizado um certificado emitido pela certbot - uma empresa que emite certificados digitais gratuitamente. Para poder ligar o certificado em uma URL, foi necessário a compra de um domínio [www.petfinder.digital](http://www.petfinder.digital) e foi ligado nele o certificado.

### 6.3.3 Nginx

O Nginx funciona como um reverse proxy. Assim, os chamados do aplicativo para o backend não irão se conectar diretamente na porta em que roda o programa em python. Dentro do servidor, é montado um arquivo que faz a decodificação dos pedidos que chegam e para onde devem ser enviados dentro da própria máquina. Como é o servidor que está encarregado de garantir a conexão HTTPS, o backend roda na URL <http://localhost:5000> dentro do servidor, porém os pedidos que chegam de fora virão na forma [www.petfinder.digital](http://www.petfinder.digital).

Figura 73: Conversão no arquivo do Nginx.

```
location /login {  
    proxy_pass http://localhost:5000/login;  
    proxy_http_version 1.1;  
    proxy_set_header Upgrade $http_upgrade;  
    proxy_set_header Connection 'upgrade';  
    proxy_set_header Host $host;  
    proxy_cache_bypass $http_upgrade;  
}
```

Fonte: elaborado pelos autores.

Assim, o caminho do pedido que chega vem pela entrada [www.petfinder.digital/login](http://www.petfinder.digital/login) e, dentro da máquina, fica em <http://localhost:5000/login>.

### 6.3.4 Log

Foi implementado um log no backend para permitir rastrear os pedidos realizados pelos usuários, tanto em seu conteúdo quanto no momento em que foram realizados.

Ele é escrito em um arquivo .txt e possui vários níveis de alerta: Debug, Info, Error, Warning. Permitindo assim que as mensagens sejam classificadas por nível de criticidade.

Figura 74: Exemplo de log.

```
2020-08-16 12:54:59,767 - INFO - Starting log of day 20200816
2020-08-16 12:54:59,960 - INFO - * Restarting with stat
2020-08-16 12:55:02,433 - INFO - Starting log of day 20200816
2020-08-16 12:55:02,450 - WARNING - * Debugger is active!
2020-08-16 12:55:02,590 - INFO - * Debugger PIN: 571-109-280
2020-08-16 12:55:02,687 - INFO - * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Fonte: elaborado pelos autores.

Nenhum erro é retornado diretamente ao usuário, porém será logado com o timestamp juntamente com o username para inspeção posterior se necessária.

### 6.3.5 Armazenamento de senhas com hash

Para aumentar a segurança existente no projeto e proteger as senhas dos usuários, foi decidido pelo armazenamento utilizando o “hasheamento” das mesmas. Ao utilizar a função hash em uma senha, garante-se que o valor da mesma nunca poderá ser recuperado do banco de dados, evitando assim, o sucesso de possíveis ataques ao banco.

Para isso, a biblioteca bcrypt do Python foi utilizada no cadastro e no login dos usuários, da seguinte maneira:

Code Snippet 15: Uso de Hash no cadastro dos Usuários.

```
hashed=bcrypt.hashpw(user_data['password'].encode('utf-8'), bcrypt.gensalt())
```

Fonte: elaborado pelos autores.

Code Snippet 16: Uso de Hash no login dos usuários.

```
if bcrypt.checkpw(login_data['password'].encode('utf-8'), user.password):
    returnValue = return_user_json(user)
```

Fonte: elaborado pelos autores.

## 6.4 GEOLOCALIZAÇÃO

No início, o campo de endereço dos cadastros de animais era apenas um texto, o que dificultava muito sua utilização como parâmetro de match, pois não tem um padrão de preenchimento ou conhecimento da distância. Portanto, foi desenvolvida uma tela em que se exibe um mapa, para poder selecionar o lugar específico intencionado no cadastro.

Quando o mapa é clicado, um marcador vermelho é exibido onde houve o clique e suas coordenadas são armazenadas. Essas coordenadas também são enviadas por um request para a opção *reverse* da API de Geocoding da LocationIQ, que converte as coordenadas em um endereço legível, que é então apresentado abaixo do mapa, para deixar claro para o usuário qual será o endereço salvo naquele momento.

Figura 75: Mapa com marcador e o endereço convertido abaixo.

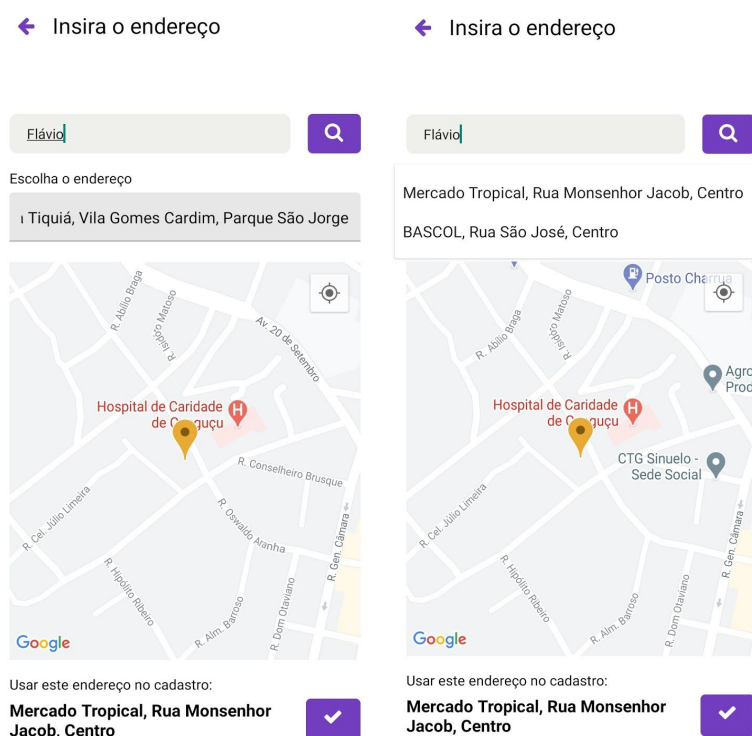


Fonte: elaborado pelos autores.

Além disso, nessa página também é possível fazer uma pesquisa por endereço, que retorna todos os resultados possíveis em relação às palavras chaves buscadas. Essa busca também é feita por um request para a API de LocationIQ, mas para a opção de *search*.

Os resultados são exibidos em um *dropdown* e marcados com um marcador amarelo no mapa. Quando outro resultado é selecionado no *dropdown*, o mapa é movido para a nova localização com uma animação e o endereço exibido abaixo do mapa, para deixar claro para o usuário qual será o endereço salvo se o processo for concluído naquele momento.

Figuras 76 e 77: Exibição dos resultados da busca da palavra “Flávio” e da marcação do primeiro item da lista no mapa.



Fonte: elaborado pelos autores.

Quando esse processo é finalizado, clicando no botão de *check*, as coordenadas selecionadas são guardadas e, se o processo de cadastro for finalizado, elas são salvas no banco de acordo com o cadastro. Esses valores posteriormente são utilizados no match de animais perdidos, dependendo da distância e na procura por animal para adoção, por filtro de distância.



## 6.5 SPRINTS REALIZADAS

Todo esse desenvolvimento foi feito numa divisão de 6 sprints, com várias *issues* (as tarefas, na nomenclatura do Jira) por sprint. No início de cada sprint, os integrantes conversavam entre si para determinar quais eram as próximas issues a serem desenvolvidas e para distribuir entre eles quem faria quais. As sprints eram completadas normalmente quando aquelas issues escolhidas eram finalizadas, sem uma restrição rígida de tempo.

A seguir, são mostrados os relatórios de cada uma das 6 sprints, para poder analisar o processo de desenvolvimento durante o ano.

Figura 78: Relatório da primeira sprint, a *Pinscher*, que durou de 27/Abril a 18/Maio.

TIME \* Issue added to sprint after start time

Status Report

Completed Issues [View in Issue Navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (-)
PET-1	Autenticação de usuário	Story	↑ Medium	DONE	-
PET-8	Criar arquivos básicos do app	Story	↑ Medium	DONE	-
PET-11 *	[Cadastro Pet Perdido] Tela de cadastro de animal perdido	Story	↑ Medium	DONE	-
PET-13 *	[Cadastro Pet Perdido] Tela de resultados (match a partir do cadastro)	Story	↑ Medium	DONE	-

Issues Not Completed [View in Issue Navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (-)
PET-10 *	Estrutura do App	Story	↑ Medium	BACKLOG	-
PET-14 *	[Cadastro Pet Perdido] Match por Imagem - integração Vize	Story	↑ Medium	SELECTED FOR DEVELOPMENT	-
PET-15 *	Integração AWS S3	Story	↑ Medium	IN PROGRESS	-

Fonte: elaborado pelos autores, utilizando o Jira (<https://www.atlassian.com/software/jira>)

Figura 79: Relatório da segunda sprint, a *Golden Retriever*, que durou de 18/Maio a 8/Junho.

Status Report \* Issue added to sprint after start time

Completed Issues [View in Issue Navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (-)
PET-10	Estrutura do App	Story	Medium	DONE	-
PET-14	[Cadastro Pet Perdido] Match por Imagem - integração Vize	Story	Medium	DONE	-
PET-15	Integração AWS S3	Story	Medium	DONE	-
PET-16 *	[Cadastro Pet Perdido] Match por características	Story	Medium	DONE	-

Issues Removed From Sprint [View in Issue Navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (-)
PET-17 *	Organização do Código	Story	Medium	BACKLOG	-

Fonte: elaborado pelos autores, utilizando o Jira (<https://www.atlassian.com/software/jira>)

Figura 80: Relatório da terceira sprint, a *Corgi*, que durou de 8/Junho a 14/Julho.

Status Report \* Issue added to sprint after start time

Completed Issues [View in Issue Navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (-)
PET-17	Organização do Código	Story	Medium	DONE	-
PET-33	Definir peso e quais características no match por características	Story	Medium	DONE	-
PET-34	AWS S3 - Permitir setar ambiente automaticamente	Story	Medium	DONE	-
PET-44 *	Ajustes no css	Story	Medium	DONE	-
PET-45 *	Ajustes no fluxo de cadastro	Story	Medium	DONE	-

Issues Removed From Sprint [View in Issue Navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (-)
PET-39 *	Criar apk da v1	Story	Medium	IN PROGRESS	-

Fonte: elaborado pelos autores, utilizando o Jira (<https://www.atlassian.com/software/jira>)

Figura 81: Relatório da quarta sprint, a *Husky*, que durou de 14/Julho a 31/Agosto.

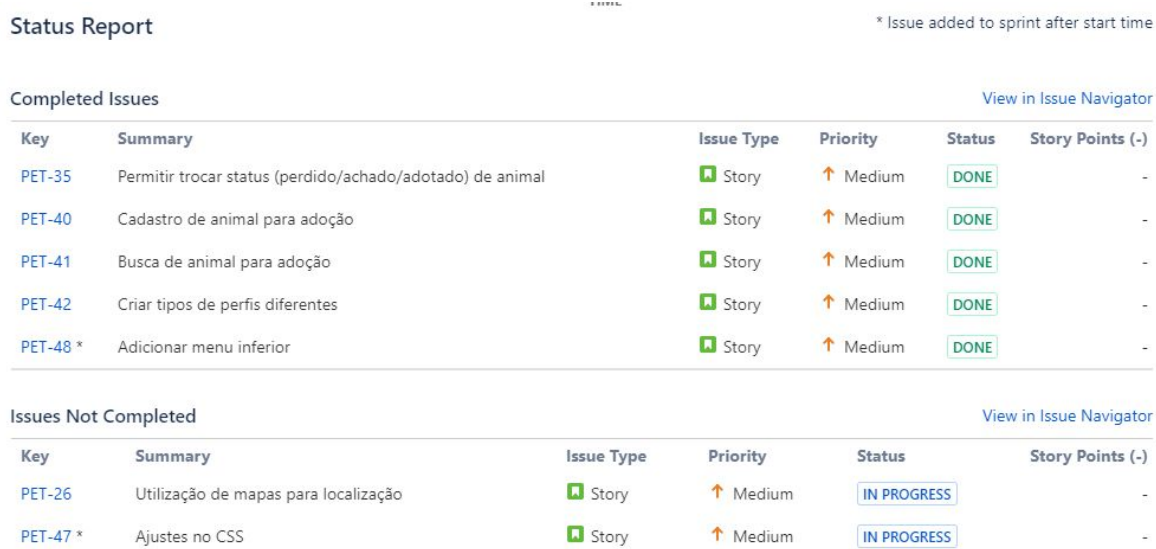
Status Report

Completed Issues [View in Issue Navigator](#)

Key	Summary	Issue Type	Priority	Status	Story Points (-)
PET-43	Algoritmo de reconhecimento de imagem	Story	Medium	DONE	-
PET-46	Segurança	Story	Medium	DONE	-

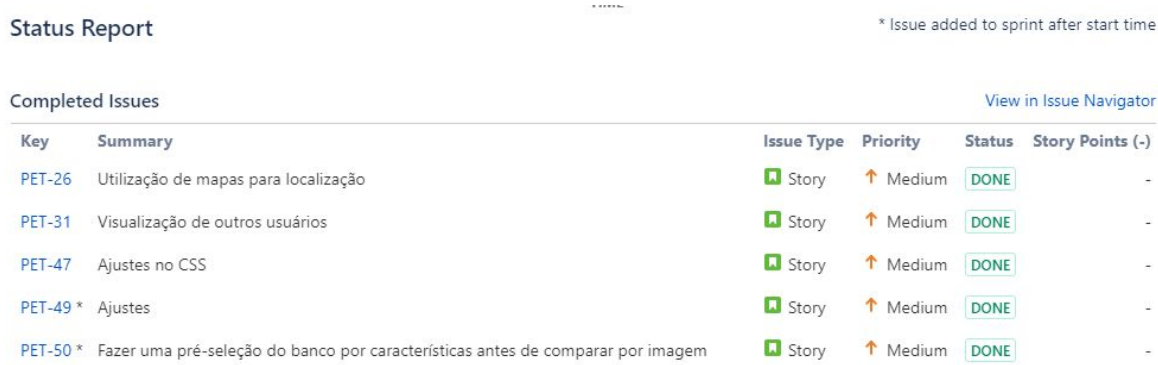
Fonte: elaborado pelos autores, utilizando o Jira (<https://www.atlassian.com/software/jira>)

Figura 82: Relatório da quinta sprint, a *Vira-lata caramelo*, que durou de 31/Agosto a 24/Outubro.



Fonte: elaborado pelos autores, utilizando o Jira (<https://www.atlassian.com/software/jira>)

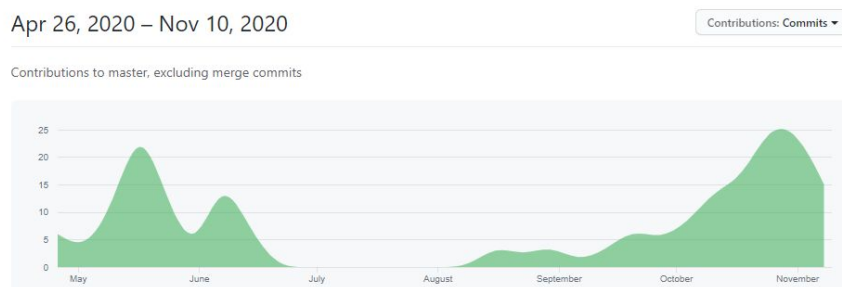
Figura 83: Relatório da sexta e última sprint, a *Salsicha*, que durou de 24/Outubro a 09/Novembro.



Fonte: elaborado pelos autores, utilizando o Jira (<https://www.atlassian.com/software/jira>)

Além dos relatórios das sprints no Jira, é relevante mostrar o gráfico de frequência de código do repositório deste projeto, que mostra a quantidade de commits feitos por mês.

Figura 84: Gráfico de frequência de código do repositório principal do projeto.



Fonte: elaborado pelos autores, utilizando o GitHub (<https://github.com/>)

É possível observar a falta de commits durante o mês de Julho e Agosto. Isso se deu pois durante esses meses o desenvolvimento estava focado no algoritmo de análise de imagem e do estudo da parte de segurança. Para a implementação do algoritmo, foi feito um outro repositório e depois o código finalizado foi integrado ao repositório principal. Por isso, abaixo está o gráfico de frequência de código do outro repositório, específico do algoritmo de análise de imagem, e é possível observar como ele preenche o vácuo dos meses de Julho e Agosto do repositório principal.

Figura 85: Gráfico de frequência de código do repositório do algoritmo de análise de imagem.  
Jul 19, 2020 – Nov 10, 2020

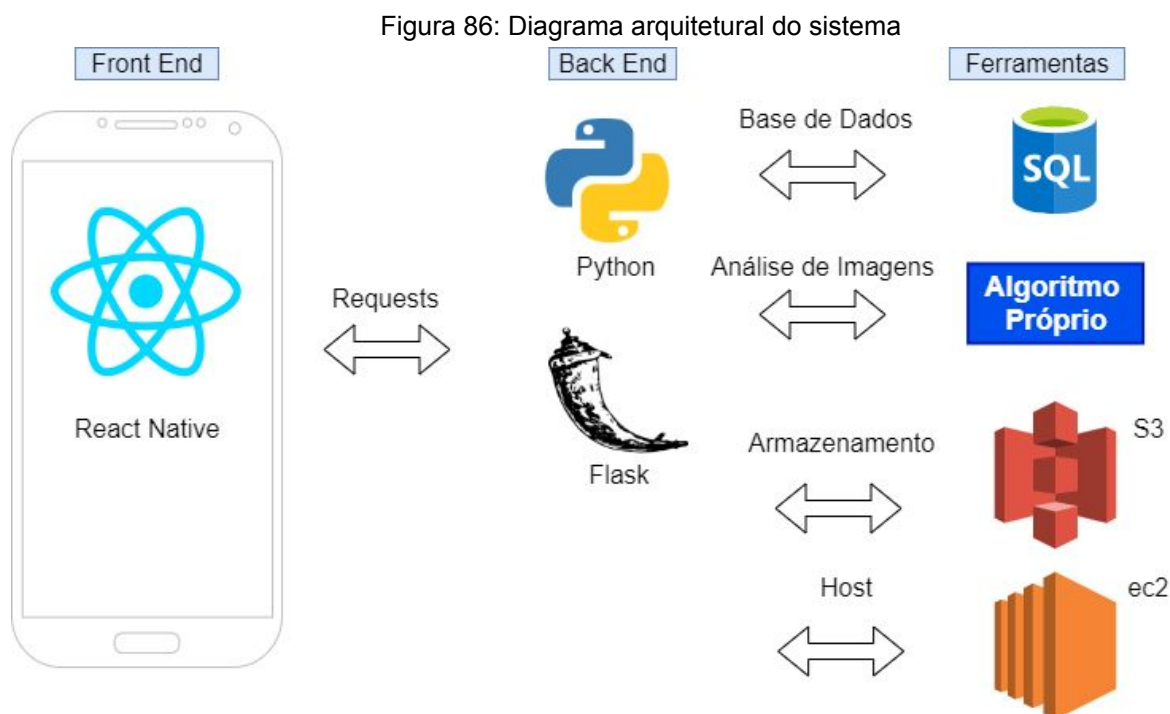
Contributions to master, excluding merge commits



Fonte: elaborado pelos autores, utilizando o GitHub (<https://github.com/>)

## 6.6 DIAGRAMA ARQUITETURAL DO SISTEMA

Abaixo encontra-se o diagrama arquitetural do sistema, separado em três grupos: frontend, backend e ferramentas.



Fonte: elaborado pelos autores.

Essa arquitetura nos permitiu implementar os requisitos não funcionais discutidos anteriormente como necessários para o projeto. Abaixo é listado a relação entre os requisitos e os componentes da arquitetura que os implementam.

- **Segurança:** Implementado nas interfaces do Python, S3 e EC2;
- **Usabilidade:** Implementado na interface do frontend;
- **Disponibilidade:** implementado na interface da EC2;
- **Acessos múltiplos:** implementado nas interfaces da EC2 e S3;
- **Tempo real:** implementado na interface de Python e do algoritmo próprio;
- **Portabilidade:** projeto desenvolvido apenas para usuários Android;
- **Log estatístico:** não implementado.

## 7 TESTES E AVALIAÇÃO

Durante e após o desenvolvimento deste projeto, foram feitos diversos testes para determinar se as decisões feitas eram realmente as mais adequadas. Neste tópico, será descrito o planejamento e procedimentos realizados para teste de avaliação do componente de processamento e comparação de imagens do sistema.

### 7.1 ALGORITMO DE ANÁLISE DE IMAGEM

Como as principais funcionalidades da vertical de animal perdido criadas neste projeto estão diretamente ligadas ao algoritmo de análise de imagem, é necessária uma inspeção completa para conferir se o algoritmo utilizado está adequado para as utilizações previstas. Assim, fizemos testes de cada etapa da análise das imagens de cachorros, para determinar o tempo médio demorado a cada processo e seu total, além da qualidade dos resultados.

#### 7.1.1 Preparação das imagens

Inicialmente, foram feitos testes das duas etapas de preparação da imagem para comparação: a etapa de *Object Detection* e a de *Foreground Extraction*.

Para os testes, foram selecionadas 50 imagens de cachorro, tanto fotos pessoais dos integrantes do grupo quanto fotos alheias compartilhadas em redes sociais. As fotos escolhidas passaram por apenas um critério visual, antes do processo de análise pelo algoritmo: elas devem conter apenas um cachorro (dado que o *Object Detection* retorna erro se tiver mais de um animal na foto). Dentre as imagens, foram selecionadas algumas que já se esperava que falhassem no processo do algoritmo, por diversos motivos. Elas serão explicadas especialmente em cada item.

A cada teste, foram anotados dois fatores: o tempo de processamento daquela etapa e uma nota para seu resultado. As notas foram divididas em 1, 2 e 3, sendo 3 a melhor nota e 1 a pior nota. O critério para as notas de cada etapa foi explicado em seus respectivos tópicos.

Além disso, tentamos utilizar fotos mais realistas, que se assemelham mais com fotos tiradas por donos comuns de cachorros, ao invés de imagens perfeccionistas de *photoshoots*, revistas e similares, para aproximar o máximo possível da utilização real dos futuros usuários da plataforma.

#### 7.1.1.1 Object Detection

Para o caso dessa etapa, foram dadas apenas nota 1 e 3, sem ter uma nota intermediária. A classificação foi simples e direta: se houve um recorte correto do cachorro, a nota dada é 3 - independente de ter cortado fora alguma parte do animal, como a orelha ou uma pata, como é possível acontecer. Se não houve uma seleção correta do cachorro na imagem (dando erro de “Não foi reconhecido nenhum animal na imagem”), a nota atribuída foi 1.

Foram passadas por essa etapa as 50 imagens selecionadas. 44 delas foram corretamente detectadas e recortadas. Uma das que receberam nota 1 foi exibida abaixo, por ser do cachorro de um dos integrantes do grupo, mas as outras foram omitidas por terem sido retiradas de redes sociais e não terem o consentimento dos autores para sua publicação neste trabalho.

Figuras 87: Uma das imagens não tratada corretamente pelo YOLO Object Detection. É possível ver a posição incomum do cachorro, o que justifica o erro do algoritmo.



Fonte: foto tirada por um dos autores.

Tabela 2: Resultados de tempo da etapa de Object Detection.

<b>Média</b>	<b>Mediana</b>	<b>Desvio padrão</b>	<b>Tempo mínimo</b>	<b>Tempo máximo</b>
0.69438 seg	0.66351 seg	0.12597 seg	0.588 seg	1.09162 seg

Fonte: elaborada pelos autores.

A performance obtida, exemplificada pelos tempos mostrados na tabela acima, são bem satisfatórios, tendo menos de um segundo de processamento por imagem.

Apesar de 12% das imagens terem falhado nesta etapa, duas delas já se esperavam erro (a exibida acima, com o animal em posição não costumeira, e outra em que o cachorro estava num campo cheio de ovelhas, o que pode confundir o algoritmo por exibir outros animais). Assim, consideramos que houve 8% de erros inesperados. Apesar do valor não ser tão baixo, este erro não é totalmente impeditivo para o usuário: quando um erro da análise da imagem acontece, deve ser exibida uma mensagem para o usuário explicando o que houve e pedindo para inserir uma outra imagem, até mesmo podendo dar dicas de quais imagens são mais bem analisadas pelo algoritmo da plataforma.

#### 7.1.1.2 Foreground Extraction

Como esta etapa é seguinte à de *Object Detection*, apenas as 44 imagens com bons resultados na etapa anterior foram repassadas para esta etapa.

Nesta etapa, foram dadas as 3 notas possíveis, e seu critério foi o seguinte:

- **Nota 3:** extração ótima, quase sem erros.
- **Nota 2:** extração boa, com alguns erros perceptíveis.
- **Nota 1:** extração inutilizável.



Figuras 88 e 89: Imagem recortada pelo *Object Detection* e a com extração feita, respectivamente. Este resultado recebeu nota 3.



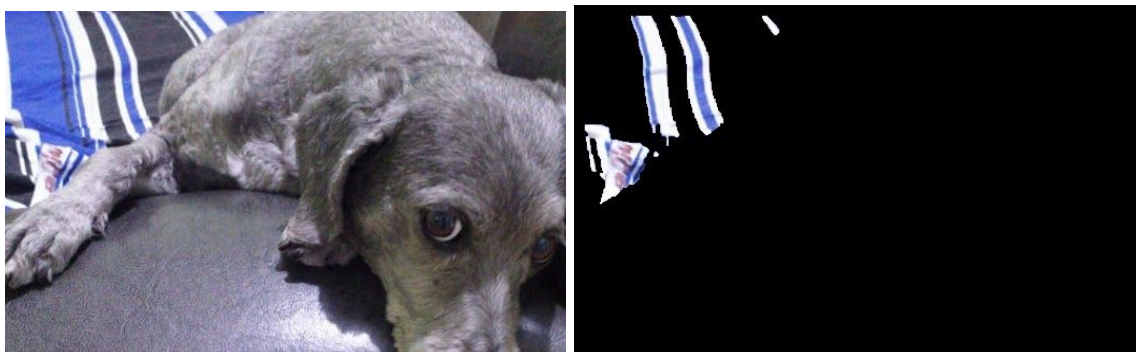
Fonte: foto original tirada por um dos autores.

Figuras 90 e 91: Imagem recortada pelo *Object Detection* e a com extração feita, respectivamente. Este resultado recebeu nota 2, por ter retirado parte das feições do animal, mas por ainda estar majoritariamente adequado.



Fonte: foto original tirada por um dos autores.

Figuras 92 e 93: Imagem recortada pelo *Object Detection* e a com extração feita, respectivamente. É possível ver como a extração foi feita do conteúdo errado. Este resultado recebeu nota 1.



Fonte: foto original tirada por um dos autores.

Tabela 3: Resultados de tempo da etapa de Foreground Extraction.

<b>Média</b>	<b>Mediana</b>	<b>Desvio padrão</b>	<b>Tempo mínimo</b>	<b>Tempo máximo</b>
93.61578 seg	60.9013 seg	95.06327 seg	2.81103 seg	386.7594 seg

Fonte: elaborada pelos autores.

Tabela 4: Resultados de notas da etapa de Foreground Extraction.

<b>Quantidade de notas 1</b>	<b>Quantidade de notas 2</b>	<b>Quantidade de notas 3</b>
4	14	26

Fonte: elaborada pelos autores.

Em relação às notas, pode-se observar que apenas 4 imagens, das 44 obtidas, foram consideradas inutilizáveis, o que equivale a 9%. É uma margem de erro considerável, mas, como foi dito no item anterior, este erro não é completamente impeditivo - é possível alertar o usuário e pedir o recadastro com outra imagem, provavelmente mais adequada para o processamento.

O tempo gasto nesta etapa é muito superior ao da etapa anterior, chegando a uma mediana de um minuto. Este tempo está longe de atender o requisito de tempo de cadastro de animal, que é de 15 segundos, porém não é proibitório. Para manter o requisito funcional atendido, é possível que durante o cadastro apenas faça o *Object Detection* e salve o resultado no S3, e então já indique o final do processo para o usuário. Para finalizar o processo (com o *Foreground Extraction*), deve-se fazer conferências periódicas (diárias,

provavelmente) em que se processa automaticamente todas as imagens que foram cadastradas desde a última conferência. Caso haja um erro no processo, é possível avisar o usuário posteriormente e pedir o recadastramento com outra foto. Porém, essa alternativa não é ideal, pois deve-se considerar que o usuário não tenha outra imagem se for avisado só posteriormente que o cadastro não foi completado. Portanto, foi decidido manter o cadastro em tempo real, mesmo que demorasse mais tempo do que foi idealizado.

#### 7.1.1.3 Resultados

A partir dos testes feitos de cada etapa do processamento das imagens, foi possível tirar diversas conclusões do que afeta o tempo e o resultado de cada etapa. Estas informações foram registradas para possivelmente entrarem numa lista de dicas para o usuário de qual imagem colocar no cadastro do seu animal a fim de encontrar o match correto.

Abaixo, são listadas as principais conclusões obtidas, acompanhadas com exemplos para melhor visualização.

- A cor do fundo da imagem e a do cachorro influencia o *Foreground Extraction*. Caso as cores sejam similares, a extração não é satisfatória.
- Animais que são majoritariamente de uma cor e tem apenas uma parte do corpo de outra cor costumam ter esta parte do corpo recortada fora pelo *Foreground Extraction*.

Figuras 94 e 95: Imagem resultado do Object Detection e do Foreground Segmentation, respectivamente. Na segunda imagem, é possível ver que parte das costas do cachorro foi recortada, por ser preta, além de boa parte do chão ter sido mantido, pela similaridade à cor majoritária do animal.



Fonte: foto tirada por um dos autores.

- A interpretação do que é “fundo” e do que é “frente” pelo GrabCut é influenciada pela cor. Assim, se houver uma cor vibrante no fundo e o cachorro ter cor esmaecida, é possível que haja um corte errado, como nas imagens 75 e 76.
- Imagens com muitos outros elementos na imagem além do animal podem falhar no *Object Detection*. O ideal é ter uma imagem em que o animal está centralizado, grande e sem muitos itens de tamanho semelhante ao redor (como brinquedos, panelas, bolas, etc).
- É mais fácil detectar o animal quando ele está de frente ou de lado para a câmera. Fotos de quando o cachorro está em posições mais variadas tendem a não ser bem sucedidas no *Object Detection*, como a da figura 70.

### 7.1.2 Comparação das imagens

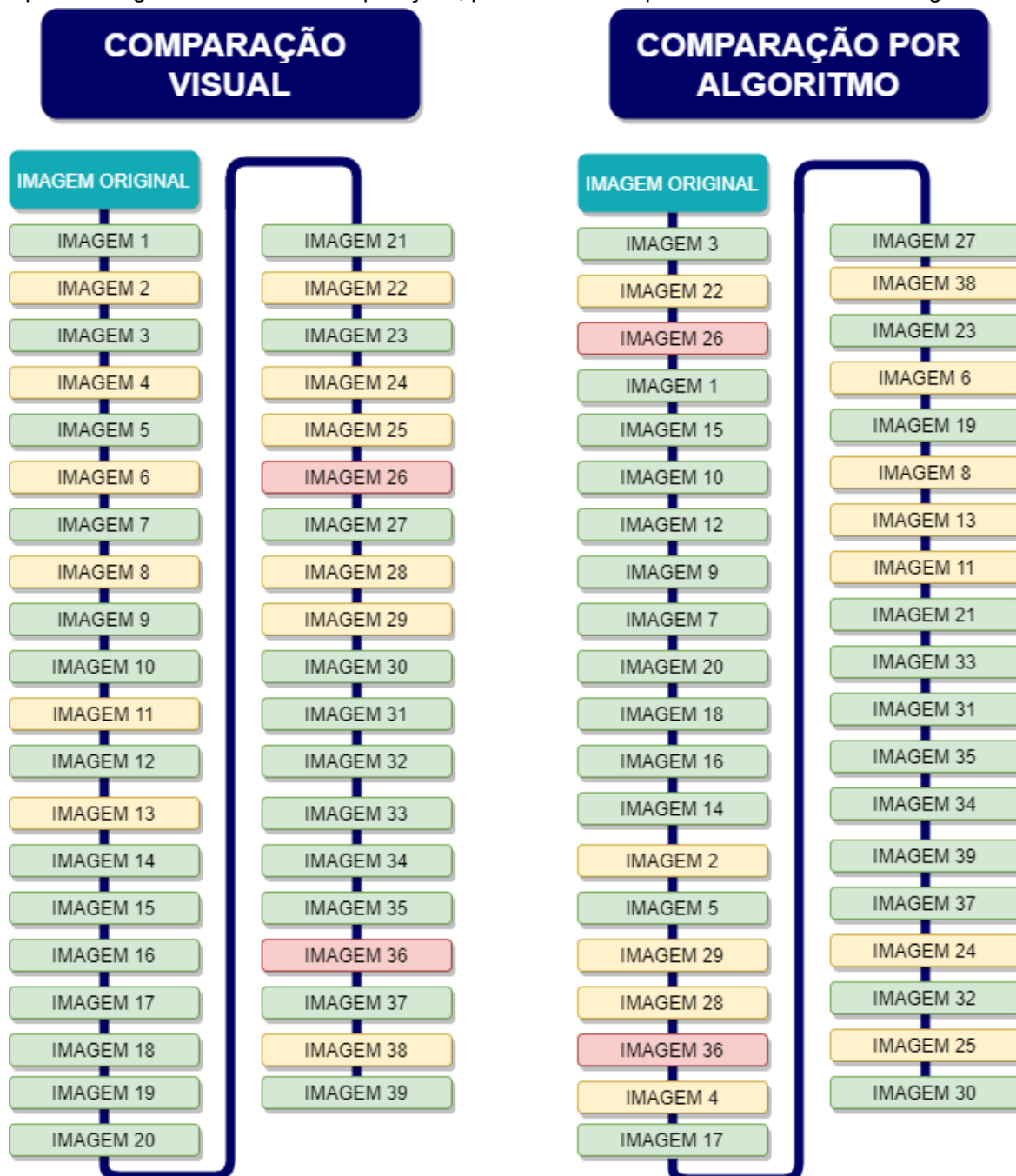
Depois de as imagens terem sido preparadas durante o cadastro, com as etapas de *Object Detection* e *Foreground Extraction*, resta realizar a comparação entre os resultados.

Para o teste desta etapa, foram utilizadas as 40 imagens resultantes dos testes anteriores (considerando as 44 passadas para a etapa de *Foreground Extraction* e retirando as 4 que foram consideradas inutilizáveis). Foi escolhida a primeira imagem para ser comparada com todas, portanto houve 40 comparações finais. Neste teste, não foram dadas notas individuais para cada comparação feita, por não ser tão visivelmente mensurável quanto os testes de preparação das imagens - porém, foi feita uma comparação visual entre as 40 imagens e elas foram ordenadas, do mais parecido para o mais diferente, e esta ordenação foi comparada com a ordenação do match pelo algoritmo.

Para determinar a qualidade do resultado, abaixo está uma comparação entre a ordenação feita pelo algoritmo e uma feita visualmente pelos integrantes do grupo. Pelo fato de não termos a permissão de exibir as imagens, principalmente as obtidas em redes sociais, elas foram referenciadas por números. Para facilitar a comparação das ordenações apresentadas, houve a codificação por cores:

- **Verde:** imagens que ficaram a até 10 posições de diferença entre as duas ordenações;
- **Amarelo:** imagens que ficaram de 11 a 20 posições de diferença entre as duas ordenações;
- **Vermelho:** imagens que tiveram mais de 20 posições de diferença entre as duas ordenações.

Figura 96: Diagrama das ordenações feitas pelo algoritmo de comparação por cor e visualmente pelos integrantes. A comparação com a imagem original foi omitida, por obviamente ter ficado em primeiro lugar em ambas as comparações, portanto houve apenas 39 itens em cada diagrama.



Fonte: elaborado pelos autores.

Pelo diagrama, é possível ver que os resultados deste algoritmo não são ideais - há até algumas posições extremamente divergentes, como a da imagem 26 na posição 2 do algoritmo. Ainda assim, dos 39 itens observados, apenas 2 (5%) tiveram erro completo (cor vermelha) e 12 (30%) foram resultados

regulares. Dado que o requisito de precisão do algoritmo de comparação é de 70%, o resultado desse algoritmo foi considerado satisfatório.

Além dos resultados, também foi contado o tempo de cada comparação, o que inclui a conversão da imagem para histograma e a comparação por intersecção dos histogramas.

Tabela 5: Resultados de tempo da comparação por cores. O tempo mínimo foi anotado como 0 segundos pois foi pequeno demais para ser calculado; o tempo total é das 40 comparações feitas.

<b>Média</b>	<b>Mediana</b>	<b>Desvio padrão</b>	<b>Tempo mínimo</b>	<b>Tempo máximo</b>	<b>Tempo total</b>
0.000974 seg	0.0009995 seg	0.000897 seg	0 seg	0.003996 seg	0.036 seg

Fonte: elaborada pelos autores.

É possível ver como o tempo de cada comparação é pequeno, algo que é imprescindível dado que o match é feito com grande parte do banco de dados (exceto pelos cadastros eliminados na etapa de match por características) e, portanto, é esperada a comparação de muitas imagens para um único match.

Considerando o requisito de tempo máximo de comparação em um match, que é de 1 minuto como *soft limit*, e utilizando o valor médio de 0.000974 segundos para cada comparação de imagem, é possível fazer mais de 60 mil comparações de imagem em um único match sem ultrapassar o limite de tempo, desconsiderando o tempo de conexão entre frontend/backend, o tempo para obter os animais do banco e o tempo de comparação de característica.

Para ter mais específico o tanto de imagens possíveis para comparação dentro do *soft limit*, foi feito um teste destes tempos que não entram na funcionalidade de comparação por imagem. Para um banco de 50 animais, o tempo total do match por característica (somando o tempo de obtenção dos objetos do banco e da comparação das características) foi de 0.156 segundos, o que dá uma média de 0.00313 segundos por animal no banco, o que não se equivale ao tanto de animais comparados por imagem, por causa da pré-seleção feita pelas características. Assim, considerando 3 segundos o tempo de conexão entre frontend e backend, e que 80% dos objetos do banco são considerados no

match por imagem, é possível fazer o cálculo de quantos animais no banco pode-se comparar num match completo em 1 minuto:

$$2 + 0.00313 \times y + 0.8 \times 0.000974 \times y = 60 \text{ segundos} \rightarrow y = 14836$$

Portanto, conclui-se que é possível ter quase 15 mil objetos em cada tabela (de animal perdido e animal encontrado) e ainda assim não passar nem do *soft limit* do tempo de comparação.



## 8 CONSIDERAÇÕES FINAIS

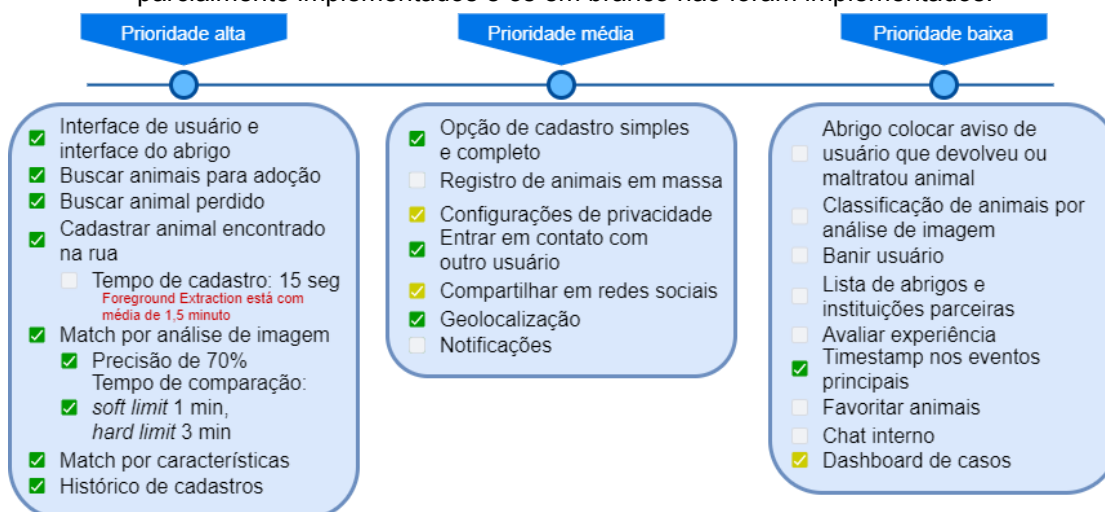
A seção final deste documento contará com a recapitulação do que foi desenvolvido, apresentando o balanço final do que foi ou não realizado, ressaltando sempre o que foi de desenvolvimento próprio do grupo e o que foi importado de terceiros, e as dificuldades enfrentadas pelos autores no desenvolvimento.

Aqui também será analisado o resultado final do projeto, frente aos problemas descritos no começo deste documento e avaliando a real efetividade da solução proposta contra os pontos que eram considerados limitantes para eventuais usuários.

### 8.1 CONCLUSÕES DO PROJETO DE FORMATURA

No item 5.3.1, foram discutidos os requisitos funcionais a serem implementados durante este projeto. Estes requisitos foram divididos em prioridades alta, média e baixa, para organizar quais deveriam ter preferência de implementação. Agora, é possível fazer uma análise de quantos desses requisitos foram efetivamente desenvolvidos, quais foram ignorados e por quais motivos.

Figura 97: Análise da completude dos requisitos funcionais, divididos em prioridades. Os requisitos marcados em verde foram inteiramente implementados, os em amarelo foram parcialmente implementados e os em branco não foram implementados.



Fonte: elaborado pelos autores.

As funcionalidades feitas foram descritas em seus respectivos tópicos, portanto serão omitidas neste item. Em relação aos itens não feitos, pode-se ver como foram quase todos de prioridade baixa, o que não afeta o resultado do trabalho. Os itens de prioridade média não implementados também eram considerados aditivos e não funcionalidades necessárias, portanto não obtiveram prioridade dentre os outros de prioridade média e, assim, permaneceram não implementados.

O projeto apresentou um custo total de aproximadamente 20 dólares estadunidenses aos membros do grupo, devido a emissões de certificado e à utilização dos serviços da S3 e EC2. Se o projeto fosse oficialmente liberado ao público, é provável que a opção de máquina utilizada na EC2 e o tamanho de storage do S3 demandassem aumento, o que elevaria o preço total para manter o projeto ativo.

Além do balanço do trabalho, apresentado acima, é necessário relembrar a grande mudança que o mundo passou durante o ano de 2020. Devido à situação incomum e imprevista de pandemia e a consequente quarentena, iniciada no final de Março/2020 em São Paulo, alguns dos aspectos planejados inicialmente para o projeto tiveram que ser adaptados.

Primeiramente, o contato direto e possível envolvimento no projeto de abrigos de animais foi severamente limitado. Por isso, foi dado um foco maior nas funcionalidades ligadas ao fluxo de pet perdido, em que as percepções de valor poderiam ser baseadas muito mais em opiniões próprias e pesquisa aberta, sem necessariamente o envolvimento de uma terceira parte. Conceitualmente não se perdeu muito, já que esse fluxo inclui a análise de imagens e utilização de banco de dados. O fluxo de animais para adoção também foi implementado, porém sem maior direcionamento por parte de entidades interessadas, o que diminui um pouco a sua proposta de valor, além de que algumas funcionalidades foram adiadas, com a expectativa de final da quarentena, e portanto não foram feitas, como o cadastro em massa de animais de abrigos (que necessitava de

informações internas dos abrigos, sobre como eles mantêm o controle dos animais que abrigam).

Outros efeitos da pandemia foram devidamente minimizados pelos autores e por seu orientador, buscando trabalhar sempre em sistemas de compartilhamento pelo cloud e marcar reuniões via plataformas de videoconferência online.

## 8.2 CONTRIBUIÇÕES

Todo o desenvolvimento do projeto foi de autoria do grupo, sendo que este utilizou alguns artifícios já existentes de terceiros para auxílio na execução das ideias propostas pelos autores, como APIs (Interface de Programação de Aplicações) e SDKs (Kit de Desenvolvimento de Software) públicas, disponibilizadas por empresas ou instituições que visam o auxílio na programação de aplicações.

Além disso, diversas bibliotecas das linguagens escolhidas (Python e React Native) foram utilizadas no desenvolvimento do projeto, como no armazenamento hashado de informações ou no desenvolvimento do algoritmo próprio de análise de imagens, visando otimizar os resultados e tornar o sistema mais simples e confiável.

## 8.3 PERSPECTIVAS DE CONTINUIDADE

Para análise do resultado obtido do projeto, deve-se julgar o sistema final contra os aspectos que motivaram o engajamento no tema, e avaliar se as exigências que os mesmos possuem foram cumpridas e tornaram essa aplicação uma alternativa satisfatória às presentes no mercado.

Primeiramente, ao analisar a vertical de animais para adoção, um dos principais problemas identificados foi a superlotação de ONGs e falta de capacidade para acolhimento de animais em situação vulnerável. Esse problema é combatido em duas instâncias: a primeira é a criação de uma interface de

adoção simples e de fácil usabilidade para usuários, facilitando o processo de adoção e consequentemente disponibilizando capacidade nas ONGs; a segunda é a possibilidade de conexão das ONGs com animais perdidos e de rua, por meio da interface de busca de animal perdido.

Em seguida, analisando a vertical de animais perdidos, vimos que os maiores problemas eram a dificuldade de identificar animais em uma plataforma consolidada e a falta de uma base de usuários de tamanho considerável. Esses problemas são atacados na fase de projeto tanto com a criação de uma interface simples e amigável, parecida com a maioria dos aplicativos que as pessoas já tem em seus celulares, como com a utilização do algoritmo de reconhecimento de imagem, tornando mais conveniente a inserção de novos animais e tornando o processo de encontrar o animal perdido mais confiável e preciso.

Porém, a questão de ter uma base de usuários consolidada não para na elaboração da ferramenta. Existe trabalho de divulgação e de consolidação que deve ocorrer depois do aplicativo ser disponibilizado para o público.

Para dar continuidade ao projeto, será necessário a transferência da responsabilidade para entidades com interesse nos assuntos abordados, com o devido auxílio dos autores. Alguns tópicos levantados que poderiam ser implementados para melhor funcionamento do sistema são:

- Possibilidade dos abrigos de registrar animais em grande quantidade
- Configurações de privacidade para usuários (escolher o que quer ou não deixar como visualização pública)
- Implementação de um chat interno ao aplicativo
- Banir usuários da plataforma ou colocar *red flag* quando o usuário já devolveu ou maltratou animais
- Notificações para usuários
- Dashboard estatístico da plataforma
- Lista com ONGs e instituições parceiras (se existirem)
- Avaliação de experiência no app e com abrigos/usuários específicos
- Criação de um ambiente web para rodar a aplicação.

## REFERÊNCIAS

International Organization for Standardization, 2009. Information technology — Open distributed processing — Reference model: Architecture — Part 3

Velasco, C., 2020. Brasil Tem Mais De 170 Mil Animais Abandonados Sob Cuidado De Ongs, Aponta Instituto. [online] G1. Available at: <<https://g1.globo.com/sp/sao-paulo/noticia/2019/08/18/brasil-tem-mais-de-170-mil-animais-abandonados-sob-cuidado-de-ongs-aponta-instituto.ghtml>> [Accessed 26 March 2020].

Sqlalchemy.org. 2020. Sqlalchemy - The Database Toolkit For Python. [online] Available at: <<https://www.sqlalchemy.org/>> [Accessed 1 May 2020].

Flask.palletsprojects.com. 2020. Welcome To Flask — Flask Documentation (1.1.X). [online] Available at: <<https://flask.palletsprojects.com/en/1.1.x/>> [Accessed 1 May 2020].

Android Developers. 2020. Android Developers. [online] Available at: <<https://developer.android.com/>> [Accessed 5 May 2020].

Postman.com. 2020. [online] Available at: <<https://www.postman.com/>> [Accessed 15 May 2020].

Marketplace.visualstudio.com. 2020. *Git Graph - Visual Studio Marketplace*. [online] Available at: <<https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph>> [Accessed 14 June 2020].

Atlassian. 2020. *Jira | Issue & Project Tracking Software | Atlassian*. [online] Available at: <<https://www.atlassian.com/software/jira>> [Accessed 14 June 2020].

Atlassian. 2020. *Why Git | Atlassian Git Tutorial*. [online] Available at: <<https://www.atlassian.com/git/tutorials/why-git>> [Accessed 14 June 2020].

Scrum.org. 2020. *What Is Scrum?*. [online] Available at: <<https://www.scrum.org/resources/what-is-scrum>> [Accessed 14 June 2020].

GitHub. 2020. *React-Native-Community/React-Native-Geolocation*. [online] Available at: <<https://github.com/react-native-community/react-native-geolocation>> [Accessed 15 June 2020].

Lee Brock Camargo Advogados, n.d. *O QUE MUDA COM A NOVA LEI DE DADOS PESSOAIS*. [online] LGPD Brasil. Available at: <<https://www.lgpdbrasil.com.br/o-que-muda-com-a-lei/>> [Accessed 18 June 2020].

Planalto.gov.br. 2020. *L13709*. [online] Available at: <[http://www.planalto.gov.br/ccivil\\_03/\\_ato2015-2018/2018/lei/L13709.htm](http://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/L13709.htm)> [Accessed 18 June 2020].

Dados, S., 2020. *O Que Muda Com A LGPD — LGPD - Lei Geral De Proteção De Dados Pessoais*. [online] Serpro.gov.br. Available at: <<https://www.serpro.gov.br/lgpd/menu/a-lgpd/o-que-muda-com-a-lgpd>> [Accessed 18 June 2020].

Idec.org.br. 2020. *Home*. [online] Available at: <<https://idec.org.br/dadospessoais>> [Accessed 18 June 2020].

Anchor. 2020. *Dadocracia - Episódio 09 By Dadocracia • A Podcast On Anchor*. [online] Available at: <<https://anchor.fm/dadocracia/episodes/Dadocracia---Episdio-09-edu3cc>> [Accessed 18 June 2020].

Idec.org.br. 2020. *Lei De Proteção De Dados Traz Desafios A Empresas, Cidadãos E Governo*. [online] Available at: <<https://idec.org.br/idec-na-imprensa/lei-de-protecao-de-dados-traz-desafios-empr-esas-cidadaos-e-governo>> [Accessed 18 June 2020].

Observatório - Por Data Privacy. 2020. *Memória - Observatório - Por Data Privacy*. [online] Available at: <<https://observatorioprivacidade.com.br/memorias/>> [Accessed 18 June 2020].

Docs.python.org. 2020. *25.3. Unittest — Unit Testing Framework — Python 2.7.18 Documentation*. [online] Available at: <<https://docs.python.org/2/library/unittest.html>> [Accessed 20 June 2020].

International Organization for Standardization, n.d. *International Organization For Standardization*. [online] ISO. Available at: <<https://www.iso.org/home.html>> [Accessed 21 June 2020].

Institutopetbrasil.com. n.d. *Quem Somos – Instituto Pet Brasil*. [online] Available at: <<http://institutopetbrasil.com/quem-somos/>> [Accessed 21 June 2020].

Docs.aws.amazon.com. 2020. *Conceitos Básicos Do Amazon Simple Storage Service - Amazon Simple Storage Service*. [online] Available at: <[https://docs.aws.amazon.com/pt\\_br/AmazonS3/latest/gsg/GetStartedWithS3.htm](https://docs.aws.amazon.com/pt_br/AmazonS3/latest/gsg/GetStartedWithS3.htm)> [Accessed 21 June 2020].

WHEELAN, C. Estatística: O que é, Para que Serve, Como Funciona. Rio de Janeiro: Zahar, 2016.

Medium. 2020. A Dog Detector And Breed Classifier. [online] Available at: <<https://towardsdatascience.com/a-dog-detector-and-breed-classifier-4feb99e1f852>> [Accessed 13 July 2020].

Imageai.readthedocs.io. 2020. Custom Training: Prediction — Imageai 2.1.5 Documentation. [online] Available at: <<https://imageai.readthedocs.io/en/latest/custom/index.html>> [Accessed 13 July 2020].

Medium. 2020. Image Recognition With 10 Lines Of Code. [online] Available at: <<https://medium.com/@guymodscientist/image-prediction-with-10-lines-of-code-3266f4039c7a>> [Accessed 13 July 2020].

Brownlee, J., 2020. How To Classify Photos Of Dogs And Cats (With 97% Accuracy). [online] Machine Learning Mastery. Available at: <<https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/>> [Accessed 16 July 2020].

Rosebrock, A., 2020. How-To: Python Compare Two Images - Pyimagesearch. [online] PyImageSearch. Available at: <<https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/>> [Accessed 17 July 2020].

Brownlee, J., 2020. How To Perform Object Detection With Yolov3 In Keras. [online] Machine Learning Mastery. Available at: <<https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>> [Accessed 20 July 2020].



Colab.research.google.com. 2020. Google Colaboratory. [online] Available at:  
<[https://colab.research.google.com/github/google/eng-edu/blob/master/ml/pc/exercises/image\\_classification\\_part1.ipynb](https://colab.research.google.com/github/google/eng-edu/blob/master/ml/pc/exercises/image_classification_part1.ipynb)> [Accessed 20 July 2020].

Medium. 2020. Image Classification Vs Object Detection Vs Image Segmentation. [online] Available at:  
<<https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81>> [Accessed 20 July 2020].

Rosebrock, A., 2020. Opencv Grabcut: Foreground Segmentation And Extraction - Pyimagesearch. [online] PyImageSearch. Available at:  
<<https://www.pyimagesearch.com/2020/07/27/opencv-grabcut-foreground-segmentation-and-extraction/>> [Accessed 27 July 2020].

Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2020. You Only Look Once: Unified, Real-Time Object Detection. [online] arXiv.org. Available at:  
<<https://arxiv.org/abs/1506.02640>> [Accessed 27 July 2020].

PyPI. 2020. Opencv-Python. [online] Available at:  
<<https://pypi.org/project/opencv-python/>> [Accessed 27 July 2020].

Rosebrock, A., 2020. YOLO Object Detection With Opencv - Pyimagesearch. [online] PyImageSearch. Available at:  
<<https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/>> [Accessed 27 July 2020].

Rosebrock, A., 2020. Opencv Tutorial: A Guide To Learn Opencv - Pyimagesearch. [online] PyImageSearch. Available at:

<<https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/>> [Accessed 01 August 2020].

TutorialKart. 2020. Opencv Python Save Image - Cv2.Imwrite() - Example. [online] Available at: <<https://www.tutorialkart.com/opencv/python/opencv-python-save-image-example/>> [Accessed 01 August 2020].

Gist. 2020. Text: Imagenet 1000 Class Idx To Human Readable Labels (Fox, E., & Guestrin, C. (N.D.). Coursera Machine Learning Specialization.). [online] Available at: <<https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a>> [Accessed 03 August 2020].

ŚWIEŻEWS, J. 2020. What is YOLO Object Detection?. [online] Available at: <<https://appsilon.com/object-detection-yolo-algorithm/>> [Accessed 07 August 2020].

Svi.nl. n. d. Image histograms. [online] Available at: <<https://svi.nl/ImageHistogram> > [Accessed 07 August 2020].

Findingrover.com. 2020. Finding Rover - Pet Facial Recognition. [online] Available at: <<https://findingrover.com/>> [Accessed 29 August 2020].

Siwalu Software. 2020. Siwalu - AI-Based Image Recognition To Identify Animals. [online] Available at: <<https://siwalusoftware.com/>> [Accessed 29 August 2020].

Amazon Web Services, Inc. 2020. Amazon Web Services (AWS) – Serviços De Computação Em Nuvem. [online] Available at: <<https://aws.amazon.com/pt/>> [Accessed 12 October November 2020].

Python.org. 2020. Welcome To Python.Org. [online] Available at: <<https://www.python.org/>> [Accessed 12 October 2020].

Google Cloud. 2020. Apis De Geolocalização | Google Maps Platform | Google Cloud. [online] Available at: <<https://developers.google.com/maps?hl=pt-br>> [Accessed 25 October 2020].

Locationiq.com. 2020. Locationiq - Free & Fast Geocoding, Reverse Geocoding And Maps Service. [online] Available at: <<https://locationiq.com/>> [Accessed 25 October 2020].

StatCounter Global Stats. 2020. Mobile Operating System Market Share Brazil | Statcounter Global Stats. [online] Available at: <<https://gs.statcounter.com/os-market-share/mobile/brazil>> [Accessed 10 November 2020].

Docs.opencv.org. 2020. Opencv: Histogram Comparison. [online] Available at: <[https://docs.opencv.org/3.4/d8/dc8/tutorial\\_histogram\\_comparison.html](https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html)> [Accessed 17 November 2020].

Rosebrock, A., 2020. How-To: 3 Ways To Compare Histograms Using Opencv And Python - Pyimagesearch. [online] PyImageSearch. Available at: <<https://www.pyimagesearch.com/2014/07/14/3-ways-compare-histograms-using-opencv-python/>> [Accessed 17 November 2020].

Reactnative.dev. 2020. React Native. [online] Available at: <<https://reactnative.dev/>> [Accessed 20 November 2020].

Flask-sqlalchemy.palletsprojects.com. 2020. Flask-Sqlalchemy — Flask-Sqlalchemy Documentation (2.X). [online] Available at:

<<https://flask-sqlalchemy.palletsprojects.com/en/2.x/>> [Accessed 20 November 2020].

Terra. 2020. Saiba O Que É Preciso Para Adotar Um Animal De Estimação. [online] Available at: <<https://www.terra.com.br/vida-e-estilo/mulher/saiba-o-que-e-preciso-para-adotar-um-animal-de-estimacao,07b96ee9f9e27310VgnCLD100000bbcceb0aRCRD.html>> [Accessed 29 November 2020].

## APÊNDICE A - Termo de Consentimento

### TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

O(A) Sr.(a) está sendo convidado(a) como voluntário(a) a participar deste questionário/entrevista que tem como objetivo entender o contexto de pessoas que perdem seus animais de estimação e daquelas que possivelmente encontrarão estes animais.

O motivo que nos leva a estudar é buscar relatos de pessoas que já passaram por essa situação e entender quão frequentemente isso acontece.

Para este estudo adotaremos os seguintes procedimentos: um questionário, em que a coleta de dados será realizada por meio de formulários online, e entrevistas, via ligações ou chat, visando uma compreensão mais profunda do que os entrevistados pensam sobre o tema.

O motivo deste convite é que o(a) Sr.(a) se enquadra nos seguintes critérios de inclusão: [possui interesse por animais de estimação e é um usuário ativo de tecnologias como internet e *smartphones*. (versão para questionário)] / [já teve experiência encontrando um animal perdido na rua ou perdeu seu animal.]

O(A) Sr.(a) deverá deixar de participar da pesquisa nos casos em que forem observados os seguintes critérios de exclusão: menores de idade.

Para participar deste estudo o(a) Sr.(a) não terá nenhum custo, nem receberá qualquer vantagem financeira, mas será garantido, se necessário, o ressarcimento de eventuais custos com ligações telefônicas.

O(A) Sr.(a) será esclarecido(a) sobre o estudo em qualquer aspecto que desejar e estará livre para participar ou recusar-se a participar, retirando seu consentimento ou interrompendo sua participação a qualquer momento. A sua participação é voluntária e a recusa em participar não acarretará qualquer penalidade ou modificação na forma em que é atendido pelo pesquisador.

O pesquisador irá tratar a sua identidade com padrões profissionais de sigilo e privacidade, sendo que em caso de obtenção de fotografias ou gravações de voz os materiais ficarão sob a propriedade do pesquisador responsável. Seu nome ou o material que indique sua participação não será liberado sem a sua permissão. O(A) Sr.(a) não será identificado(a) em nenhuma publicação que possa resultar deste estudo.

Caso hajam danos decorrentes dos riscos desta pesquisa, o pesquisador assumirá a responsabilidade pelo ressarcimento e pela indenização.

- Li e concordo com os termos acima descritos
- Concordo que os materiais e as informações obtidas relacionadas à minha pessoa poderão ser utilizados em atividades de natureza acadêmico-científica, desde que assegurada a preservação de minha identidade.