CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS

Recredenciado pela Portaria Ministerial nº 1.162, de 13/10/16, D.O.U. nº 198, de 14/10/2016
AELBRA EDUCAÇÃO SUPERIOR - GRADUAÇÃO E PÓS-GRADUAÇÃO S.A.

Stéfani Carol Almeida de Arruda Gonçalves

PERDI MEU PET: uma plataforma para busca de animais domésticos perdidos com análise de características.

Palmas – TO
2022

Stéfani Carol Almeida de Arruda Gonçalves

PERDI MEU PET: Uma plataforma para busca de animais domésticos perdidos com análise de características.

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Projeto Tecnológico de Engenharia de Software do curso de bacharel em Engenharia de Software pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. Esp. Fábio Castro Araújo

Palmas – TO

2022

Stéfani Carol Almeida de Arruda Gonçalves

PERDI MEU PET: Uma plataforma para busca de animais domésticos perdidos com análise de características.

Projeto de Pesquisa elaborado e apresentado como requisito parcial para aprovação na disciplina de Projeto Tecnológico de Engenharia de Software do curso de bacharel em Engenharia de Software pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).

Orientador: Prof. Esp. Fábio Castro Araújo

Aprovado em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. Esp. Fábio Castro de Araújo

Orientador

Centro Universitário Luterano de Palmas – CEULP

_____

Prof. Ma. Madianita Bogo Marioti

Centro Universitário Luterano de Palmas - CEULP

_____

Prof. Dr. Jackson Gomes de Souza

Centro Universitário Luterano de Palmas - CEULP

Palmas – TO

2022

RESUMO

GONÇALVES, Stéfani Carol Almeida de Arruda. **PERDI MEU PET: Uma plataforma para busca de animais domésticos perdidos com análise de características.** 2022. 48 f. Projeto Tecnológico de Engenharia de Software – Curso de Engenharia de Software, Centro Universitário Luterano de Palmas, Palmas/TO, 2022.

Animais de estimação estão cada vez mais presentes nos domicílios brasileiros e por terem múltiplas funções na sociedade humana, muitas vezes são considerados uma forma de trazer conforto, companheirismo e aumentar a autoestima das pessoas. Por meio de uma pesquisa feita pela ASPCA (2012) em Nova York com o intuito de entender quantos animais de estimação perdidos ainda continuavam perdidos e quantos acabaram voltando para casa. Foi constatado que apenas 74% dos gatos e 93% dos cães perdidos foram recuperados. No Brasil, normalmente quando um animal de estimação foge seus tutores imprimem e espalham nas ruas cartazes com a foto do animal, compartilham fotos em redes sociais, aplicativos de mensagens instantâneas ou grupos de proteção aos animais, sendo na maioria dos casos difícil encontrar um local específico para estruturar essas informações, deixando a divulgação dos dados do animal perdida e desordenada dentre tantas outras, impossibilitando o processo de identificação. Junto com essa análise veio a proposta deste trabalho, que é desenvolver um aplicativo móvel para centralizar informações acerca de animais perdidos e encontrados, para assim ampliar as possibilidades de tutores encontrarem informações na qual poderão levar a reaver seus animais perdidos. Além disso, foi proposta também a implementação de um *ranking* de animais semelhantes dentro da plataforma usando o cálculo da distância euclidiana para estimar as semelhanças de acordo com suas características.

Palavras-chave: Animais de estimação, distância euclidiana.

LISTA DE FIGURAS

Figura 1 - Recursos nativos do Flutter 16

Figura 2 - Código e o resultado da compilação Flutter 16

Figura 3 - Hierarquia dos *widgets* 17

Figura 4 - Visão geral da arquitetura do Flutter 18

Figura 5 - Fluxo de trabalho do servidor Node.JS 20

Figura 6 - Coleções mongodb 21

Figura 7 - Documentos de banco de dados noSQL 22

Figura 8 - Documentos com referência para outros documentos 22

Figura 9 - Etapas do desenvolvimento da API e do aplicativo 23

Figura 10 - Endpoints para cadastro de características 26

Figura 11 - Endpoints para recuperar as características do animais 27

Figura 12 - Endpoints para deletar as características do animais 27

Figura 13 - Endpoints para atualizar as características do animais 28

Figura 14 - Endpoints para cadastrar animais perdidos e achados 29

Figura 15 - Código *JavaScript* para cálculo da distância euclidiana 31

Figura 16 - JSON para ranquear animais 31

Figura 17 - Body de retorno do cálculo de distância entre os animais 32

Figura 18(a) - Tela inicial de pets perdidos 33

Figura 18(b) - Tela inicial de pets achados 33

Figura 19 - Botões de navegação inferior 34

Figura 20 - Método de filtros por espécie 34

Figura 21 - ComboBox da tela cadastre seu pet 35

Figura 22 - Tela de cadastro 36

Figura 23 - Telas de inserir imagem da câmera ou galeria 37

Figura 24(a) - Tela principal de pets perdidos 38

Figura 24(b) - Tela detalhes 38

Figura 25 - Menu gaveta 39

Figura 26(a) - Listagem de pets cadastrados pelo usuário 40

Figura 26(b) - Detalhes do pet 40

Figura 27 - Tela de ranking 41

Figura 28 - Modelo de dados das características 42

Figura 29 - Modelo de dados das publicações de animais perdidos e achados 43

LISTA DE ABREVIATURAS E SIGLAS

ASPCA - *American Society for the Prevention of Cruelty to Animals*
AOT - *Ahead of Time*
API - *Application Programming Interface*
CEULP - Centro Universitário Luterano de Palmas
CPU - Unidade de Central de Processamento
IBGE - Instituto Brasileiro de Geografia e Estatística
HTTP - Hypertext Transfer Protocol
ID - *Identity*
IoT - Internet of Things
JIT - *Just in Time*
JSON -JavaScript Object Notation
KDD - *Knowledge Discovery in Databases*
GPS - *Global Positioning System*
REST - Representational State Transfer
SDK - *Software Development Kit*
SGBD - Sistema de Gerenciamento de Banco de Dados
ULBRA - Universidade Luterana do Brasil
UI - *User Interface*
UX - *User Experience*
TCC - Trabalho de Conclusão de Curso
WEB - *World Wide Web*

SUMÁRIO

1.	**INTRODUÇÃO**	**7**
2.	**REFERENCIAL TEÓRICO**	**8**
2.1.	Animais de estimação	8
2.2.	Aplicativos móveis	9
2.3.	API REST	10
2.4	Distância Euclidiana	11
3.	**METODOLOGIA**	**12**
3.1.	Materiais	12
3.1.1	Dart	13
3.1.2.	Flutter	14
3.1.3	Node.JS	17
3.1.3	MongoDB	19
3.2	Métodos	22
4.	**RESULTADOS**	**23**
4.1	Desenvolvimento da API	24
4.1	Aplicação do cálculo da Distância Euclidiana	26
4.3	Desenvolvimento do aplicativo	30
4.4	Modelo de dados	42
5.	**CONSIDERAÇÕES FINAIS**	**40**
6.	**REFERÊNCIAS**	**41**

1. INTRODUÇÃO

Os animais de estimação têm múltiplas funções na sociedade humana e muitas vezes são considerados uma forma de trazer conforto, companheirismo e aumentar a autoestima das pessoas. Conforme Brotto (2019), a psicologia reconhece que os efeitos benéficos da convivência com animais de estimação em adultos e crianças são diversos, com foco na redução do estresse, no combate às crises depressivas e no aumento do senso de responsabilidade.

Normalmente quando um animal de estimação foge, seus tutores imprimem cartazes com a foto do animal e espalham nas ruas. Da mesma forma, compartilham fotos em redes sociais, aplicativos de mensagens instantâneas e grupos de proteção aos animais. Sendo, na maioria dos casos, difícil encontrar um local específico para estruturar essas informações, deixando a divulgação dos dados do animal perdida e desordenada dentre tantas outras, impossibilitando o processo de identificação.

Vários animais se perdem nos centros urbanos, e, dependendo da distância percorrida por eles, dificilmente conseguirão voltar para casa sem o auxílio de uma pessoa. Por isso, há uma grande demanda a ser atendida dado a necessidade de centralizar as informações a respeito dos animais desaparecidos, com recursos tecnológicos como por exemplo: a tecnologia de análise de dados e filtros de buscas.

Os aplicativos móveis tornaram-se aplicativos essenciais na realidade da maioria das pessoas, com o desenvolvimento contínuo de recursos em smartphones, existem aplicativos para todos os tipos de consumo, como: câmeras, sensores, GPS, conectividade Bluetooth, WiFi, e mais. Quanto mais funções, os celulares, tablets, smartphones e smartwatches têm, mais possibilidades e detalhes um aplicativo pode oferecer.

As abordagens sobre aplicações móveis são divididas em: nativa, web, híbrida, interpretada e gerada. Os aplicativos nativos são desenvolvidos usando as ferramentas e linguagens de programação fornecidas para uma determinada plataforma móvel. Esses aplicativos são executados apenas em celulares com a plataforma de destino.

Os aplicativos móveis que são construídos nativamente na modelagem estruturada para atender especificamente ao sistema operacional do qual são oriundos, tendem a ter melhor desempenho e dispõem de uma experiência mais facilitada ao usuário final. Tendo em vista que foi completamente desenvolvido para atender o mesmo sistema da plataforma do dispositivo e ou equipamento móvel da qual o usuário final já faz uso.

Flutter é uma estrutura conhecida por economizar tempo dos desenvolvedores, pois eles não precisam escrever código para todas as plataformas individualmente. Por ser híbrido, pode-se criar produtos digitais para várias plataformas ao mesmo tempo, customizando designs para os usuários e capturando todas as funcionalidades sem perder as características nativas.

A proposta deste trabalho foi a implementação de um aplicativo para unificar as informações acerca dos dados dos animais domésticos. Passando a responsabilidade pelo exame e interpretação de dados para o computador, que implicam em técnicas para filtragem de animais por status e espécie, bem como a possibilidade de fazer um ranqueamento entre animais semelhantes perdidos e achados.

O aplicativo desenvolvido tem intuito de ajudar tutores a reaver animais de estimação desaparecidos, devido a centralização de informações de animais perdidos e encontrados pode ser facilitador no auxílio e reencontro dos animais perdidos com os seus respectivos donos. Ao unir todas as informações referentes à busca em uma única plataforma torna mais fácil a procura por um animal desaparecido.

Ao acessar a plataforma várias tarefas podem ser realizadas, tais como: cadastrar animais perdidos e encontrados, filtro por status e espécie, associação automática por intermédio da função de ranqueamento e a de informar se o animal foi encontrado e devolvido ao dono. Bem como, a possibilidade de entrar em contato com o autor de alguma das postagens via contato telefônico.

Uma simples busca na *Google Play Store* revela alguns aplicativos com a finalidade de encontrar animais perdidos, a saber: *PeTrace*, um aplicativo colaborativo para emissão de alertas e notificações de animais perdidos ou

encontrados e *Pupz* do Brasil, chamado SuperAppPet do Brasil, por conter vários produtos e serviços para animais de estimação em um só lugar. Entretanto, não foi possível encontrar um aplicativo que além de buscas de animais perdidos, também fizesse o ranqueamento de animais semelhantes conforme uma das propostas deste trabalho.

2. REFERENCIAL TEÓRICO

Esta seção apresenta os principais conceitos e definições relacionadas aos animais de estimação, aplicativos móveis, API REST e distância euclidiana. A compreensão desses conceitos foram a base para o desenvolvimento deste projeto.

2.1 Animais de estimação

Segundo Serpell (2011), as interações entre humanos e animais evoluíram ao longo do tempo. Com isso, certas categorias de animais passaram a ocupar espaços especiais na vida das pessoas, evoluindo para um vínculo afetivo. Este vínculo define o animal como um animal de estimação. O início da domesticação animal tem raízes antigas e não é necessariamente um grande evento ocidental e moderno.

Além de representar uma fonte de apego e afeto, os animais de estimação desempenham diversos papéis na vida humana, no ambiente familiar, e ainda mais amplamente, como cães de caça, cães de guarda, cães pastores, cães de trabalho policial e guias para grupos especiais de necessidades do cão cego e outras atividades, (SERPELL, 1993).

No Brasil, análises estatísticas apresentam um crescimento relevante de animais domésticos em domicílios, o que pode servir como dado importante para conceituar a relação entre o homem e o animal. De acordo com o IBGE (2013), em 2013, o Brasil chegou a ocupar a 4ª posição em número de animais de estimação em comparação com outros países do mundo, o equivalente a 132 milhões de animais. Validando que a convivência entre seres humanos e animais tem aumentado e se tornado cada vez mais existente na sociedade.

O crescimento do número de animais de estimação ajuda a perceber que os animais são de grande importância na preservação da saúde física e mental das pessoas. Curiosamente, os animais com os quais os humanos têm contato direto podem realmente ter um impacto relevante na qualidade de vida (SAVALLI & ADES, 2016). Dado esse cenário, há um vínculo entre as pessoas e os seus animais de estimação a partir da construção de uma memória afetiva que está ligada diretamente com o físico e mental dos seres humanos (JAROLEMAN, 1998).

Devido a grande importância dos animais de estimação para o ser humano, um estudo realizado pela ASPCA (2012) em Nova York, envolveu uma pesquisa com um número aleatório de donos de animais de estimação para questionar a quantidade de cachorros ou gato perdidos nos últimos cinco anos, com o intuito de entender quantos desses animais de estimação que ainda continuam perdidos e quantos acabaram voltando para casa. O resultado da pesquisa foi que os donos de gatos eram menos propensos a encontrar seus gatos - apenas 74% dos gatos perdidos foram recuperados, em comparação com 93% dos cães perdidos.

2.2. Aplicativos móveis

Os aplicativos móveis tornaram-se aplicativos essenciais na realidade da maioria das pessoas, com o desenvolvimento contínuo de recursos em smartphones, existem aplicativos para todos os tipos de consumo, como: câmeras, sensores, GPS, conectividade Bluetooth, WiFi, e mais.

Considerar a funcionalidade de um aplicativo pode requerer experiência, criatividade, programação e domínio de mercado. Quanto mais funções, os celulares, tablets, smartphones e smartwatches têm, mais possibilidades e detalhes um aplicativo pode oferecer.

Segundo Lopes (2013, p.21), os aplicativos têm acesso direto ao hardware e aos recursos do sistema operacional do dispositivo. Ele pode ser integrado com recursos avançados e outros aplicativos, podendo manipular a operação do dispositivo e até substituir ou complementar a funcionalidade nativa.

Ainda segundo Lopes (2013, p.22), executar um aplicativo dentro do navegador é mais seguro que instalar aplicativos no aparelho, as restrições de seguranças são maiores. Boa parte dos usuários não sabem o impacto que pode levar as permissões que eles aprovam, principalmente quando a lista é longa e cheia de termos diferentes.

Em suma, os aplicativos móveis que são construídos nativamente na modelagem estruturada para atender especificamente ao sistema operacional do qual são oriundos. Tendem a ter melhor desempenho e dispõem de uma experiência mais facilitada ao usuário final, tendo em vista que foi completamente desenvolvido para atender o mesmo sistema da plataforma do dispositivo e ou equipamento móvel da qual o usuário final já faz uso.

Embora todas essas características anteriormente citadas sejam suficientes para desenvolver um aplicativo móvel, existe a necessidade de armazenar dados em outro ambiente, e não apenas na memória do celular. Linguagens de programação de *back-end* lidam com a funcionalidade de bastidores de aplicativos da web. É o código que conecta a Internet aos bancos de dados, gerencia as conexões do usuário e alimenta os aplicativos móveis. O *back-end* trabalha em conjunto com o front-end para entregar o produto ao usuário final.

2.3. API REST

Os aplicativos de *software* que interagem entre aplicativos de alto desempenho precisam executar essas consultas com rapidez e segurança. A abstração arquitetônica de *software* REST API é mais adequada para construir esses modelos de negócios. A API REST fornece um conjunto de rotinas e padrões estabelecidos para o desenvolvimento de aplicativos robustos e fáceis de criar, fornecendo recursos para consumo de dados sem afetar as regras de negócios do aplicativo.

Traduzido para o português, *Application Programming Interface* significa "Interface de Programação de Aplicativos", ou seja, um conjunto estabelecido e documentado de rotinas e padrões pelos quais um aplicativo de software está autorizado a usar a funcionalidade fornecida por esse aplicativo sem conhecer o consentimento da implementação.

Segundo Souza (2020), *Application Programming Interface* trata-se de um conjunto de requisições que permite a comunicação de dados entre aplicações. A criação de uma API utilizando o protocolo HTTP facilita a independência dos componentes dos sistemas e consequentemente a proteção dos dados (JORGE, 2020). Para isso, a API utiliza requisições responsáveis pelas operações básicas necessárias para a manipulação dos dados tais como: *GET, POST, PUT, PATCH* e *DELETE*.

Um dos objetivos do *REST* consiste em oferecer suporte à implantação gradual e fragmentada de alterações em uma arquitetura já implantada. O HTTP foi modificado para dar suporte a esse objetivo, introduzindo requisitos e regras de controle de versão para estender cada elemento de sintaxe do protocolo. (FIELDING, 2000).

Dentre as vantagens da API *REST*, a separação de aplicações *front-end* e *back-end*, é considerada um passo fundamental para a proteção do armazenamento de dados, dessa forma são realizadas apenas as trocas de informações, e não há tratamento de regras de negócio (CARVALHO, 2022).

2.4. Distância Euclidiana

Segundo Aquino Neto et al. (2020), a distância euclidiana é definida como a soma das raízes quadradas das diferenças entre x e y em suas respectivas dimensões. Matematicamente, a distância euclidiana entre dois pontos é o comprimento da linha que os conecta [18,19]. Se u = (x_1, y_1) e v = (x_2, y_2) são dois pontos no espaço euclidiano, então a distância euclidiana entre u e v é dada por:

$$EU(u,v) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (1)$$

Para pontos que possuírem n dimensões, como A = $(x_1, x_2,..,x_n)$ e B = $(y_1, y_2,..,y_n)$ a distância euclidiana entre A e B pode ser calculada por (2) e generalizada em (3).

$$EU(a,b) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \quad (2)$$

$$= \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (3)$$

Segundo Aquino Neto et al. (2020), a distância euclidiana é a equação conhecida por calcular distâncias entre diferentes tipos de objetos, como distâncias ponto a linha. Em matemática, a noção de distância foi generalizada para espaços métricos abstratos. Em algumas aplicações, especialmente ao comparar distâncias, pode ser vantajoso omitir a raiz quadrada final ao calcular a distância euclidiana.

O resultado dessa omissão é chamado de distância euclidiana ao quadrado. Pode ser expresso como uma soma de quadrados. Dados dois pontos P = $(x_1, x_2,\ldots,x_n)$ e Q= $(y_1, y_2,\ldots,y_n)$

$$EU^2(P,Q) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2$$

$$= \sum_{i=1}^{n} (x_i - y_i)^2$$

A métrica de distância euclidiana é baseada na distância entre dois vetores representados em um espaço bidimensional. Ela calcula a diferença entre dois pontos projetados em um plano e seus cálculos são baseados no teorema de Pitágoras.

3. METODOLOGIA

Nessa seção são apresentados os materiais e os métodos utilizados no processo de desenvolvimento do aplicativo e da API.

3.1 Materiais

Os materiais utilizados para o desenvolvimento deste trabalho são o Dart, Flutter, Nodejs e Mongodb, conforme descritos a seguir.

Dart é uma linguagem de programação multiparadigma para o desenvolvimento de aplicativos web, móveis e *desktop* que tem sua documentação

no site oficial da linguagem (DART, 2021). A linguagem Dart permite que o código seja criado para ser executado no cliente e também no servidor, mas neste trabalho o Dart será usado apenas no cliente móvel como linguagem de desenvolvimento.

Flutter é um framework baseado na linguagem de programação Dart que pode criar aplicativos compilados nativamente para sistemas operacionais Android, iOS, *Windows*, Mac, Linux, Fuchsia e Web. Neste trabalho será utilizado para desenvolvimento de interface de usuário e pode ser encontrado no site oficial do framework (FLUTTER, 2021).

Node.js é um ambiente de tempo de execução *Javascript* do lado do servidor. A principal razão para sua adoção é sua alta escalabilidade. Além disso, sua arquitetura, flexibilidade e baixo custo o tornam uma boa opção para implementação de componentes de microsserviços e arquiteturas serverless. Neste trabalho o Node.js foi utilizado para o desenvolvimento da API, cujo sua principal função foi possibilitar a comunicação e consumo de dados externos ao aplicativo. A documentação do *framework* pode ser encontrada no site oficial da ferramenta (NODE.JS, 2022).

O MongoDB é um banco de dados de código aberto, de alto desempenho e flexível, e é considerado um dos principais banco de dados NoSQL que pode ser encontrado no site oficial do banco de dados (MONGO, 2022). Ao contrário dos bancos de dados de modelo relacional onde trabalha-se com registros em linhas e colunas. O NoSQL trabalha com coleções e documentos, sendo que os documentos podem ser descritos como dados em um formato de valor-chave, neste caso, usando o formato *JSON* (JavaScript Object Notation). Neste trabalho o MongoDB foi utilizado para armazenar os dados da plataforma em servidor externo.

3.1.1 Dart

Dart é a linguagem de programação base do Flutter, criada pelo Google em 2012. A linguagem foi constituída para ter a melhor performance e incorporar o máximo possível de dispositivos da atualidade, tanto *web* quanto os dispositivos móveis e *desktop*. Assemelhando bastante ao *JavaScript*, a linguagem foi criada para substituí-lo e ser a principal linguagem embutida nos navegadores. No entanto, a *Google* abandonou totalmente a ideia do Dart substituir o *JavaScript*. Com isso,

criou mecanismos para que as linguagens consigam atuar juntas, tornando-as uma opção agradável e amigável (HOSTGATOR, 2020).

A linguagem Dart utiliza mais de uma plataforma, e cada uma delas traz diversos recursos altamente versáteis, podendo ser utilizados em diferentes plataformas, seja aplicativo *mobile*, *desktop*, *scripts* e *back-end*. Seguindo o paradigma de programação orientada a objetos, suporta interfaces, classes abstratas, genéricos e tipos opcionais. Seus objetos herdam da classe *Object*, ou seja, implementam herança simples. Sua sintaxe é baseada na linguagem C, tornando também muito similar a linguagens atualmente populares, como Java e C#. Ainda assim, o Dart tenta reduzir um pouco os ruídos característicos de linguagens baseadas no C.

Por ser fortemente tipada, não é necessário colocar um tipo, pois o Dart consegue inferir os tipos automaticamente. Diferentemente de outras linguagens, para tornar-se um atributo, método ou classe como privado, no Dart basta colocar um *underline* (_) no início do nome. Sua compilação pode ser em *ahead-of-time* (AOT) que modifica o código implementado em código de máquina, possibilitando a performance de uma aplicação nativa e *just-in-time* (JIT) que compila o código diretamente no *device*, com a tempo de execução, permitindo um retorno em tempo real da modificação e aumentando a velocidade do ciclo de desenvolvimento. (ALMEIDA, 2019).

3.1.2 Flutter

Flutter é um *framework* criado pela *Google* para o desenvolvimento de aplicativos *Android* e *iOS* nativos com uma única base de código. A princípio, o *framework* foi desenvolvido para compilação móvel, mas atualmente o Flutter é multiplataforma, e seu código também pode ser compilado para plataformas *desktop* e *web*. Uma das principais vantagens do *framework* é que os desenvolvedores podem resolver problemas comuns de uma maneira comum, focando em problemas de aplicativos em vez de arquitetura e configuração.

Ao usar o Flutter para desenvolver um aplicativo, o código é compilado na linguagem base do dispositivo, ou seja, o aplicativo é verdadeiramente nativo, de forma que os recursos do dispositivo podem ser acessados sem a necessidade de

terceiros com um desempenho excelente, independente da plataforma ser iOS ou Android, conforme Figura 1.

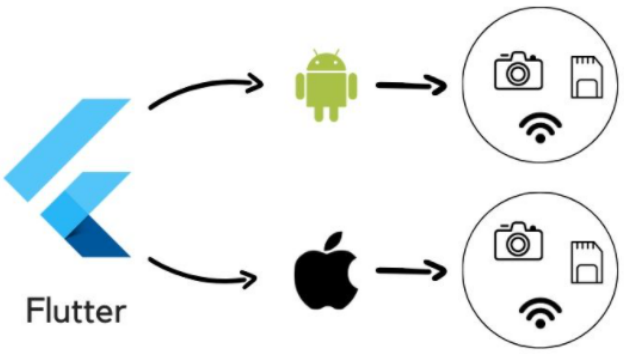

Figura 1 - Recursos nativos do Flutter - Fonte: Andrade (2020)

O Flutter utiliza a linguagem Dart para desenvolvimento de suas aplicações e a criação da interface é feita por meio de árvores de *widgets*. Sendo assim, como *widgets* são nativos do Flutter, sem necessidade de intermediários, diminui a incompatibilidade na compilação do mesmo código em múltiplos sistemas operacionais. A Figura 2 demonstra uma aplicação com o Flutter e Dart, sendo a primeira o código e a segunda o resultado da compilação.

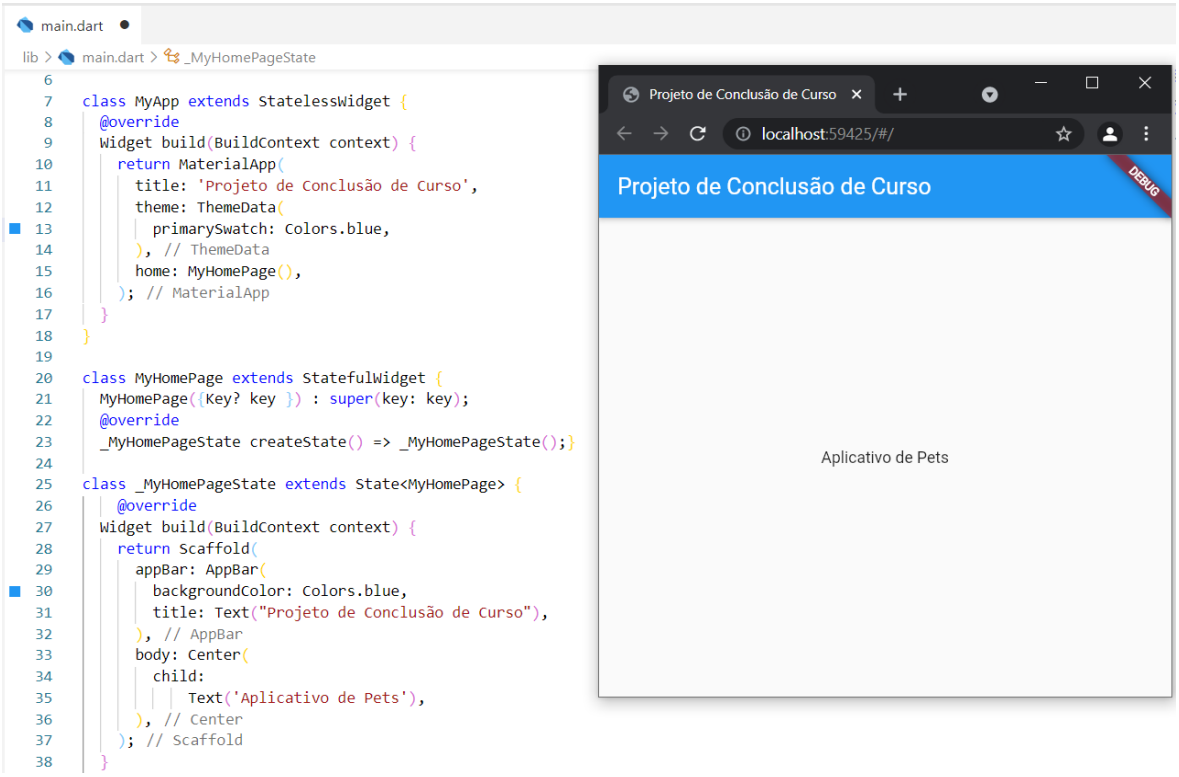

Figura 2 - Código e o resultado da compilação Flutter

Conforme Zammetti (2020), o Flutter tem um rico sistema para estilizar os *widgets* de várias formas, uma vez que sua essência é orientada a widgets e também por meio de *widgets* que foram feitos para estilização de outros *widgets*. Como pode ser observado nas imagens anteriores, o fluxo de desenvolvimento do Flutter é orientado ao design, e os *widgets* são os blocos básicos da interface de aplicação. Os *widgets* podem ser estruturais tais como botões, menus e etcs, de estilo como: fontes, cores e formatação de textos, de *layout* para alinhamentos de margens e espaçamentos, e *widgets* com *design* nativos para as plataformas *Android* (*Material*) e *iOS* (*Cupertino*).

Para mais, o Flutter foi criado para otimizar a criação de novos *widgets*, bem como, a customização dos já existentes. A hierarquia dos *widgets* do Flutter é formada por objetos compostos, expressos por meio de referências de um objeto para o outros, ou seja, onde cada um herda propriedades de seus superiores.

Os *widgets* descrevem a aparência de sua visualização de acordo com sua configuração e estado atual. Quando o estado de um *widget* muda, o *widget* reconstrói sua descrição, que a estrutura difere da descrição anterior para determinar as mudanças mínimas necessárias na árvore de renderização subjacente para a transição de um estado para o próximo. A Figura 3 demonstra essa hierarquia.

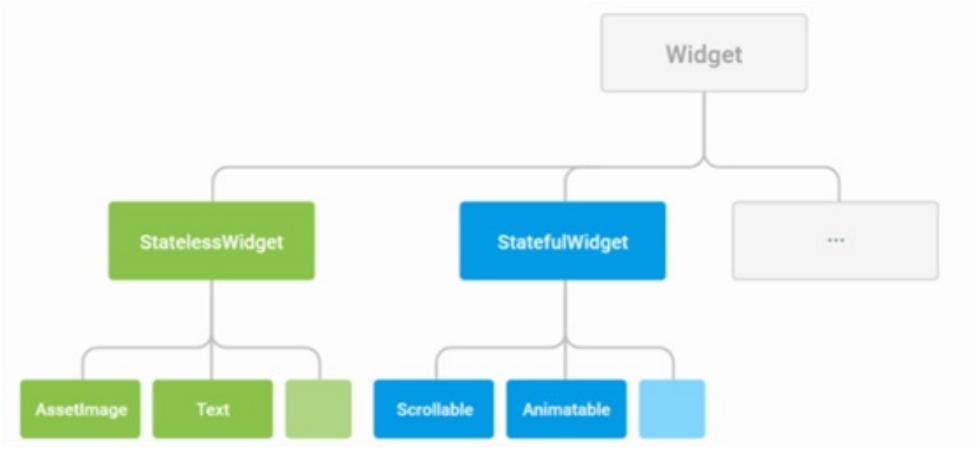

Figura 3 - Hierarquia dos *widgets* - Fonte: Flutter (2022)

O Flutter possui uma arquitetura dividida em três camadas, sendo elas: *framework*, *engine* e *embedder. Framework* é a camada implementada em Dart que engloba os *widgets*, pacotes como *SKY_ENGINE* responsável pela *UI* e *UX* do

Flutter. O framework em si, é construído sobre a *Engine* C/C++ e é puramente código Dart, enquanto o *embedder* é a camada que incorpora o Flutter à várias plataformas, como podemos visualizar na Figura 4.

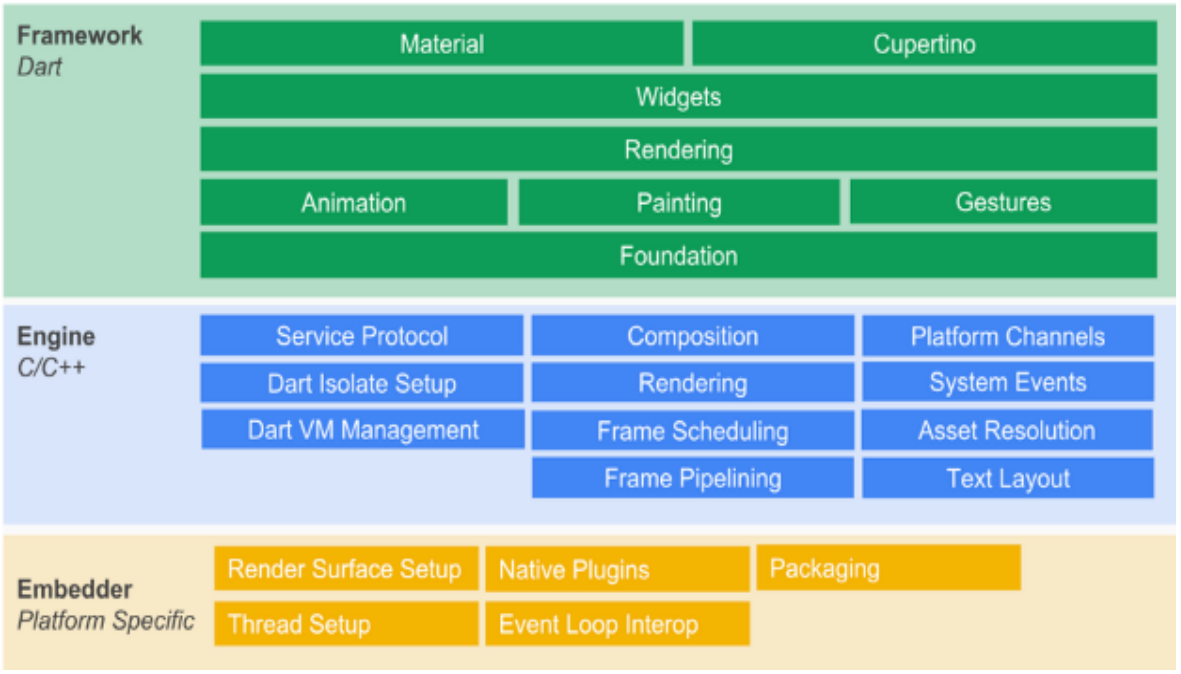

Figura 4 - Visão geral da arquitetura do Flutter - Fonte: Flutter (2022)

3.1.3 Node.JS

Node.JS é uma tecnologia usada para executar código *JavaScript* fora do navegador. Com ele é possível construir aplicações em geral, desde websites até APIs e microsserviços.

Conforme Pessôa (2022), devido a sua característica altamente versátil Node.JS pode ser utilizado nas famosas APIS REST, web *scrapping*, *chatbots*, IoT, web servers e aplicações *desktop*. Dentre as principais vantagens podemos citar:

- Possibilidade de criar aplicativos *desktop*, aplicativos móveis e até sites *SaaS*;
- Programável em diferentes paradigmas, desde orientado a objetos, funcional, imperativo e dirigido à eventos;
- Código aberto, ou seja, qualquer pessoa pode ter acesso ao seu código fonte, podendo customizar ou contribuir diretamente para a comunidade;
- Totalmente escalável, de acordo com sua documentação oficial, o Node.JS foi criado para construir aplicações web escaláveis.

Do ponto de vista do desenvolvimento de um servidor web, o Node.JS possui vários benefícios, dentre eles, a alta performance, uma vez que foi projetado para otimizar a taxa de transferência e a escalabilidade em aplicações web, tornando-se uma excelente combinação para resolver muitos problemas comuns no desenvolvimento da web (DEVELOPER, 2022).

Ainda segundo Developer (2022), outra vantagem do Node.JS é o robusto Gerenciador de Pacotes do Node (NPM, na sigla em inglês) que fornece acesso a centenas de milhares de pacotes reutilizáveis. NPM possui a melhor coleção de dependências e também pode ser usado para automatizar a maior parte da cadeia de ferramentas de compilação.

O principal bloco de construção de aplicações web com *JavaScript* e *Node*.JS é o *Express*. Com *Express*.js, é possível aprimorar diferentes aspectos do aplicativo da web. Usado para determinar configurações como a localização dos modelos que serão usados para a resposta ou a porta que será usada para estabelecer uma conexão.

O fluxo de trabalho do servidor Node.JS funciona da seguinte forma: os usuários enviam solicitações ao servidor para realizar operações, posteriormente as solicitações entram primeiro na fila de eventos no lado do servidor. Em seguida, a fila de eventos passa as solicitações sequencialmente para o loop de eventos, onde, o loop de eventos verifica a natureza da solicitação.

O *event loop* processa as solicitações sem bloqueio que não requerem recursos externos e retorna as respostas aos clientes correspondentes. Para solicitações de bloqueio, um único *thread* é atribuído ao processo para concluir a tarefa usando recursos externos. Após a finalização da operação, a requisição é redirecionada para o *event loop* que devolve a resposta ao cliente, conforme fluxo da Figura 5.

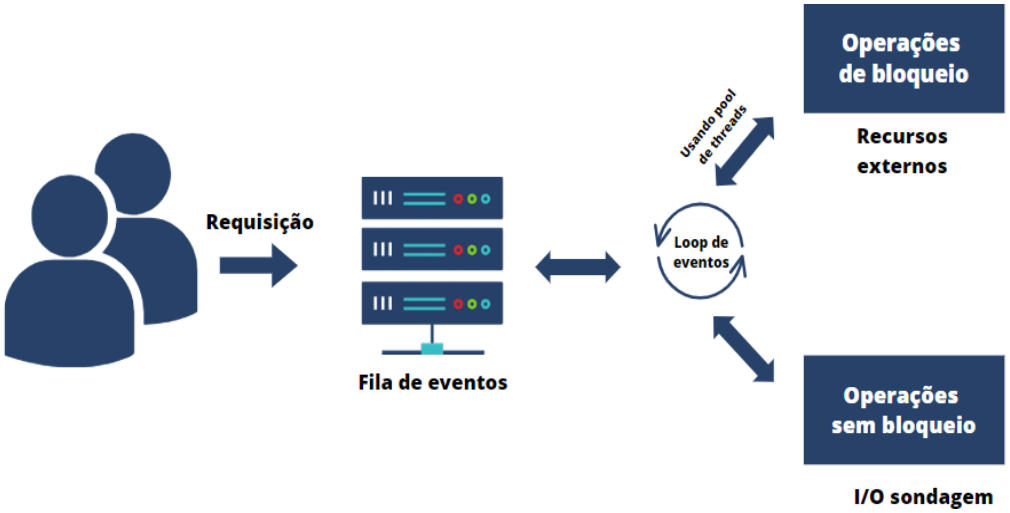

Figura 5 - Fluxo de trabalho do servidor Node.JS

3.1.4 MongoDB

O MongoDB é um banco de dados orientado a documentos de código aberto, projetado para armazenar grandes quantidades de dados e permitir que sejam processadas grandes quantidades de dados com eficiência. Ele pertence aos bancos de dados NoSQL (não relacional) porque o armazenamento e a recuperação de dados no MongoDB não são feitos em formato de tabela (KOVACS, 2021).

Ainda segundo Kovacs (2021), os projetos com Mongo não estão necessariamente relacionados, pois sua hierarquia é totalmente flexível devido ser um banco de dados NoSQL. As informações são armazenadas em coleções e

documentos, onde coleções são subdivisões independentes, que permitem a criação de vários bancos de dados e várias coleções dentro do banco de dados principal.

As coleções são análogas às tabelas em bancos de dados relacionais, conforme Figura 6, se uma coleção não existir, o MongoDB criará a coleção quando for armazenado os dados dessa coleção pela primeira vez (MONGOBD, 2022).

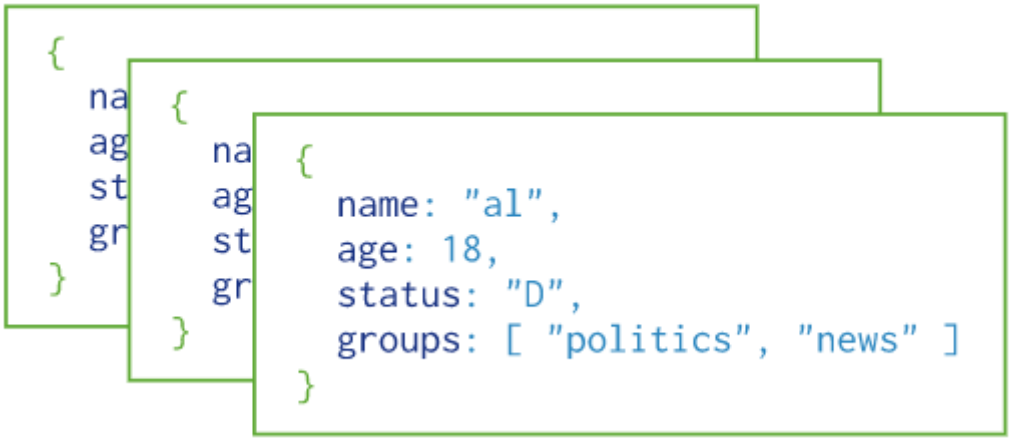

Figura 6 - Coleções mongodb (MONGODB, 2022)

De acordo MongoBD (2022), por padrão, as coleções não exigem que seus documentos tenham o mesmo esquema, ou seja, os documentos em uma única coleção não precisam ter o mesmo conjunto de campos e os tipos de dados dos campos podem diferir entre os documentos de uma coleção. Para alterar a estrutura dos documentos em uma coleção, como adicionar novos campos, remover campos existentes ou alterar os valores dos campos para novos tipos, basta atualizar os documentos para a nova estrutura.

As coleções recebem um UUID imutável, conforme _id representado pela Figura 7, ou seja, um identificador único para cada objeto que será armazenado no banco de dados. Sempre que houver um novo registro no banco de dados, atribui-se um UUID para esse registro, caso seja necessário realizar modificações no objeto, ele pode ser recuperado pelo UUID (LIMA, 2022).

```
_id: ObjectId('638d3a5d819c9cc97de07148')
returned: false
cat_breed: null
dog_breed: "SRD"
color: "branco e preto"
gender: "macho"
owner: "stefanicarol@gmail.com"
pelage: "curta"
size: "grande"
species: "cachorro"
status: "perdido"
photo: "https://firebasestorage.googleapis.com/v0/b/perdimeupet-2444b.appspot.…"
contact: "(63) 98123-9541"
city: "Palmas/TO"
observation: "Possui coleira vermelha"
date: "04/12/2022"
__v: 0
```


Figura 7 - Documentos de banco de dados noSQL

Os documentos criados dentro das coleções NoSQL podem armazenar dados aninhados. Essa conexão de dados permite criar relacionamentos complexos entre eles e armazená-los no mesmo documento, conforme Figura 8, o que torna o trabalho e a pesquisa muito mais eficientes em comparação com o SQL.

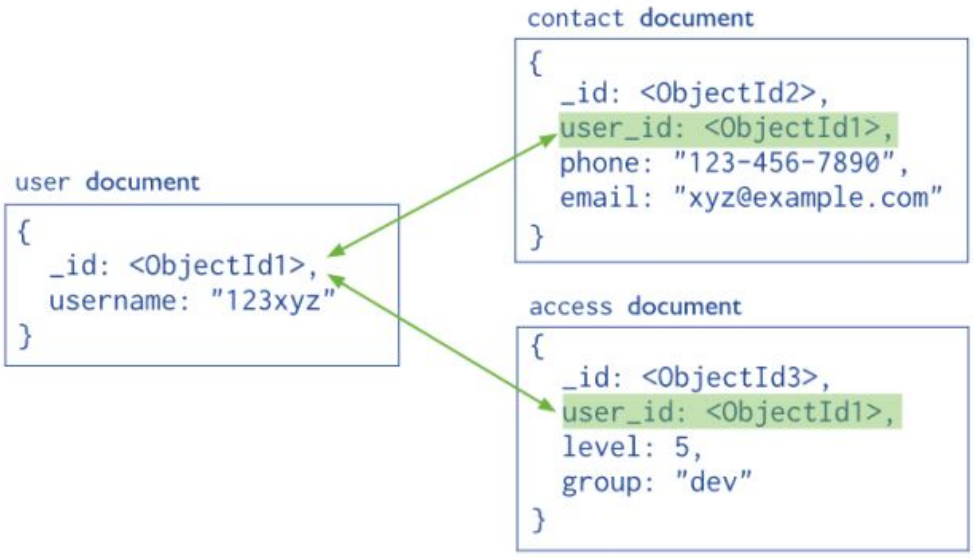

Figura 8 - Documentos com referência para outros documentos (FUMAROLA, 2022)

O MongoBD também fornece suporte de driver oficial para todas as linguagens populares, como C, C++, C# e .Net, Go, Java, Node.js, Perl, PHP,

Python, Engine, Ruby, Scala, Swift, Mongoid. Portanto, pode-se usar qualquer uma dessas linguagens para criar aplicativos usando MongoDB.

3.2. Métodos

Nessa seção são apresentadas as etapas que foram executadas no processo de desenvolvimento da API e aplicativo para auxiliar tutores a reverem seus animais domésticos perdidos, conforme é ilustrado na Figura 9.

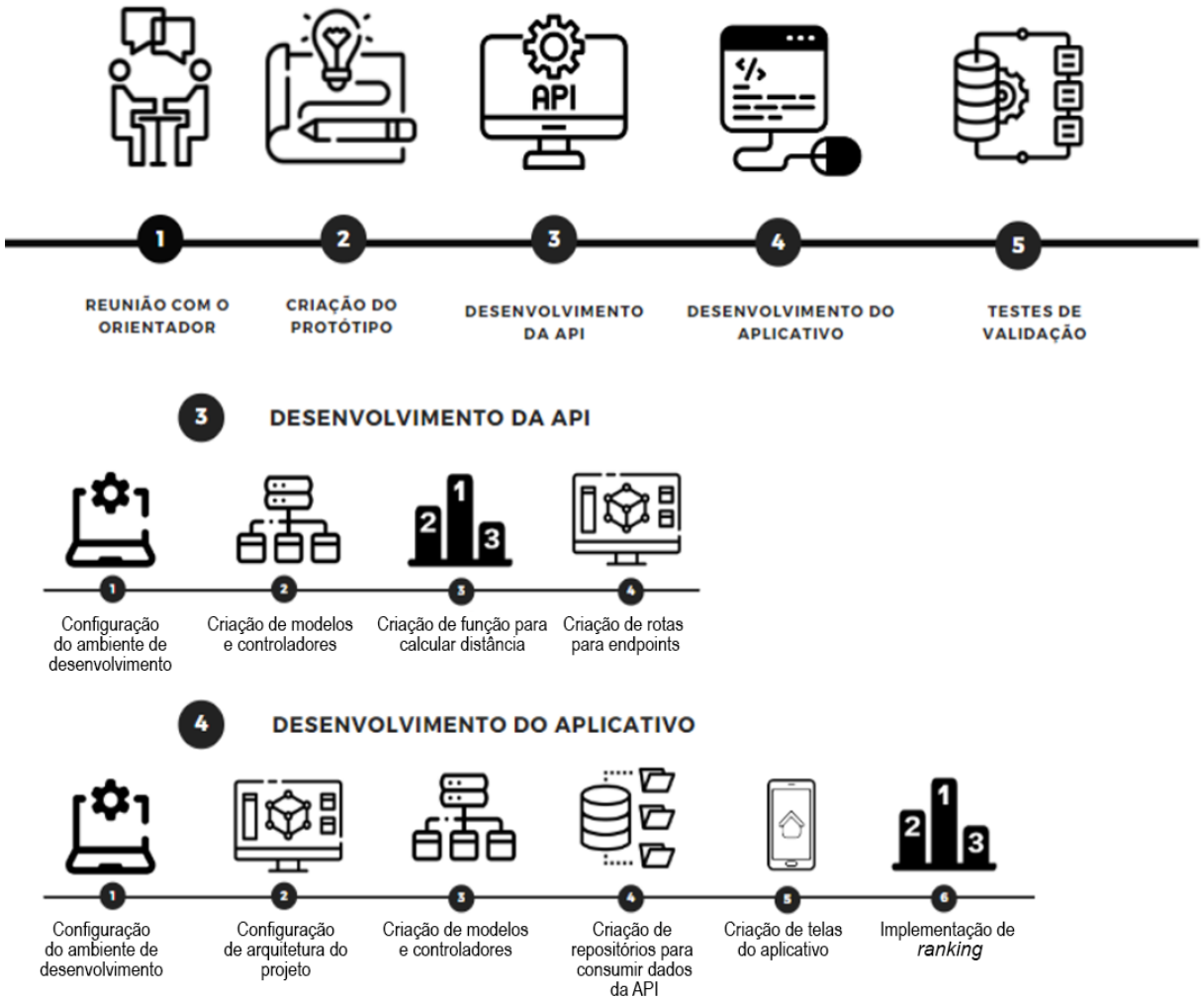

Figura 9 - Etapas do desenvolvimento da API e do aplicativo

A primeira etapa consistiu em reunir com o orientador para levantar o que seria necessário de requisitos para o desenvolvimento do trabalho. Na segunda

etapa foi desenvolvido o protótipo estrutural do aplicativo, com o objetivo de definir o *layout*, as funcionalidades e como seriam as interações entre as funcionalidades.

A terceira etapa tratou-se do desenvolvimento da API para fornecer integração do aplicativo com os dados cadastrados no banco de dados. Este desenvolvimento foi dividido em 4 seções, sendo elas: configurar o ambiente de desenvolvimento, modelar os dados dos animais, implementar função para calcular a distância euclidiana entre animais perdidos e entrados e por último criar os *endpoints* de cadastro, listagem, filtragem e ranqueamento dos animais.

A quarta etapa foi o desenvolvimento do aplicativo, no qual foi dividido em 6 seções, sendo elas: configurar o ambiente de desenvolvimento, implementar a arquitetura do aplicativo, modelar os dados que foram utilizados para cadastro, listagem, filtragem e ranqueamento dos animais, criar repositórios para consumir os dados da API e por último implementar o módulo para ranquear animais a partir de um animal selecionado.

E por fim a quinta etapa, onde foram feitos os testes e validações das funcionalidades do aplicativo, com o propósito de certificar se a plataforma atende o objetivo do trabalho. Tais como: cadastro de animais perdidos, cadastrados de animais achados, verificação se os animais perdidos e achados cadastrados foram listados corretamente e utilização da função de ranqueamento de animais.

4. RESULTADOS

O presente trabalho propõe uma solução que amplie as possibilidades de um tutor reaver seu animal doméstico perdido. Para resolver tal problema, foi desenvolvido um aplicativo e uma API que cruzasse informações de animais aplicando uma função que fizesse um *ranking* de semelhança entre as características dos animais perdidos e animais encontrados.

4.1 Desenvolvimento da API

A API proposta neste trabalho foi desenvolvida utilizando um formato padronizado baseado em requisições HTTP, aplicando os verbos GET, PUT, DELETE e POST. O objetivo é permitir chamadas para o cadastro e listagem das características dos animais, para o cadastro e listagem dos animais perdidos e encontrados, bem como para a aplicação da função para calcular distância entre animais.

A API desenvolvida possui três *endpoints,* um para cadastro, atualização e deleção de características dos animais, outro para para cadastro, listagem, atualização e deleção de dados dos animais e um último para cálculo da distância euclidiana para gerar o ranqueamento dos animais. A Figura 10 apresenta informações do *endpoint* de manipulação das características dos animais, onde está sendo utilizado o verbo *POST* que serve para o cadastro das características.

Para fazer o cadastro de uma característica, deve-se enviar no *Body* (corpo da requisição) o ID (número identificador), a descrição da característica e uma lista com itens conforme Figura 10. Na Figura 10 também é possível verificar o retorno de uma requisição de sucesso utilizando o cliente HTTP gráfico POSTMAN, onde o código do status é 200 (OK) e o retorno de uma requisição no qual o ID já exista no banco de dados no qual o status é 400 (*Bad Request)*.

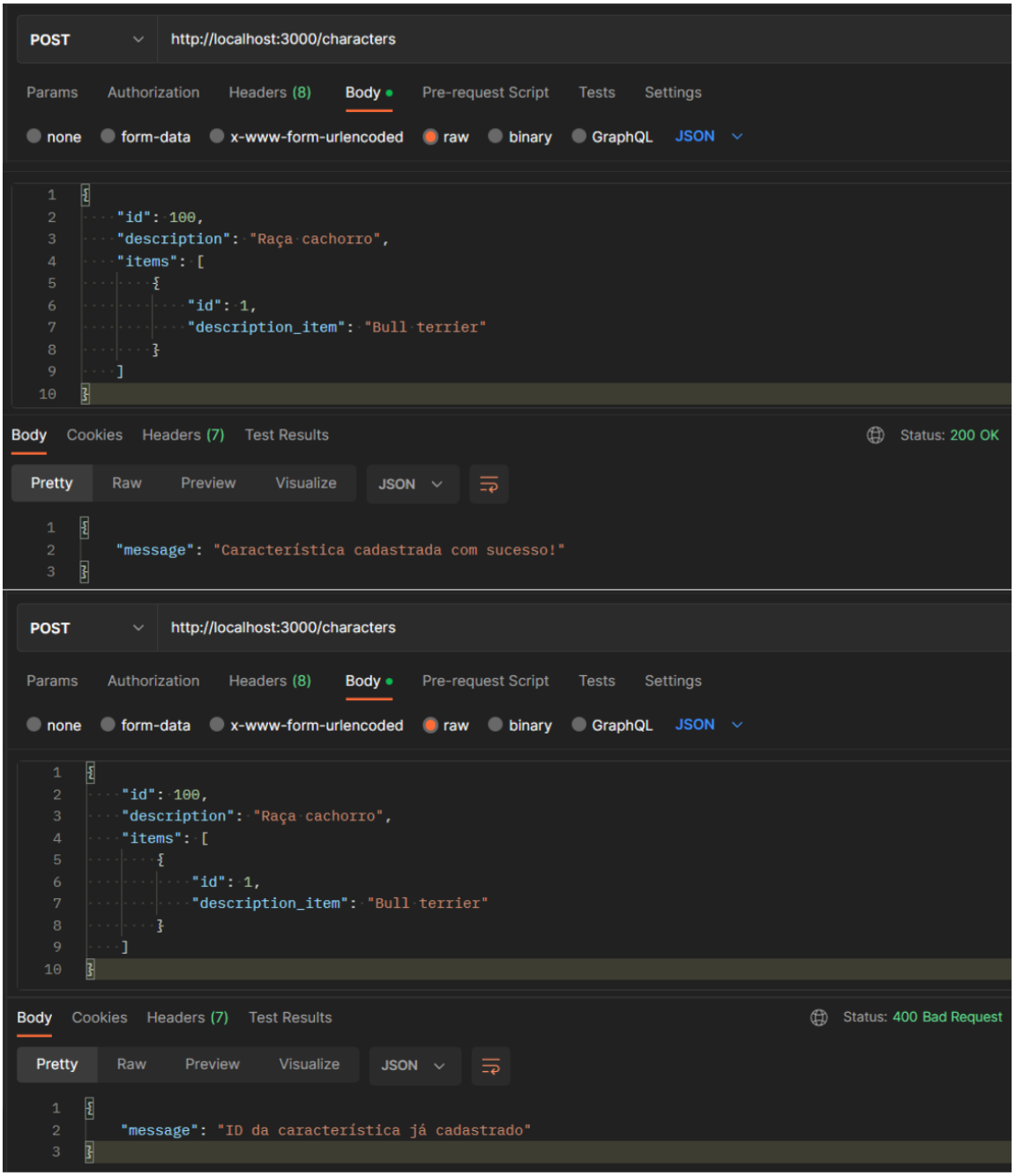

Figura 10 - *Endpoint* para cadastro características

A Figura 11 mostra uma requisição para recuperar os dados de características dos animais utilizando o verbo GET, que em caso de sucesso é retornado uma lista com todas as características cadastradas no banco de dados.

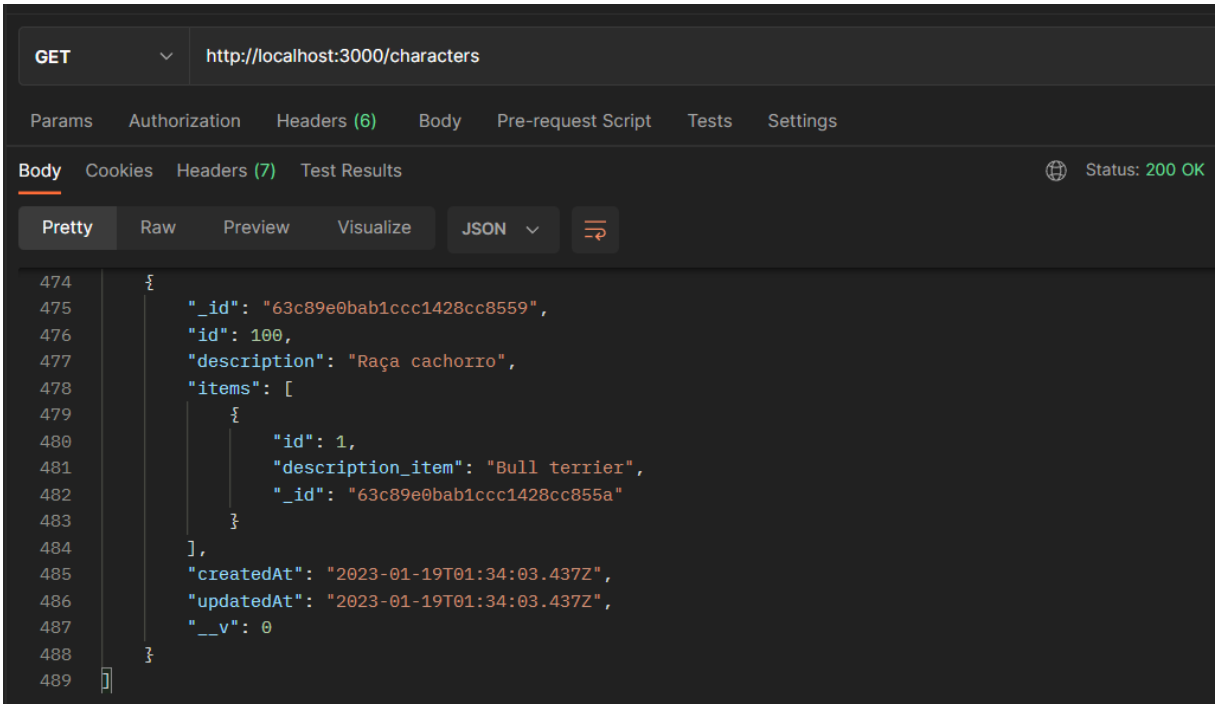

Figura 11 - *Endpoint* para recuperar as características dos animais

A Figura 12 mostra uma requisição para excluir uma característica dos animais utilizando o verbo DELETE, onde deverá ser passado o ID da característica na URL e caso o ID da característica não seja encontrado no banco de dados é retornado o status 404 (*Not Found)* e uma mensagem de erro informando que a característica não foi encontrada e em caso de sucesso é retornado o status 200 (OK) e a mensagem de característica deletada com sucesso.

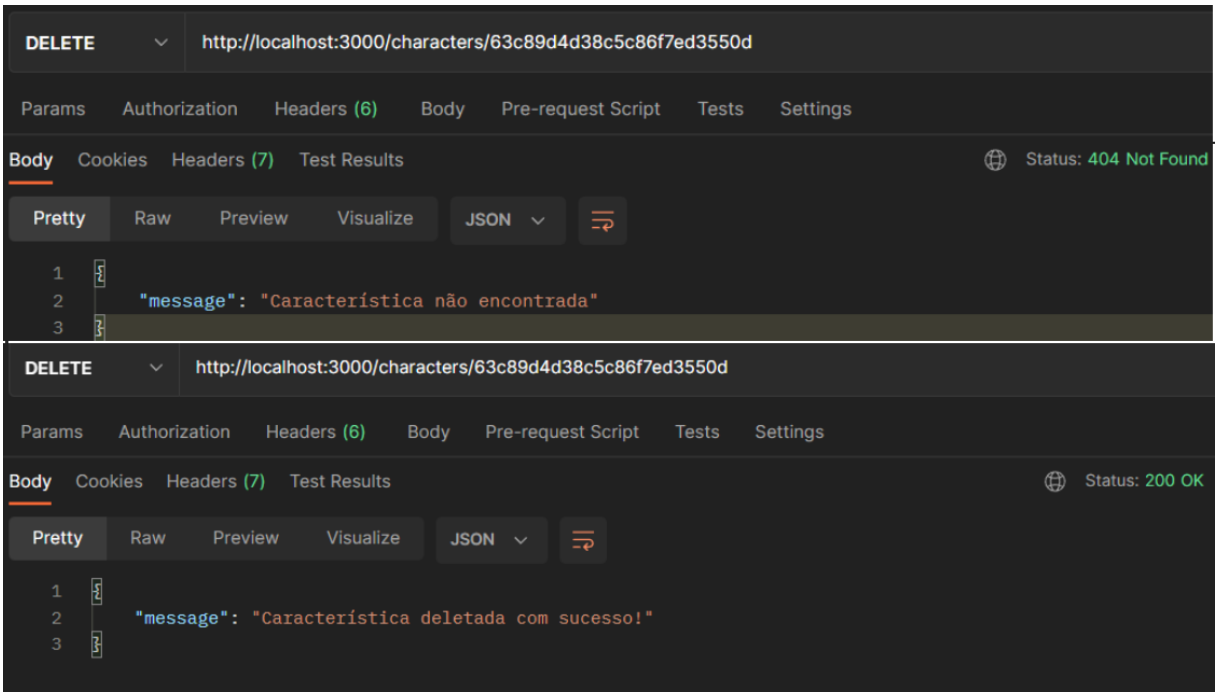

Figura 12 - *Endpoint* para deletar as características dos animais

A Figura 13 mostra uma requisição para atualizar uma característica dos animais utilizando o verbo PUT, onde deverá ser passado o ID da característica na URL e caso o ID da característica não seja encontrado no banco de dados é retornado o status 404 (*Not Found)* e uma mensagem de erro informando que a característica não foi encontrada e em caso de sucesso é retornado o status 200 (OK) e a mensagem de característica atualizada.

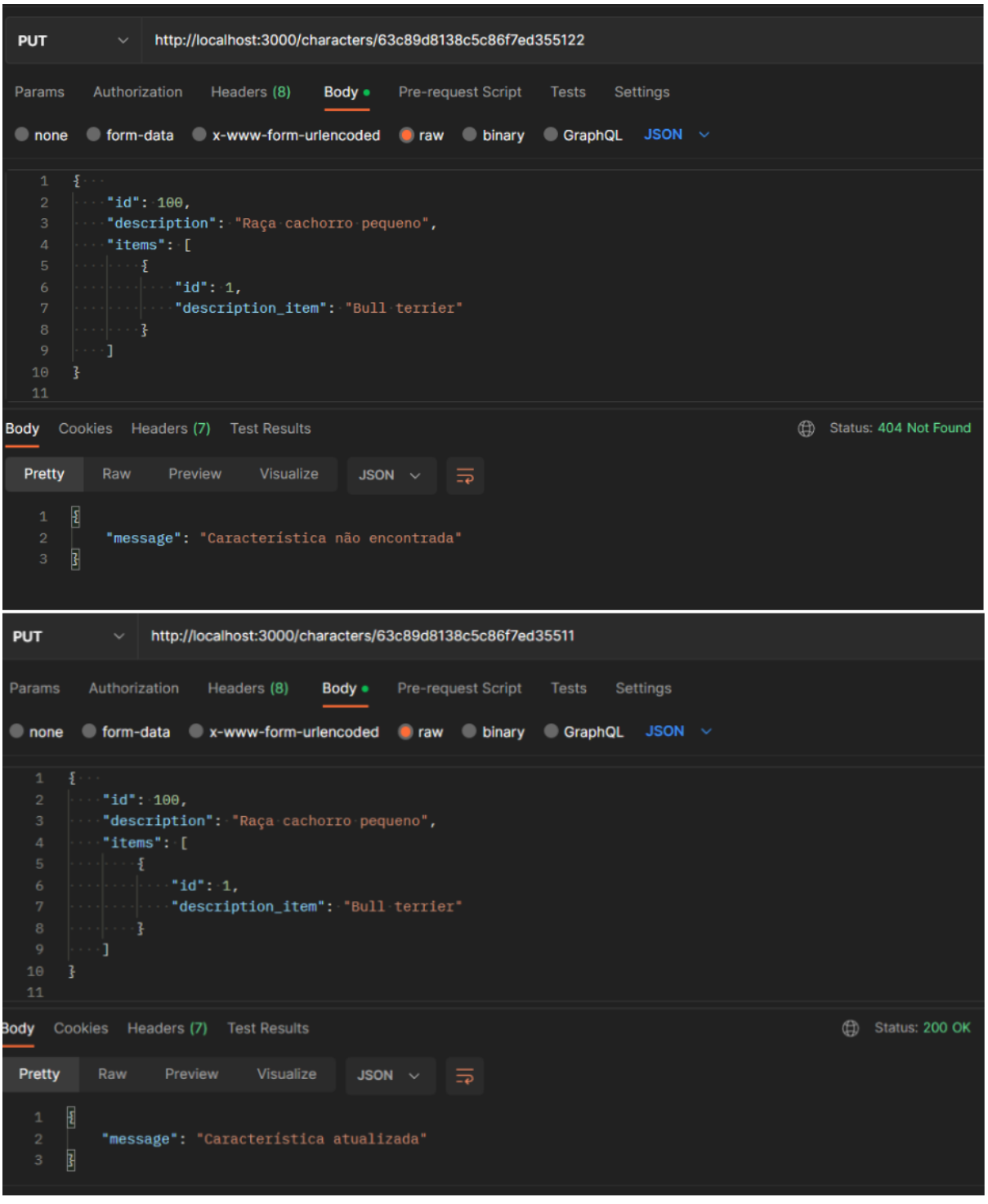

Figura 13 - *Endpoint* para atualizar as características dos animais

O *endpoint* para cadastrar animais possibilita a postagem de animais perdidos e achados na plataforma *mobile*. O verbo utilizado para para o este *endpoint* foi o POST, onde deverá ser passado no *Body* um *JSON* contendo todas as informações referentes às características do animal, conforme Figura 14.

POST http://localhost:3000/pets

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON


```
{
    "returned": false,
    "characters": [
        {
            "id": 1,
            "description": "Raça cachorro",
            "item": {
                "id": 1,
                "description_item": "Bull terrier"
            }
        },
        {
            "id": 2,
            "description": "Gênero",
            "item": {
                "id": 1,
                "description_item": "Fêmea"
            }
        },
        {
            "id": 3,
            "description": "Pelagem",
            "item": {
                "id": 1,
                "description_item": "Curta"
            }
        },
        {
            "id": 4,
            "description": "Cor",
            "item": {
                "id": 1,
                "description_item": "Branca"
            }
        },
        {
            "id": 5,
            "description": "Tamanho",
            "item": {
                "id": 1,
                "description_item": "pequeno"
            }
        },
        {
            "id": 6,
            "description": "Espécie",
            "item": {
                "id": 1,
                "description_item": "Cachorro"
            }
        }
    ],
    "owner": "stefanicarol.dev@gmail.com",
    "status": "perdido",
    "photo": "https://www.petlove.com.br/images/breeds/202905/profile/original/vira-lata-p.jpg",
    "contact": "(63) 9 9999-8888",
    "city": "Palmas/TO",
    "observation": "Sem observação",
    "date": "22/11/2022"
}
```


Figura 14 - *Endpoint* para cadastrar animais perdidos e encontrados

Após cadastrar um animal, caso os dados tenham sido inseridos corretamente o *endpoint* traz como retorno um *body* com status 200 (OK) e mensagem de cadastro efetuado com sucesso e caso haja alguma inconsistência nos dados enviados, o *endpoint* retorna o status 400 (*Bad Request)*.

4.2 Aplicação do cálculo da Distância Euclidiana

Em matemática, a distância euclidiana é a distância entre dois (a e b) pontos que pode ser provada pela aplicação repetida do teorema de Pitágoras. Na API desenvolvida para este trabalho, o cálculo da distância euclidiana recebe duas variáveis como entrada. A primeira variável são os índices das características do animal (a) e a segunda variável é a lista composta pelos índices das características dos animais (b).

Para o desenvolvimento deste trabalho, foi utilizado a função *Math.sqrt() do JavaScript*. Onde para saber a raiz quadrada qualquer de número, basta usar o método *sqrt()*, que recebe um número como parâmetro e retorna um número também.

A Figura 15 mostra o cálculo matemático e suas respectivas implementações em *JavaScript* que foram utilizadas para o desenvolvimento do *endpoint* de ranqueamento. Onde a função recebe duas listas de características de animais (a e b), a cada iteração, incrementa a *sum* variável com o valor do objeto e no final retorna a raiz quadrada de *sum* que representa distância euclidiana.

```
function euclideanDistance(a, b)
{
    sum = 0;
    for (let i = 0 ; i < a.length ; i++)
    {
        d = a[i] - b[i];
        sum = sum + d * d;
    }
    return Math.sqrt(sum);
}
```

Figura 15 - Código *JavaScript* para cálculo da distância euclidiana

Para testar o *endpoint* de ranqueamento entre animais perdidos e achados, deverá ser enviado um JSON com dois objetos (a e b), o primeiro é o objeto (a) do *pet* a ser combinado, esse objeto deverá conter ID (número identificador) de cada característica do animal. O segundo objeto (b) é a lista dos animais nos quais deverão ser ranqueados. Essa lista de *pets* deverá conter objetos com os IDs das características e o ID do animal, conforme Figura 16.

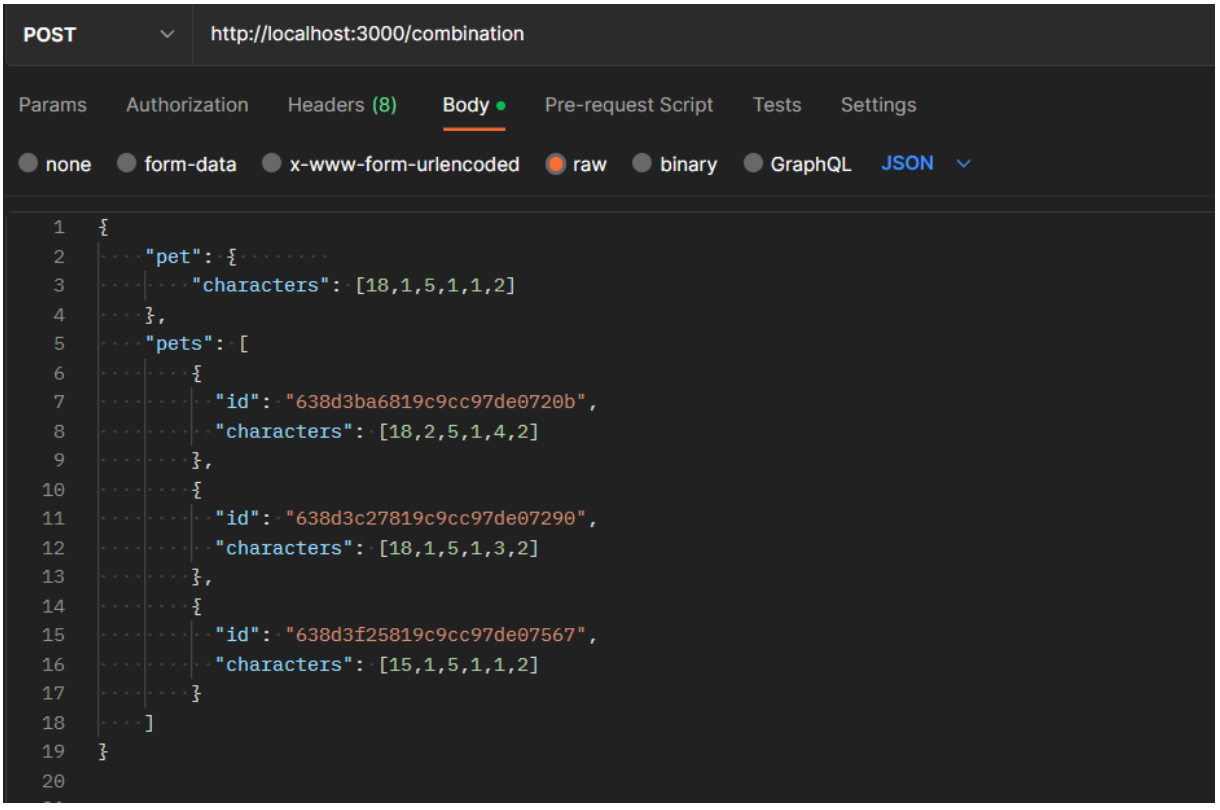

Figura 16 - JSON para ranquear animais

Após enviar os dados dos dois objetos de animais, o *endpoint* traz o *Body de* retorno listando a distância de cada animal da lista de animais (b) para o animal (a). O ID serve para identificar a qual animal pertence a distância, conforme Figura 17.

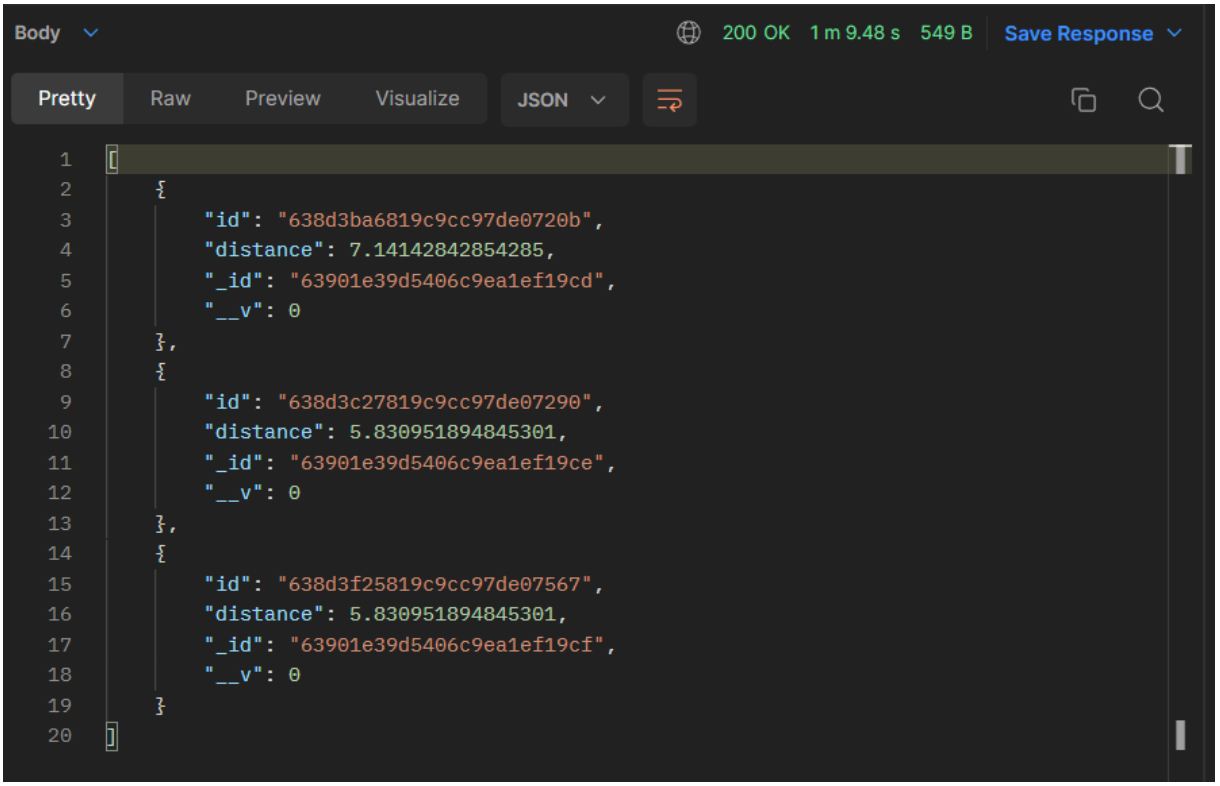

Figura 17 - Body de retorno do cálculo de distância entre os animais

Para apresentar a lista de animais ranqueados na plataforma mobile, foi necessário fazer um filtro para apresentar apenas os animais nos quais o ID fossem iguais aos IDs listados no *endpoint* de ranqueamento, ordenando de forma crescente de acordo com a distância.

4.3 Desenvolvimento do aplicativo

Nesta seção são apresentados os resultados obtidos depois do desenvolvimento do aplicativo. As Figuras 18(a) e 18(b) são constituídas por telas da parte interna do aplicativo. Depois do login, a primeira tela que o usuário vê é a representada na Figura 18(a), onde contém três itens na barra de navegação inferior, nos quais navegam para páginas de acordo com sua descrição.

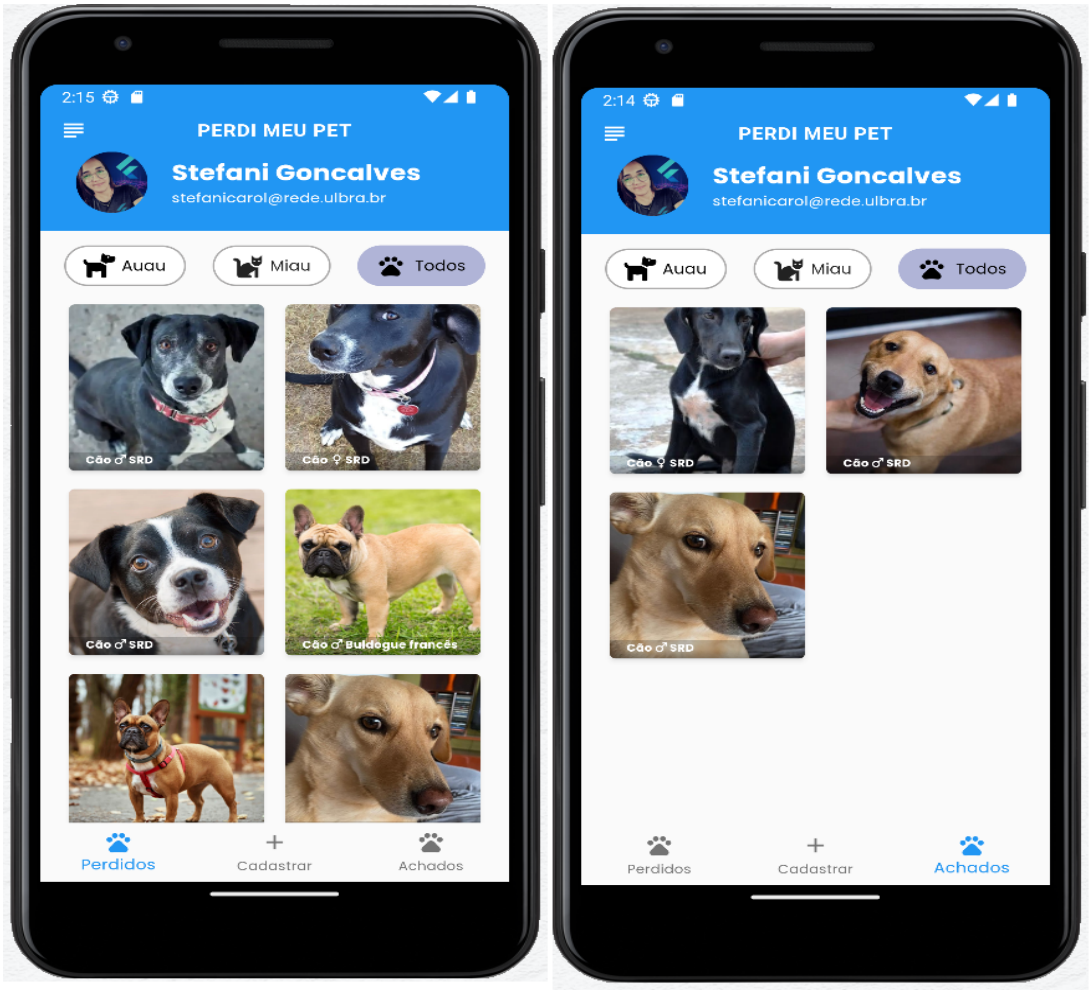

Figura 18(a) - Tela de perdidos

Figura 18(b) - Tela de achados

A barra de navegação inferior apresentada nas Figura 19 serve para navegar para as telas de animais perdidos, cadastro de animais e animais achados. A tela de animais perdidos apresentada na Figura 18(a) é a página inicial do aplicativo depois do login. Ela lista todos os animais perdidos cadastrados na plataforma.

O botão de cadastrar navega para uma página de formulário de cadastro conforme Figura 21, e o último item carrega a tela de listagem de animais achados conforme Figura 18(b), que por sua vez possui o *layout* semelhante ao da tela de animais perdidos, conforme Figura 18(a).

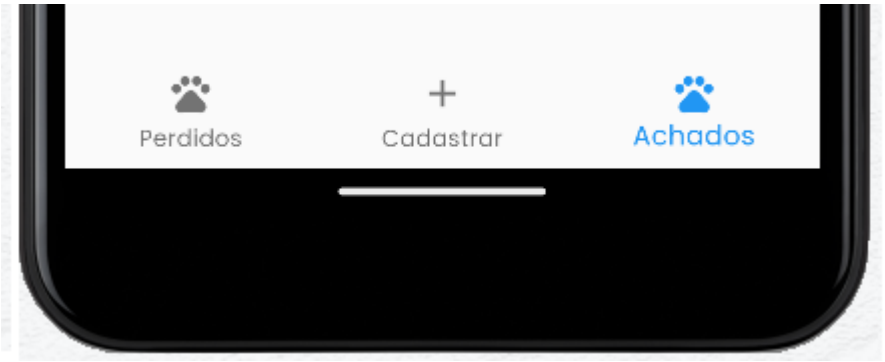

Figura 19 - Botões de navegação inferior

Os cabeçalhos das telas de animais perdidos e achados, possuem uma implementação para fazer filtros de acordo com a espécie do animal, foram implementadas três opções de filtros: filtragem apenas cachorros (Auau), apenas por gatos (Miau) ou ambas espécies (Todos), conforme Figura 20.

Figura 20 - Método de filtros por espécie

O item do meio da barra de navegação demonstrado na Figura 19, redireciona para tela de cadastro, conforme demonstrado na Figura 21. Todos os *comboBox* da tela de cadastro são preenchidos com dados obtidos do *endpoint characters* da API.

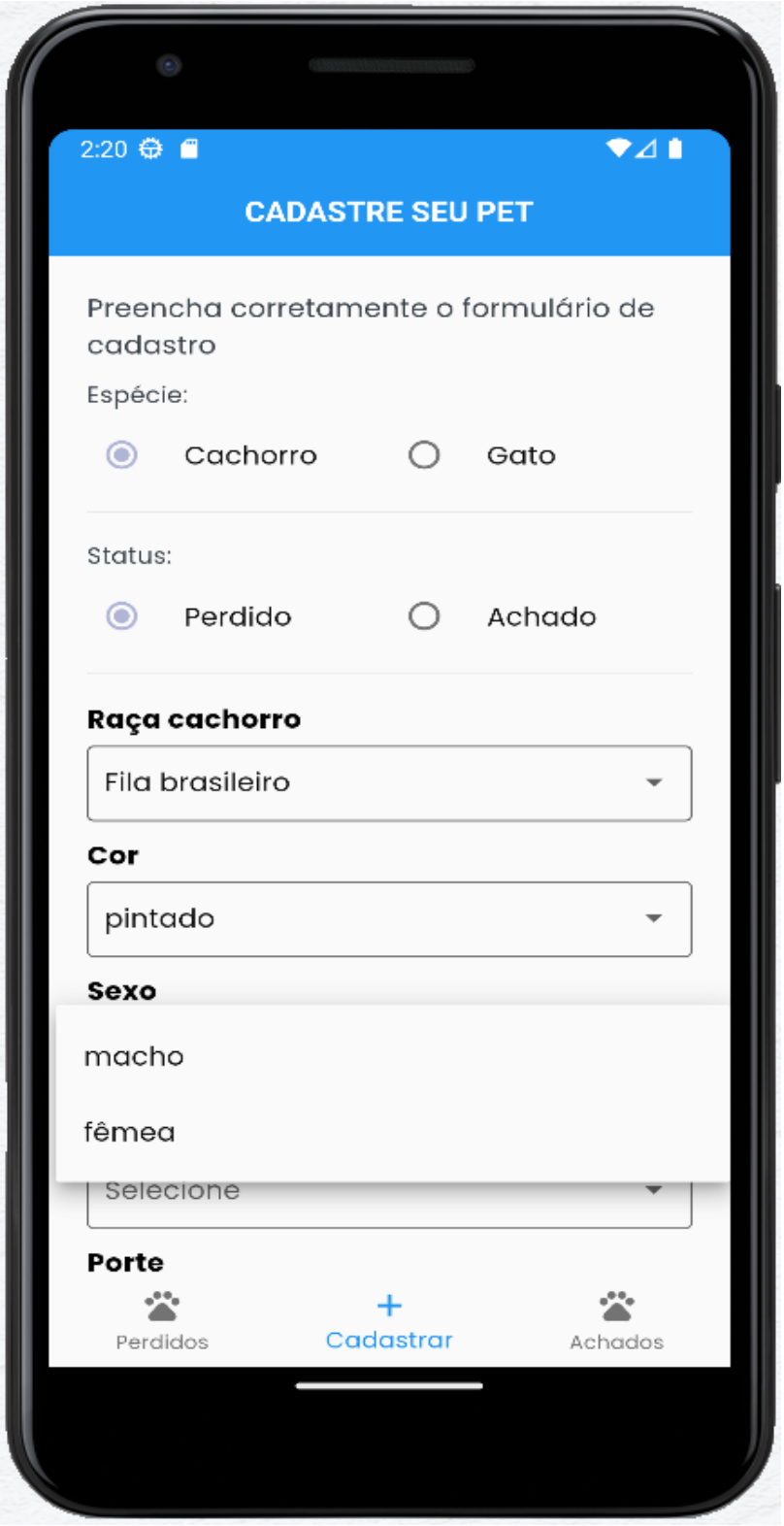

Figura 21 - ComboBox da tela cadastre seu pet

Na tela de cadastre seu *pet* o usuário deverá preencher o formulário com os dados conforme Figura 22. Apenas o campo de observação não é um item obrigatório, os demais campos deverão ser obrigatoriamente preenchidos.

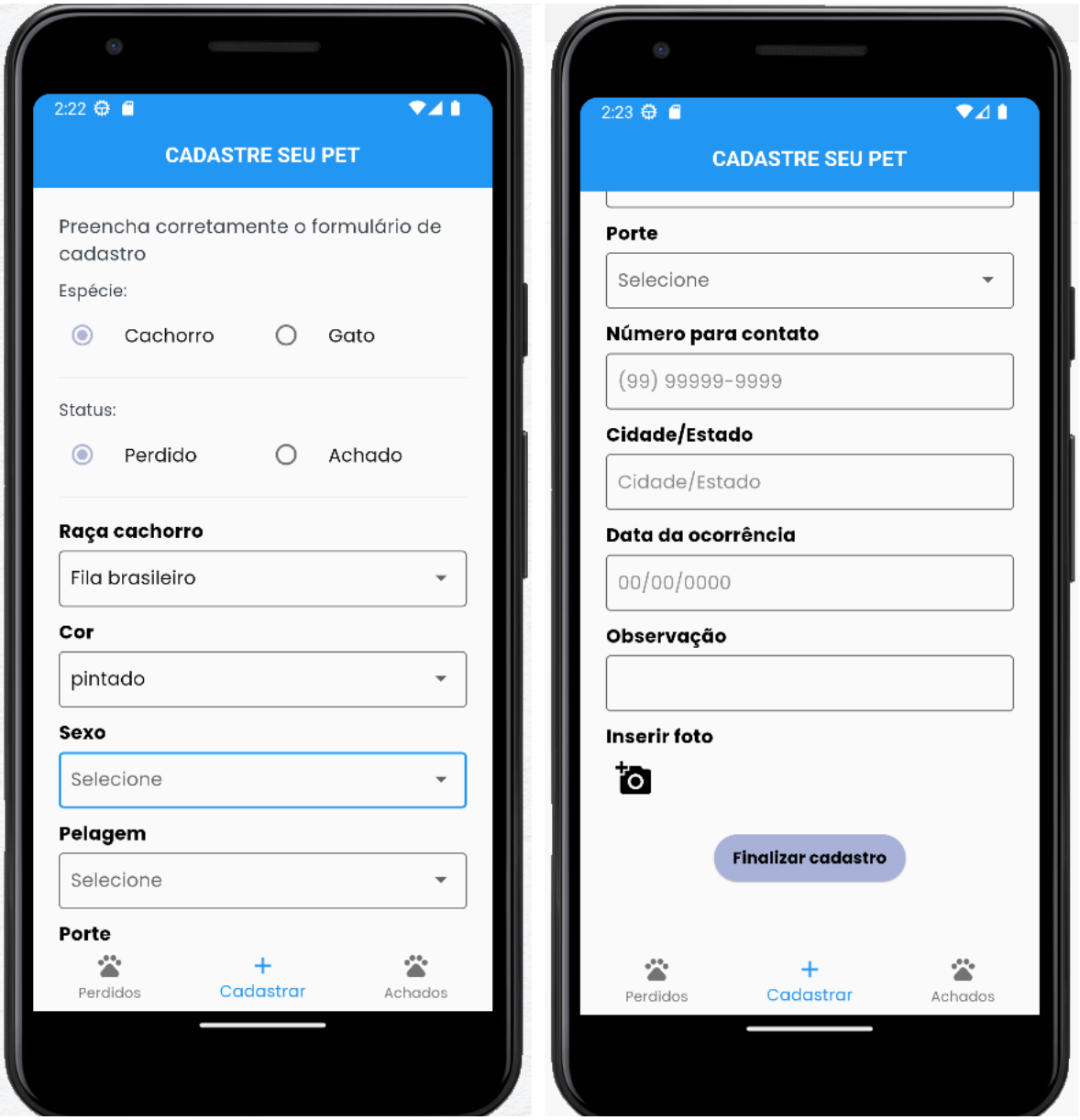

Figura 22 - Tela de cadastro

Para inserir uma foto no cadastro do *pet,* o usuário deverá clicar no ícone da câmera logo abaixo de inserir a foto e abrirá um modal para o usuário escolher se quer fotografar com a câmera ou inserir uma foto da galeria. Ao clicar na opção de câmera o aplicativo irá abrir a câmera do celular para que o usuário possa fotografar o animal. Caso a escolha seja inserir foto da galeria, o aplicativo irá redirecionar o usuário para a galeria do celular conforme Figura 23.

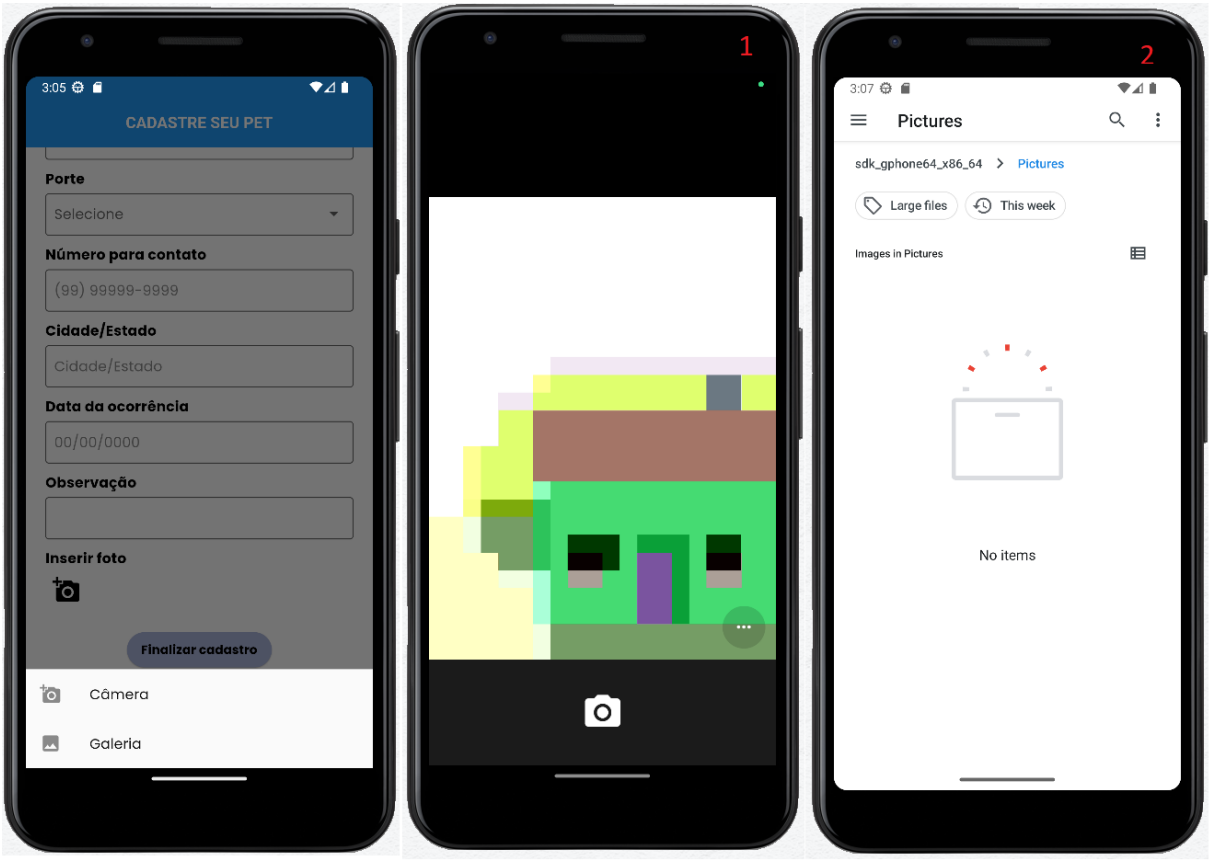

Figura 23 - Telas de inserir imagem da câmera ou galeria

Nas telas de animais perdidos e achados, logo abaixo das opções de filtros, são listados *cards* contendo as imagens e uma breve descrição do animal. Ao clicar em um dos *cards* da Figura 24(a), o usuário irá navegar para a página de detalhes da publicação na qual mostra uma descrição mais detalhada do animal, como mostra a Figura 24(b).

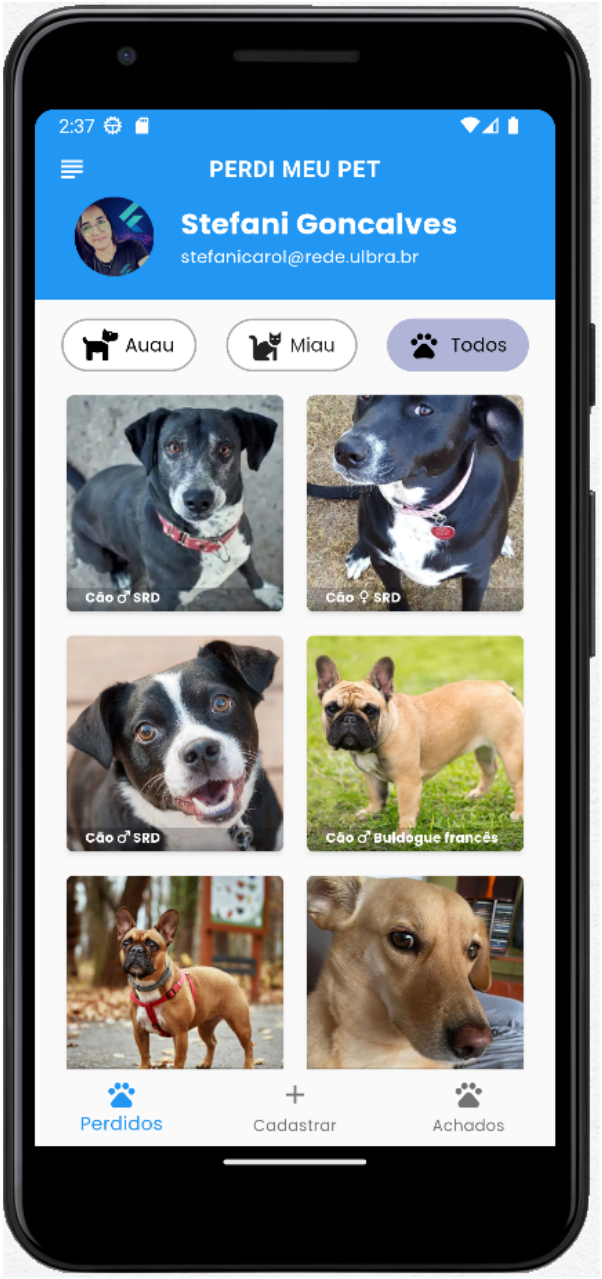

Figura 24(a) - Tela principal pets perdidos

Figura 24(b) - Tela de detalhes

Ao clicar no menu lateral que pode ser encontrado nas telas de animais perdidos ou encontrados, abre-se uma gaveta de navegação (menu drawer) que nos dá a opção de escolher sair do aplicativo ou listar pets cadastrados pelo usuário (Figura 25).

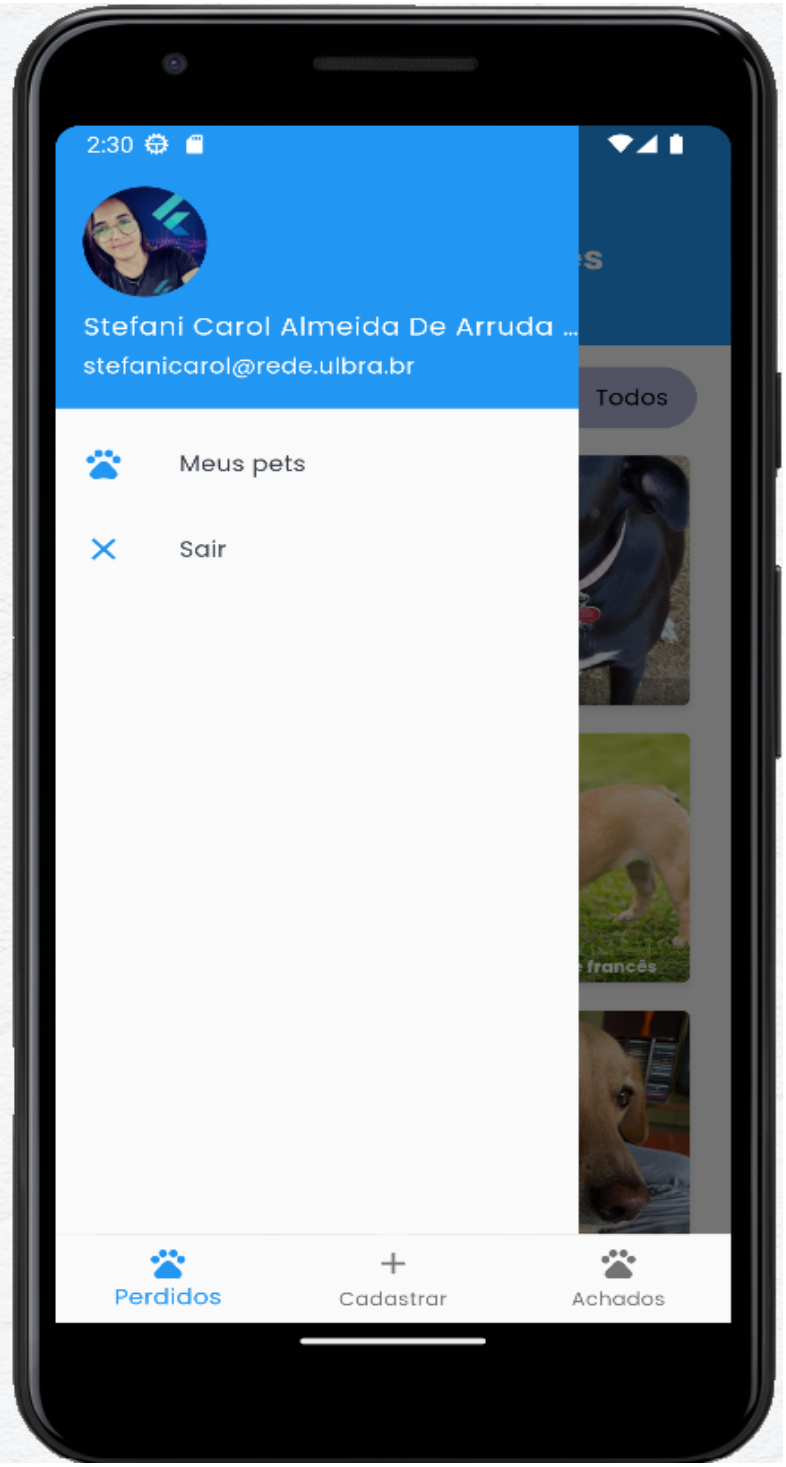

Figura 25 - Menu de gaveta

Ao clicar em Meus pets, a próxima tela nos retorna a lista com todos os pets cadastrados pelo usuário conforme Figura 26(a). Ao clicar em um dos animais listados, navega-se para a tela de detalhes deste animal, conforme Figura 26(b).

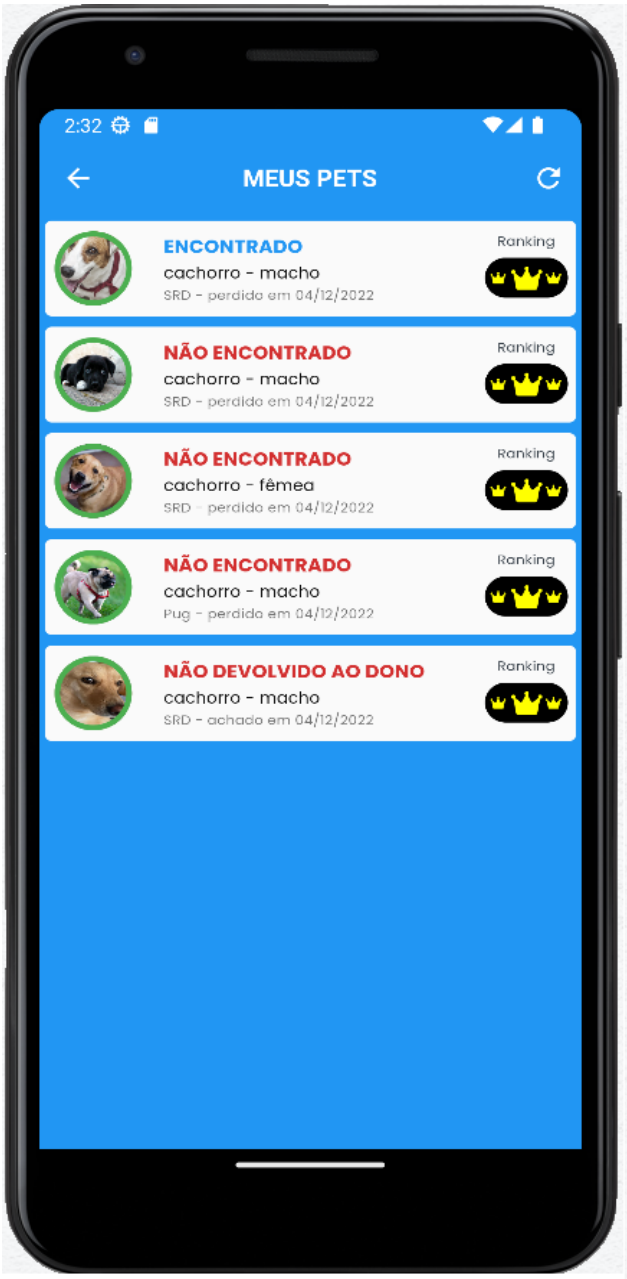

Figura 26(a) - Listagem de pets cadastrados pelo usuário

Figura 26(b) - Detalhes do pet

A Figura 26(b) apresenta as descrições do animal selecionado, bem como a opção de alterar o status do animal, informando que o animal foi encontrado. Ao clicar na opção de ranking o aplicativo consome o *endpoint* para calcular a distância euclidiana e listar os animais semelhantes, passando as informações do animal selecionado, bem como a lista de características dos animais que possuem o status oposto ao dele. Ou seja, se o animal for de status perdido o usuário poderá ranquear os animais achados que sejam semelhantes a ele, e se o animal for achado poderá ser ranqueado os animais de status perdido.

Após o ranking ser concluído o usuário irá visualizar a página de ranqueamento, onde os animais são listados de acordo com a maior semelhança entre o animal ranqueado, conforme Figura 27.

Figura 27 - Tela de ranking

Ao clicar em algum dos ranqueados, o usuário irá navegar para a tela de detalhes onde contém as informações do animal, bem como contato do autor da postagem.

4.4 Modelo dados

O modelo de dados implementado no banco de dados foi composto de duas coleções, uma para armazenar as informações de características dos animais e outra para armazenar as informações de postagem de animais perdidos e achados.

A coleção de características é composta por vários documentos, onde cada documento possui um ObjectID gerado automaticamente pelo MongoDB, um ID gerado pela API de forma incremental, uma descrição da característica e uma lista de itens da caraterística. Cada item contém o ID do item e a descrição do item, conforme Figura 28.

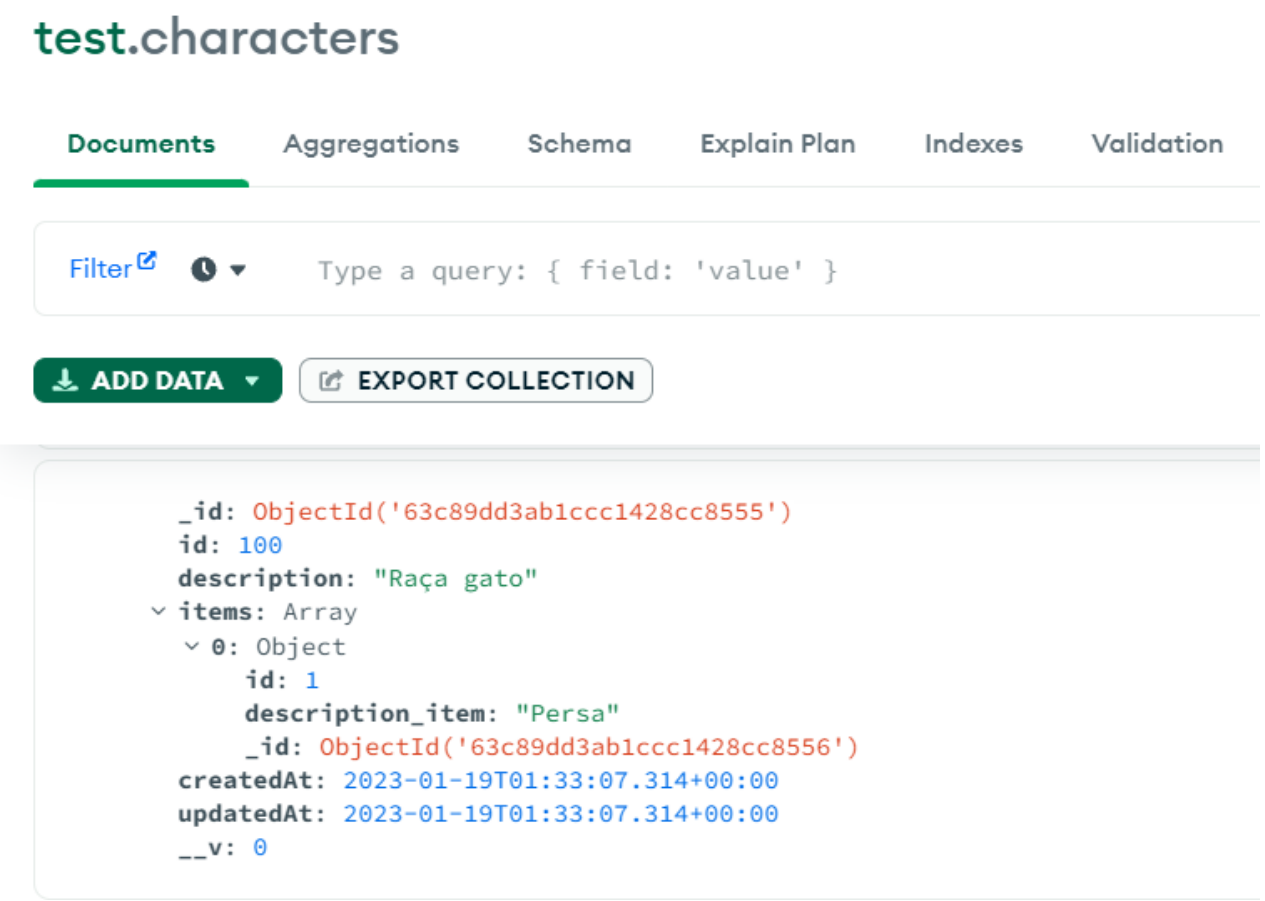

Figura 28 - Modelo de dados das características

A coleção de pets é composta por documentos, onde cada documento possui um ObjectID gerado automaticamente pelo MongoDB e objetos com informações referentes ao animal, tais como, se foi encontrado ou não, status, e-mail do usuário que cadastrou, cidade, contato, data, foto, status e uma lista com as características, conforme Figura 29.

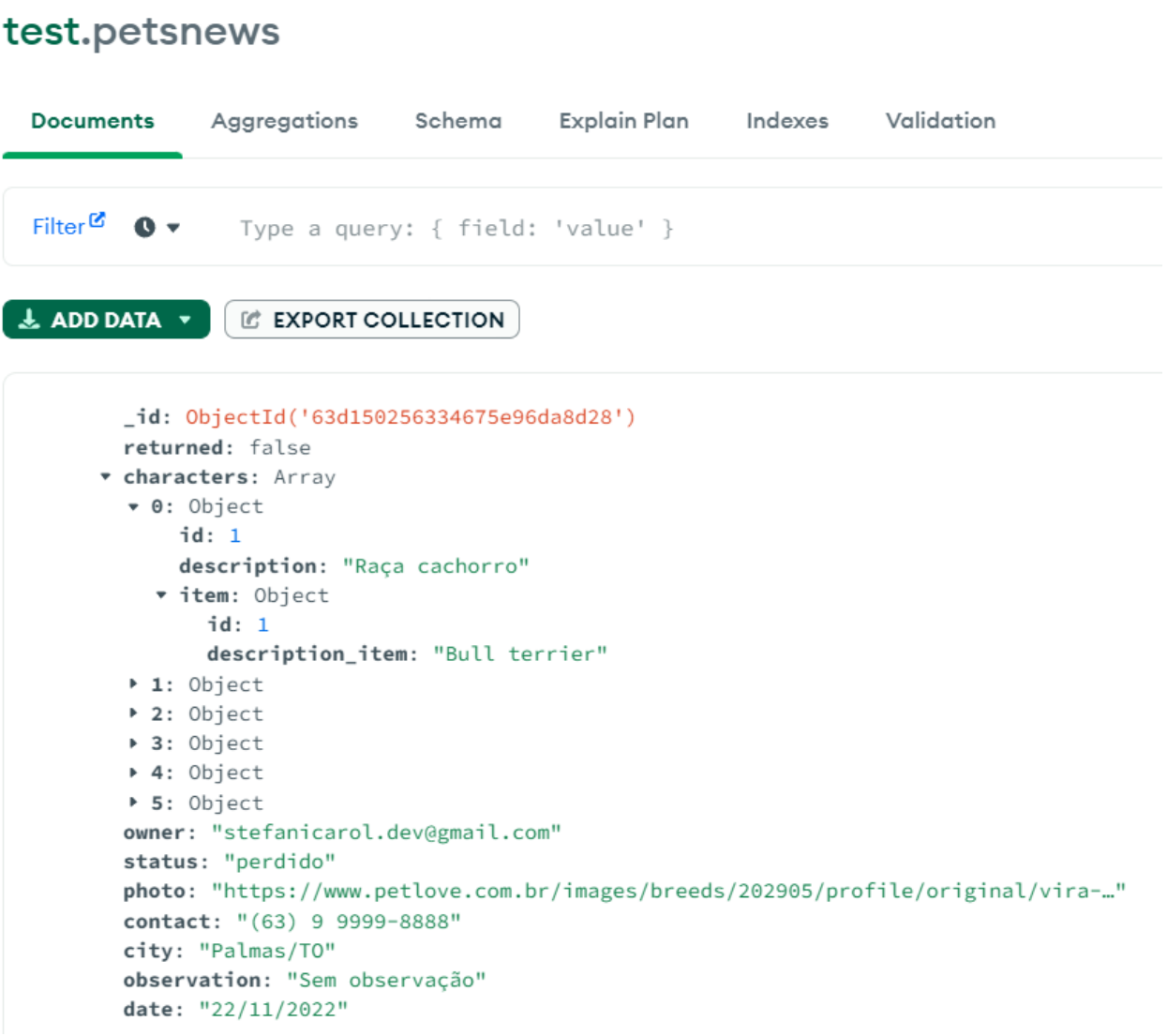

Figura 29 - Modelo de dados das publicações de animais perdidos e achados

Conforme representado nas Figuras 28 e 28, o modelo de dados não está necessariamente relacionado, sua hierarquia é totalmente flexível porém segue um padrão para atender as necessidades do projeto. Devido a utilização do banco de dados NoSQL, as informações estão armazenadas em coleções e documentos.

5. CONSIDERAÇÕES FINAIS

Animais de estimação estão cada vez mais presentes nos domicílios brasileiros. Junto com essa análise vem a preocupação quanto a recuperação de um animal perdido. Este trabalho pretendeu entender sobre os efeitos benéficos da convivência com animais de estimação a dificuldade de um tutor reaver seu animal perdido, a partir de pesquisas com o intuito de captar informações sobre quantos desses animais de estimação que ainda continuam perdidos e quantos acabaram voltando para casa.

Para se atingir uma compreensão de como desenvolver um aplicativo móvel para auxiliar tutores a recuperar seus animais domésticos aplicando o cálculo euclidiano para associar animais perdidos e encontrados a partir de características, definiu-se três objetivos específicos. O primeiro foi estruturar dados de características dos animais, o segundo, desenvolver API integração do aplicativo com o banco de dados e aplicar função para correlacionar características entre animais domésticos perdidos e achados e o terceiro desenvolver um aplicativo móvel.

Como resultado, conseguimos atingir o objetivo de implementar a API e o aplicativo para ranquear animais correlacionados de acordo com seu status e características. Em pesquisas futuras, pode-se implementar funcionalidades de notificações para quando um animal for correlacionado com outro e estiver no pódio.

6. REFERÊNCIAS

ANDRADE, Ana Paula de. **O que é Flutter?** 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-flutter. Acesso em: 26 jun. 2022.

AQUINO NETO, Arlindo Fernandes de *et al*. ACELERADOR DO CÁLCULO DE DISTÂNCIA EUCLIDIANA EM HARDWARE. **Acelerador do Cálculo de Distância Euclidiana em Hardware**, Mossoró, p. 3-3, jun. 2020. Disponível em: https://repositorio.ufersa.edu.br/bitstream/prefix/6465/1/ArlindoFAN_ART.pdf. Acesso em: 08 dez. 2022.

BROTTO, Thaiana. **Influência e Benefícios dos Animais de Estimação na Vida das Pessoas**. 2019. Disponível em: https://www.psicologoeterapia.com.br/blog/infuencia-e-beneficios-dos-animais-de-estimacao-na-vida-das-pessoas/. Acesso em: 05 jun. 2022.

CARVALHO, Thainara. **API Rest: o que é e quais são as vantagens dessa integração?** 2022. Disponível em: https://www.iugu.com/blog/api-rest-o-que-e. Acesso em: 28 nov. 2022.

FLUTTER. **Flutter architectural overview**. 2022. Disponível em: https://docs.flutter.dev/resources/architectural-overview. Acesso em: 26 jun. 2022.

FUMAROLA, Fabio. **Document Oriented Databases**. 2015. Disponível em: https://www.slideshare.net/fabiofumarola1/9-document-oriented-databases. Acesso em: 07 dez. 2022.

HOSTGATOR. **Conheça tudo sobre Dart, a linguagem do Flutter**. 2020. Disponível em: https://www.hostgator.com.br/blog/o-que-e-dart-na-programacao/. Acesso em: 26 jun. 2022.

IBGE - **População de Animais de Estimação no Brasil** - 2013 - ABINPET 79. Disponível em: <https://www.gov.br/agricultura/pt-br/assuntos/camaras-setoriais-tematicas/documentos/camaras-tematicas/insumos-agropecuarios/anos-anteriores/ibge-populacao-de-animais-de-estimacao-no-brasil-2013-abinpet-79.pdf>. Acesso em 08 de out. 2021.

DART. **Dart programming language**: Dart documentation, 2020. Disponível em: <https://dart.dev/>. Acesso em: 20 nov. 2021.

DEVELOPER. **Introdução Express/Node**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction. Acesso em: 28 nov. 2022.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. Dissertação de Doutorado - University of California, Irvine,

2000. Disponível em http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Acesso em junho de Acesso em: 28 nov. 2022.

JAROLEMAN, J. (1998). **A comparison of the reaction of children and adults: Focusing on pet loss and bereavement. Omega**, 37,133-150.

JORGE, Lucas Campos. **PROJETO E ARQUITETURA DE API REST PARA SISTEMA DE MONITORAMENTO DE REDES ÓPTICAS**. 2020. Disponível em: https://bdm.unb.br/bitstream/10483/27278/1/2020_LucasCamposJorge_tcc.pdf. Acesso em: 28 nov. 2022

KOVACS, Leandro. **O que é e para que serve o MongoDB?** 2021. Disponível em: https://www.terra.com.br/byte/o-que-e-e-para-que-serve-o-mongodb,599185a7bef2217ff870aad38ba13a4897y1dyq8.html. Acesso em: 06 dez. 2022.

LIMA, João Eurico Aguiar. **O que é UUID, para que serve e como funciona?** 2022. Disponível em: https://pt.quora.com/O-que-%C3%A9-UUID-para-que-serve-e-como-funciona. Acesso em: 06 dez. 2022.

LOPES, Sérgio, **A Web Mobile: Programe para um mundo de muitos dispositivos**.2013, Casa do Código.

MONGOBD. **Bancos de dados e coleções**. 2022. Disponível em: https://www.mongodb.com/docs/manual/core/databases-and-collections/. Acesso em: 06 dez. 2022.

NODEJS. **Node.js® is an open-source, cross-platform JavaScript runtime environment.**. 2022. Disponível em: https://nodejs.org//. Acesso em: 06 dez. 2022.

PESSÔA, Camila. **Node.JS: definição, características, vantagens e usos possíveis**. 2022. Disponível em: https://www.alura.com.br/artigos/node-js-definicao-caracteristicas-vantagens-usos. Acesso em: 28 nov. 2022.

SERPELL, J. A. (2011). **As perspectivas históricas e culturais das interações dos seres humanos com animais de estimação.** In: P. McCardle, S. McCune, J. A. Griffin, L. Esposito & L. S. Freund (Orgs). Os animais em nossa vida: família, comunidade e ambientes terapêuticos. Campinas, SP: Papirus.

SERPELL, J.A., 1993, "**Childhood Pet keeping and Humane Attitudes in Young Adulthood**", Animal Welfare, Vol.1, N. 2, p. 321-337

SILVA, Antonio Eudálio de Sousa da. **Análise comparativa entre os frameworks de desenvolvimento de aplicativos móveis multiplataforma. / Antonio Eudálio**

de Sousa da Silva. – 2021. Disponível em: <http://repositorio.ufc.br/bitstream/riufc/59037/1/2020_tcc_aedesdasilva.pdf>. Acesso em: 16 de Outubro de 2021.

SOUZA, Ivan de. **Entenda o que é Rest API e a importância dele para o site da sua empresa**. 2020. Disponível em: https://rockcontent.com/br/blog/rest-api/. Acesso em: 28 nov. 2022.

ZAMMETTI, Frank. **Flutter na prática**: melhore seu desenvolvimento mobile com o sdk open source mais recente do google. Pensilvânia: Novatec Editora, 2020.