UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MARCOS ROBERTO ALBRECHT

APLICATIVO MOBILE PARA ADOÇÃO DE PETS

TOLEDO

2022

MARCOS ROBERTO ALBRECHT

APLICATIVO MOBILE PARA ADOÇÃO DE PETS

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de tecnólogo em Sistemas para Internet daUniversidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Prof. Dr. Roberto Milton Scheffel.

TOLEDO

2022

APLICATIVO MOBILE PARA ADOÇÃO DE PETS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Sistemas para Internet da Universidade Tecnológica Federal do Paraná(UTFPR).

Data de aprovação: 24 de junho de 2022

Eduardo Pezutti Belato dos Santos
Mestre em Ciências da Computação e Matemática Computacional
Universidade Tecnológica Federal do Paraná

Vilson Luiz Dalle Mole
Doutor em Ciência da Computação
Universidade Tecnológica Federal do Paraná

Roberto Milton Scheffel
Doutor em Ciência da Computação
Universidade Tecnológica Federal do Paraná

TOLEDO

2022

AGRADECIMENTOS

Primeiramente, gostaria de agradecer à Deus.

Agradeço aos meus pais por todo o esforço investido na minha educação.

Agradeço à minha esposa que sempre esteve ao meu lado durante o meupercurso acadêmico.

Sou grato pela confiança depositada na minha proposta de projeto pelo meu professor Dr. Roberto Milton Scheffel, orientador do meu trabalho. Obrigado por me manter motivado durante todo o processo.

Por último, quero agradecer também à Universidade Tecnológica Federaldo Paraná e todo o seu corpo docente.

RESUMO

Albrecht, Marcos R. APLICATIVO MOBILE PARA ADOÇÃO DE PETS. 2022. 62f. Trabalho de Conclusão de Curso – Curso de Tecnologia em Sistemas para Internet, Universidade Tecnológica Federal do Paraná. Toledo, 2022.

O abandono de animais domésticos, especialmente cães e gatos, tem aumentado significativamente a população de animais nas ruas, tornando-se um desafio para o bem-estar dos animais e também para a saúde pública. Diversas ONGs e grupos de proteção animal, públicos ou privados, atuam no sentido de recolher, assistir e posteriormente encaminhar estes animais para adoção responsável. Neste sentido, este trabalho teve por objetivo desenvolver uma aplicação mobile para auxiliar essas ONGs e o público em geral, proporcionando maior visibilidade para os animais que se encontram para adoção. Para tanto, foi desenvolvida uma aplicação na plataforma *Android* juntamente com o kit de desenvolvimento React Native e uma API Rest, desenvolvida em Java com o framework Spring Boot.

Palavras chave: Adoção de animais, Android, React Native, JavaScript, Spring Boot, MongoDB.

ABSTRACT

Albrecht, Marcos R. MOBILE APPLICATION FOR PET ADOPTION. 2022. 62f. Completion of course work — Internet Systems Technology Course, Federal Technological University of Paraná. Toledo, 2022.

Abandonment of pets, especially dogs and cats, has significantly increased the population of animals on the streets becoming a challenge for animal welfare as well as public health. Various NGOs and animal protection groups, public or private, act to collect, assist and later forward these animals for responsible adoption. Thus this work aimed to develop a mobile application to help these NGOs and the general public, providing greater visibility for animals that are up for adoption. For that, an application was developed on the Android platform together with the React Native development kit and a Rest API, developed in Java with the Spring Boot framework.

Keywords: Animal Adoption, Android, React Native, JavaScript, Spring Boot, MongoDB.

LISTA DE FIGURAS

LISTA DE DIAGRAMAS

SUMÁRIO

INTRODUÇÃO

De acordo com um levantamento realizado pelo Instituto Pet Brasil em 2019, o Brasil possui a quarta maior população mundial de animais domésticos, com cerca de 140 milhões de animais, entre eles gatos, cachorros, aves, peixesentre outros (CENSO PET, 2019).

Em contrapartida, a Organização Mundial de Saúde, estima que o Brasil possui cerca de 30 milhões de animais abandonados, dentre eles 20 milhões de cachorros e 10 milhões de gatos (MOL, 2020).

Muitos desses animais em situação de rua já fizeram parte de uma família, mas foram abandonados por seus donos por questões sócio-econômicas, culturais ou religiosas (GARCIA, 2014). Coronato (2016) apontou que seis em cada dez brasileiros deixariam seu animal caso tivessem que se mudar de casa, sendo esse o principal motivo de abandono entre as pessoas que já tiveram um cão ou gato. Entre outros motivos ainda estão a falta de tempo, questões comportamentais e a chegada de um filho (GONÇALVES, 2021).

Um exemplo recente de motivos que levam os donos a abandonarem seus animais foi a disseminação da falsa informação de que cachorros eram transmissores do coronavírus, colaborando com a disseminação da doença Covid-19 (MOL, 2020).

Diante disso, o abandono de animais domésticos, sobretudo cães e gatos têm aumentado significativamente a população de animais nas ruas, tornando-se um desafio para o bem-estar dos animais e também para a saúde pública.

Muitas ONGs e instituições trabalham no sentido de conter o abandono, recolhendo animais abandonados e vítimas de maus-tratos, castrando para evitar a procriação e reintroduzindo os animais em famílias que possam prover uma vida digna a estes animais. O uso da Internet e da tecnologia, atualmente, é uma das principais ferramentas de divulgação de eventos de adoção de animais, colaborando também na localização de animais perdidos ou abandonados, auxiliando o trabalho das ONGs e de diversas pessoas que desejam doar, adotar ou encontrar um pet.

1.1 Objetivos

Nesta seção são apresentados o objetivo geral e na sequência os objetivos específicos.

1.1.1 Objetivo Geral

O objetivo desta proposta é desenvolver uma aplicação mobile para facilitar o contato entre instituições de proteção animal ou pessoas que possuem animais para adoção, e pessoas interessadas em adotá-los.

1.1.2 Objetivos Específicos

A partir da definição do objetivo geral, foram elencados os objetivos específicos para o desenvolvimento deste projeto. Tais objetivos são:

- Realizar o levantamento de requisitos funcionais e não funcionais para o desenvolvimento da aplicação;
- Executar a modelagem do sistema;
- Desenvolver uma aplicação que funcione como uma vitrine de animais disponíveis para adoção;
- Desenvolver uma plataforma que promova a comunicação ágil entre instituições de proteção animal ou usuários doadores de animais e os usuários candidatos à adoção;
- Explorar o desenvolvimento de aplicações servidoras empregando a linguagem Java e o framework Spring Boot;

EMBASAMENTO TEÓRICO

Este capítulo introduz acerca do abandono de animais domésticos, explanando o funcionamento das ONGs que trabalham nesta causa, e abordando como são utilizados alguns recursos como busca e tratamento dos animais de rua.

2.1 Justificativa

Nesta seção serão abordados os aspectos sociais que inspiraram a execução deste trabalho.

2.1.1 Animais de estimação

A relação entre homem e animais já acontece desde a pré-história, período em que os animais eram utilizados para defender o território em que o homem vivia e prestar auxílio na realização dos transportes (CAETANO, 2010). Hoje, os animais domésticos, representados em sua maioria pelas espécies felina e canina, são considerados por muitos como membros da família, sendo cada vez mais buscados como forma de companhia, oferecendo a seus donos além da amizade, uma série de benefícios físicos e psicológicos, como redução do estresse e depressão e aumento da autoestima (CARVALHO, 2016).

A Associação Americana de Medicina Veterinária trata a relação entre humanos e animais como “uma relação dinâmica e mutuamente benéfica entre pessoas e outros animais, influenciada pelos comportamentos essenciais para a saúde e bem-estar de ambos". Isso inclui as interações emocionais, psicológicas e físicas entre pessoas, demais animais e ambiente.” (FARACO, 2008).

No entanto, a relação homem-animal é estabelecida como uma via de mão dupla, em que ao mesmo tempo é extremamente benéfica para o ser humano, também demanda responsabilidades para com o animal. Um animal doméstico precisa de cuidados básicos para a sua sobrevivência, saúde e bem-estar. De acordo com Guerin (2009) o bem-estar animal está relacionado com o conceito das cinco liberdades: o animal não deve passar fome ou sede, nem ter

uma nutrição deficiente; ser livre de desconforto; ser livre de dor, doenças ou lesões; livre de medo e estresse; e livre para expressar seu comportamento normal.

2.1.2 Abandono de animais

De acordo com levantamento realizado pelo Instituto Pet Brasil (2019) a população de animais de estimação no Brasil é de aproximadamente 140 milhões, em que mais de 75% destes são cachorros e gatos e, segundo a Organização Mundial da Saúde (OMS), conforme pesquisa realizada em 2014, estima-se que o Brasil possui 30 milhões de animais abandonados (CENSO PET, 2019).

Infelizmente, muitos desses animais em situação de rua já fizeram parte de uma família e, por alguma razão, se tornaram vítimas do abandono. De acordo com Cardoso (2013), os principais motivos que levam ao abandono de animais são a inadequação às regras de condomínio, falta de tempo para cuidar do animal, gravidez do animal, latidos constantes (no caso dos cães), agressividade e a chegada de crianças na família.

Estes animais abandonados são submetidos a viver nas ruas, sofrendo diariamente com uma série de riscos como atropelamentos, agressões, doenças, brigas, envenenamentos e desnutrição (SANTOS, 2015). O poder público, na tentativa de minimizar os riscos para a população em geral e, principalmente, para os próprios animais, atua por meio dos centros de zoonoses e controles de doenças, que recolhem os animais das ruas, abrigando-os em canis e centros de recolhimento. Outra importante ação no acolhimento destes animais vem de organizações não governamentais, instituições de caridade e protetores independentes, que na maioria das vezes desempenham suas funções através do trabalho de voluntários. Seja por iniciativa pública ou privada, o objetivo desses órgãos de proteção é castrar os animais para diminuir sua proliferação nas ruas e disponibilizá-los para adoção, seja nos próprios centros ou feiras de animais (DE PAULA, 2012).

Mesmo reconhecendo os esforços das instituições de proteção animal, é preciso ter clareza de que a medida de maior importância no combate ao abandono é o investimento na educação e conscientização da sociedade e,

principalmente, a definição de regras mais claras para a punição das pessoas que praticam esses atos (DELABARY, 2012). O processo educativo deve ocorrer a fim de orientar os tutores de todas as responsabilidades perante o animal adotado, incluindo a capacidade de atender as necessidades básicas do pet como tratamento veterinário, espaço adequado e tempo para estar com o animal.

2.1.3 Instituições de Proteção aos Animais

Um levantamento realizado pelo Instituto Brasileiro de Geografia e Estatística (IBGE) apontou que havia, em 2010, 2.242 instituições especializadas em proteção animal ou meio ambiente no Brasil (FILHO, 2017). Muitas dessas instituições contam apenas com doações e trabalho de voluntários que se comprometem em receber doação de ração, levar ao veterinário para realizar castração, vacinação e divulgá-los em redes sociais para encontrar adotantes.

Com a popularização das redes sociais e os diferentes métodos de comunicação que as acompanham, surgiu também uma nova ferramenta facilitadora para que cães e gatos abandonados sejam exibidos para adoção. O instrumento principalmente utilizado pelas ONGs são as páginas do Facebook, que além de promoverem os animais para adoção, possibilitam avaliar os possíveis adotantes por meio de seus perfis pessoais nas redes sociais. No entanto, um grande inconveniente na utilização desse sistema é a falta de organização e a divulgação de informações, sendo anunciadas informalmente em mídias sociais, de forma incompleta e/ou imprecisa, dificultam a busca de quem procura um animal de perfil específico ou deseja ver fotos e comentários sobre ele, por exemplo (FILHO, 2017).

2.2 Fundamentação teórica

Nesta seção serão abordados os aspectos técnicos e métodos para desenvolvimento do sistema, como programação para dispositivos móveis, API Rest e banco de dados.

2.2.1 Desenvolvimento para dispositivos móveis

O uso de telefones celulares, smartphones, dispositivos móveis e aplicativos vem conquistando cada vez mais tempo e espaço na vida das pessoas. O Centro de Tecnologia de Informação Aplicada da Escola de Administração de Empresas de São Paulo da Fundação Getulio Vargas revela que há 424 milhões de dispositivos digitais, computador, *notebook*, *tablet* e *smartphone*, em uso no Brasil, e desses, 242 milhões são smartphones (FGV, 2022). É quase impossível se adaptar aos vários modelos de telefones celulares de hoje sem usar aplicativos com inúmeras funções de comunicação (VENTEU; PINTO, 2018).

Devido às diferenças de design e tecnologia, o desenvolvimento nativo do Android e do iOS é realizado em diferentes linguagens e ambientes, o que obriga quase todos os desenvolvedores que desejam atingir o maior número possível de usuários a desenvolver para essas duas plataformas. Outra possibilidade é utilizar uma plataforma de desenvolvimento híbrida, que traz benefícios em termos de reutilização e manutenção de código, mas carece de um design único em termos de cada plataforma e desempenho (FILHO, 2017).

2.2.3 Ferramentas utilizadas no desenvolvimento da aplicação

Nesta seção serão apresentadas as ferramentas a serem utilizadas. Para desenvolvimento da aplicação Android será utilizada a linguagem JavaScript aliada ao Framework React Native. Já para o desenvolvimento da API (Application Programming Interface ou Interface de Programação de Aplicação) Rest será utilizada a linguagem Java juntamente ao Framework Spring Boot.

2.2.2.1 Android

Android é uma plataforma de código aberto baseada em Linux, composta principalmente por um sistema operacional e *middleware*, uma estrutura de *software* que gerencia as funções de dispositivo e integra um conjunto de bibliotecas de API permitindo o desenvolvimento de aplicativos e interfaces de usuário, essencialmente desenvolvido para plataformas móveis, como smartphones e tablets. Seu surgimento permitiu o desenvolvimento de aplicativos que pudessem extrair o máximo proveito do uso do dispositivo.

Diferente do que acontece em outras plataformas, denominadas plataformas de proprietários, em que as aplicações nativas eram desenvolvidas e compiladas usando APIs diferentes daquelas que eram usadas nas aplicações desenvolvidas para essa plataforma, no Android tanto as aplicações nativas como as desenvolvidas para este utilizam as mesmas APIs. No mercado atual é possível encontrar dispositivos Android de diversos fabricantes com uma grande diversidade de funcionalidade e diversidade de hardwares (MEIER, 2009).

Apesar do Framework React Native funcionar com apenas um código nas plataformas Android e IOS, foi escolhido o Android devido a maior facilidade de acesso, uma vez que para desenvolver para iOS seria necessário ter um dispositivo com sistema operacional iOS.

2.2.2.2 JavaScript

> O JavaScript é uma linguagem de alto nível, dinâmica, interpretada e não tipada, conveniente para estilos de programação orientados a objetos e funcionais. A sintaxe de JavaScript é derivada da linguagem Java, das funções de primeira classe Scheme e da herança baseada em protótipos de Self (FLANAGAN, 2014).

Trata-se de uma linguagem de programação habitualmente usada na implementação de aplicações para a Web, mas que pode ser utilizada também no desenvolvimento de aplicações que funcionam fora do navegador, como no Node.JS.

A linguagem é multiparadigma, ou seja, suporta estilos de programação funcional, imperativo, orientado a objetos e orientado a eventos. JavaScript possibilita armazenar conteúdo em variáveis sem especificação de tipo, operar com strings e arrays e executar funções em resposta a ações realizadas pelo teclado e mouse, como rolar para cima e para baixo, cliques, teclas pressionadas etc. (SANTOS, 2021).

Possibilita a utilização, junto às Interfaces de Programação de Aplicativos (API), de vários tipos de códigos de construção para implementar recursos na aplicação, como por exemplo geolocalização e gráficos.

A execução do código em JavaScript é feita, como na maioria das linguagens de programação, de cima para baixo e, portanto, é preciso

ter cuidado com a ordem em que variáveis, importações e funções são adicionadosno código (SANTOS, 2021).

2.2.2.3 React Native

Lançado no ano de 2015 pelo Facebook©, o React Native é um *framework* para desenvolvimento *mobile* que compreende uma série de ferramentas que, em conjunto, possibilitam a criação de aplicativos móveis nativos para Android e iOS, utilizando ferramentas de front-end mais modernas e o desenvolvimento com base em JavaScript (SILVA E SOUSA, 2019; CABRAL, 2016).

O React Native invoca as APIs de renderização nativas em Objective-C, para iOS, ou Java para Android e, dessa forma, a aplicação é renderizada empregando componentes nativos da plataforma, diferente de outras estruturas para aplicações *mobile* que utilizam *webviews* para renderizar a aplicação (EISENMAN, 2016).

O desenvolvimento torna-se muito mais rápido, possuindo um reaproveitamento de código significativo que não existiria se fosse desenvolvido em Object-C para iOS e Java para Android (KUPKA, 2017). Ademais, o React Native conta com uma comunidade bastante colaborativa que torna mais fácil e rápida a forma de encontrar soluções em caso de problemas com desenvolvimento, com *plugins*, bibliotecas ou incompatibilidades, por exemplo. Por vezes, a comunidade disponibiliza até mesmo componentes prontos para utilização e, ainda que não sejam encontrados, um componente nativo pode ser criado apenas uma vez e, posteriormente, reutilizado (OLIVEIRA, 2018).

Possibilita a utilização de JavaScript; CSS Flexbox, deixa a tela visível em diversas resoluções e dispositivos diferentes (EIS, 2012); JSX (JavaScript XML), permite a utilização de tags HTML (Linguagem de Marcação de HiperTexto) e JavaScript, em um mesmo arquivo (MILFONT, 2017); e de diversos pacotes do NPM (Node Package Manager). Permite também, fazer *debug* na mesma *Integrated Development Environment* (IDE) utilizada para o desenvolvimento nativo com essas plataformas (CABRAL, 2016).

Como o desenvolvimento é baseado em JavaScript, para realizar a comunicação entre o setor JavaScript e o setor nativo da aplicação, existe uma ponte que é responsável por esse processo (CÂMARA, 2018).

> O *React Native* (RN) funciona a partir dessa ponte que permite a realização de chamadas nativas ao *JavaScript* e chamadas do *JavaScript* ao nativo. No Android é incluído uma *JavaScript* VM (VirtualMachine) (JavaScriptCore), no iOS a *JavaScriptCore* já é uma integração do sistema. O RN é um código JavaScript rodando em umamáquina virtual, controlado por uma interface de usuário nativa, o código gerado por JavaScript não é compilado ou convertido para a linguagem nativa do dispositivo. Tudo acontece através dessa ponte entre o JavaScript e o ambiente nativo do dispositivo (MONTEIRO, 2017).

A estrutura compreende tipos de sintaxes fáceis de entender e de reutilizar e suas formas se baseiam em *tags* HTML e tags específicas para diferentes tipos de dispositivos. Possui ainda, uma forma de descrever estilos inline, utilizando JavaScript, ao contrário dos demais *frameworks* que utilizam estilos separadamente. A partir disso, existem abordagens que fazem com que não seja necessário estilizar cada um dos itens, sendo possível criar um objeto de estilo e reutilizá-lo em diversos elementos (KUPKA, 2017).

Entre as principais conveniências da utilização desse *framework* estão: proporciona uma experiência do usuário mais fluida, uma vez que gera códigos nativos; os carregamentos e requisições são mais rápidos, já que não requer uma *webview* para intermediar esse processo; possui uma melhor integração entre funções do dispositivo, como câmera, gps, giroscópio, etc.; possui maior segurança em relação a aplicativos *web mobile* e uma performance geral, comparativamente, superior (PEDRASSANI, 2018).

Colares (2018) aponta a customização de *layouts* como uma vantagem do React Native, uma vez que é feita, inteiramente, por meio de um modelo unidimensional. Kupka (2017), por sua vez, descreve outras funcionalidades interessantes do *framework*, como: o *reload* automático após cada alteração, que aumenta a praticidade e a produtividade e faz com que o programa seja capaz de ficar rodando em desenvolvimento, e a cada mudança no código uma nova versão seja injetada na aplicação, levando menos de um segundo para atualizar; a possibilidade de depurar a aplicação via Google Chrome, como se fosse uma aplicação web padrão, facilitando o desenvolvimento; a possibilidade de mesclar códigos JavaScript com códigos nativos (Objective-C ou Java), caso haja necessidade de utilizar componentes prontos ou otimizar alguns aspectos da aplicação.

A comunidade JavaScript e React Native recomenda o uso de alguma estrutura com capacidade para gerenciar o estado, possibilitando a modelagem

de aplicativos que contenham um estado complexo, como a API Context, por exemplo, porque escrevendo em JavaScript, pode-se lançar erros em qualquer lugar da aplicação, o que não dá a devida segurança das linguagens tipadas estaticamente. Por esse motivo é extremamente importante tratar o estado de maneira eficaz e segura ao criar um aplicativo usando React Native (PEDRASSANI, 2018).

2.2.2.4 API

Para criar a comunicação entre cliente e servidor foi desenvolvido uma API (*Application Programming Interface*) utilizando a linguagem *Java,* juntamente ao *framework Spring Boot*, dessa maneira cliente e servidor poderão se comunicar sem a intervenção dos usuários. E para este projeto foi escolhida a *API Rest*.

2.2.2.5 API Rest

O design Rest (*Representation State Transfer*), conhecido como *API Rest*, possui uma excelente camada de flexibilidade que visa tirar proveito dos protocolos existentes. Ele é utilizado pelo protocolo HTTP (*Hypertext Transfer Protocol*) que é usado para API da Web, embora possa ser utilizado por muitos outros protocolos. Utilizar o protocolo HTTP significa que os desenvolvedores não precisam instalar *softwares* ou bibliotecas adicionais para ter grandes proveitos do design da API Rest. Além disso a API Rest pode ligar-se com muitos tipos de chamadas, retornar diversos formatos de dados e, até mesmo, alterar uma estrutura por meio de implementação hipermídia correta (TEIXEIRA, 2021).

Devido a liberdade e flexibilidade inerentes ao design da API Rest, ela permite que os desenvolvedores criem uma API que atenda a todas as necessidades e, também, às necessidades dos mais variados clientes, simultaneamente. O formato de mensagem Rest não se restringe apenas ao XML (*Extensible Markup Language*), mas também pode retornar JSON (*JavaScript Object Notation*), YAML (*YAML Ain't Markup Language*) ou qualquer outro formato a depender do que o cliente precisar (MULESOFT, 2021).

2.2.2.6 Java

Java é uma linguagem de programação orientada a objetos desenvolvida pela Sun Microsystems na década de 1990. Os programas escritos em Java são compilados em *bytecode*, que é o código interpretado pela máquina virtual, cujo intérprete está instalado nas principais plataformas de execução do mercado, incluindo os principais navegadores da Web. (ORACLE, 2017). E, para auxiliar no desenvolvimento da API em Java, será utilizado o *framework* Spring Boot.

2.2.2.7 Spring Boot

Spring Boot é uma plataforma que visa reutilizar as tecnologias já contidas no *Spring Framework* e melhorar a produtividade do desenvolvedor. Foi introduzido na versão Spring 4.0 e tornou-se necessário devido às dificuldades e o tempo necessário para iniciar o desenvolvimento de novos projetos.

A principal diferença está na forma como o código é configurado, organizado e na implementação da aplicação baseado em 4 princípios:

1. Fornece uma experiência de lançamento de projeto rápida e fácil.
2. Apresentar uma visão simples para configuração de projetos Spring, porém flexível o suficiente para ser facilmente alterada de acordo comas necessidades do projeto.
3. Fornece uma série de requisitos não funcionais pré-configurados para desenvolvedores, como métricas, segurança, acesso ao banco de dados, servidor/servlet de aplicativos incorporado, etc.
4. Não fornecer geração de código e minimizar a necessidade de arquivos XML para zero. Em suma, o Spring Boot pode ser entendido como fina camada de tecnologias que já estão estabelecidas no mercado.

O conceito de convenção de configuração é o grande motor por trás dos ganhos de produtividade do Spring Boot, o que significa que a maioria das configurações que um desenvolvedor tem que escrever no início de um projeto são sempre as mesmas, então o Spring Boot já está iniciando um novo projeto com elas. Configurações podem ser especificadas e editadas a qualquer momento (BIANCHI, 2015).

No desenvolvimento desta aplicação, foi criado um projeto Spring Boot utilizando o Maven, que é responsável por gerenciar o *build*. A partir disso, usando uma dependência do Spring, foi obtido tudo o que precisa para aumentar o desenvolvimento do servidor com Spring Data MongoDB para mapeamento de tabelas de banco de dados e um servidor de aplicativos que está embutido no pacote de aplicativos, neste caso o TomCat.

2.2.2.8 Banco de Dados NoSQL

O termo "NoSQL" apareceu pela primeira vez como um banco de dados não relacional de código aberto no final da década de 1990 (STROZZI, 2010). O banco de dados é liderado por Carlo Strozzi e armazena suas tabelas como arquivos ASCII (*American Standard Code for Information Interchange* ou Código Padrão Americano para o Intercâmbio de Informação). O nome NoSQL vem do fato de que o banco de dados não usa SQL (*Structured Query Language*, ou Linguagem de Consulta Estruturada) como linguagem de consulta. Em vez disso, ele é operado por meio de *scripts* de shell (SADALAGE; FOWLER, 2013).

NoSQL é uma abreviatura da frase "Not Only SQL" e mostra a tendência da comunidade em encontrar alternativas para a linguagem SQL. Johan Oskarsson, um desenvolvedor de software de Londres, organizou uma conferência em San Francisco em 2009 para aprender mais sobre armazenamento alternativo de dados. O termo "NoSQL" é o resultado dessa reunião. Oscarson não esperava que esse nome se tornasse uma tendência geral de tecnologia, porque na época ele queria apenas nomear a conferência. (SADALAGE; FOWLER, 2013).

2.2.2.9 MongoDB

MongoDB é um DBMS orientado a documentos baseado em conceitos NoSQL. A primeira versão do DBMS foi desenvolvida em 2007 por uma empresa chamada 10gen. Para armazenar informações na forma de documentos, o mongoDB utiliza o padrão de formato JSON, de forma que qualquer informação textual possa ser armazenada em sua base, mesmo que não esteja estruturada. É conhecido por seu banco de dados altamente escalável e normalmente é usado para soluções que exigem alto desempenho. BANKER (2011) disse que a flexibilidade e escalabilidade do MongoDB podem ser comprovadas quando usado como um SGBD (*Data Base Management System* ou Sistema de Gerenciamento de Banco de Dados) para sistemas Web, porque ele é projetado para atender às necessidades de alta resposta, crescimento contínuo do volume de dados e indexação de texto. Parte desse alto desempenho e flexibilidade do mongoDB pode ser atribuída aos seguintes recursos:

- Não utiliza transações em seu controle de dados, nem junções na recuperação de informações.
- Usa arquivos binários JSON padrão para armazenar dados por meio de chaves e valores.
- Eles têm uma arquitetura aberta, permite entrada dinâmica, armazenamento em cache e facilita a migração.
- Possui uma estrutura de armazenamento de arquivos baseada em GridFS, que é um sistema de arquivos que garante a persistência de grandes blocos de dados no disco e gerencia automaticamente a divisão desses blocos.
- A consulta retornada pela solicitação é simples e leve.
- Possui índice de texto e consulta dinâmica.

As características acima mostram que o MongoDB é uma opção muito relevante na escolha de um DBMS não relacional, especialmente quando o ambiente de banco de dados requer escalabilidade do aplicativo e desempenho de tempo de resposta (OLIVEIRA, 2017).

2.3 Requisitos do sistema

Nesta seção, são apresentados os requisitos funcionais e não funcionais que envolvem o aplicativo.

Sommerville (2018), diz que “os requisitos de um sistema são descrições dos serviços que o sistema deve prestar e as restrições a sua operação. Esses requisitos refletem as necessidades dos clientes de um sistema que atende a um determinado propósito […]”. Para o autor, os requisitos não são independentes, podendo gerar ou limitar outros requisitos e, ainda, não definem apenas as características ou serviços de um sistema, mas também as funcionalidades necessárias para que esses serviços e características funcionem conforme especificado.

Após um breve resumo das principais bibliotecas e tecnologias utilizadas no projeto, será apresentado o processo de desenvolvimento do aplicativo. As especificações do sistema serão explicadas, o processo de desenvolvimento defront-end e, em seguida, o desempenho dos serviços de construção de back-endserá demonstrado.

Os requisitos funcionais (RF) e não funcionais (RNF) do sistema são:

Tabela 1 - Requisitos funcionais e não funcionais

ID	Descrição
RF01	Realizar cadastro dos usuários. Cadastro para disponibilizar informações do usuário nas publicações.
RF02	Feed de publicações de adoção. O aplicativo irá conter um feed depublicações de animais para adoção.
RF03	Feed de interação entre os usuários. Onde poderão ser postados fotos dos animais adotados ou que foram encontrados e interação entre os usuários.
RF04	Filtro por localização no feed do RF02. Nesta opção o usuário poderá selecionar uma localização por DDD ou por cidade.

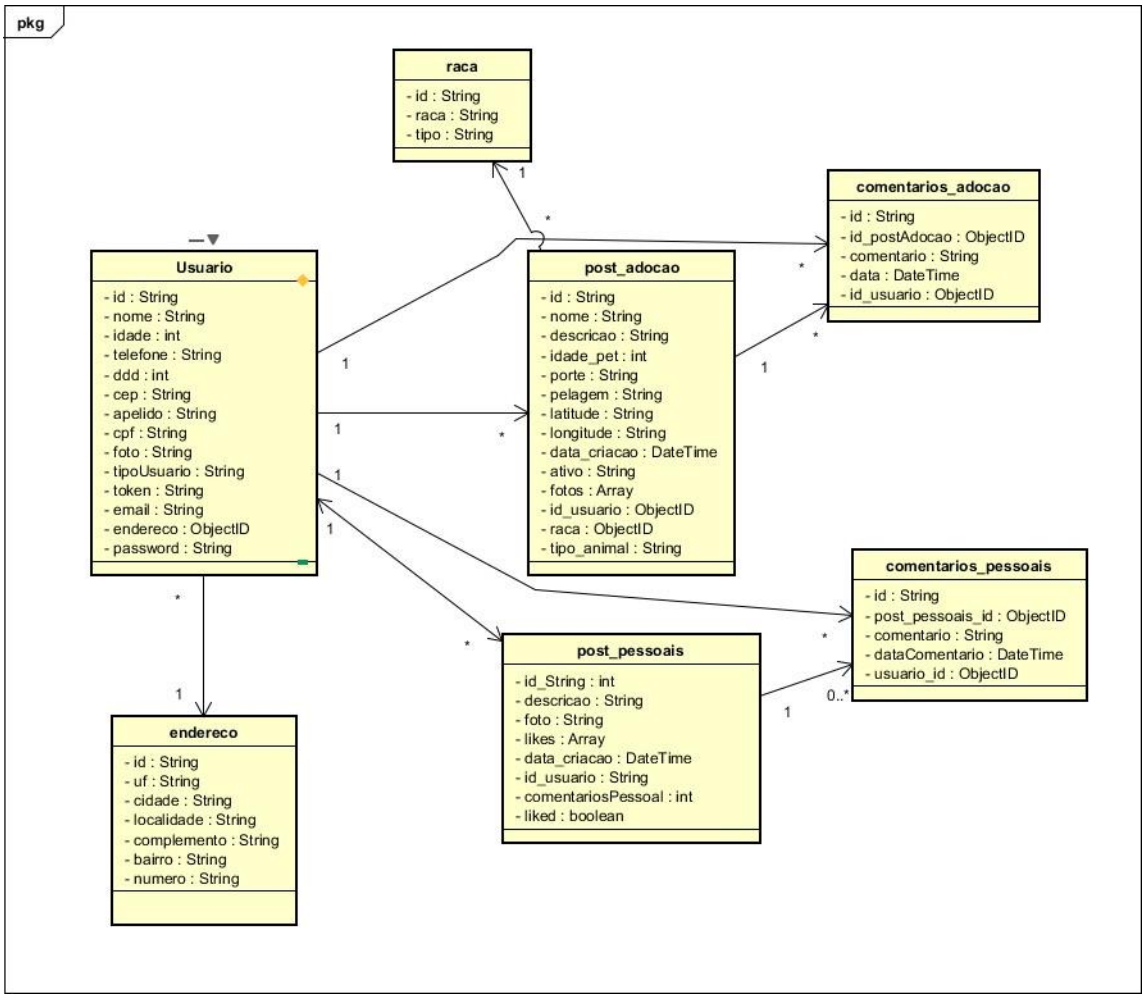
RF05	Manter publicação para doação. Essa opção o usuário poderá realizar o cadastro de uma publicação de doação.
RF06	Manter publicação de experiências com animais. Essa opção o usuário poderá realizar o cadastro de experiências e convivências com seus animais.
RF07	Carregar fotos nas publicações. Ao realizar o cadastro de uma publicação do RF02 ou RF03, o usuário poderá carregar fotos do celular ou tirar fotos com a câmera do celular.
RF08	Poder comentar e curtir as publicações do RF03.
RF09	Possibilidade dos usuários comentarem na publicação do RF02.
RF10	Possibilidade dos usuários comentarem na publicação do RF03.
RF11	Exibir dados para contato ao detalhar uma publicação para adoção.
RF12	Conter um feed das publicações de animais para adoção.
RF13	O usuário deve estar logado para curtir uma publicação de posts de pessoais.
RF14	O usuário deve estar logado para realizar o RF02 E RF03.
RNF01	As senhas dos usuários devem ser criptografadas ao salvar no banco.
RNF02	O aplicativo deve ser escrito utilizando o framework React Native.
RNF03	As operações do backend devem ser consumidas via serviço rest.
RNF04	O back-and deve ser feito na linguagem java.

Fonte: Autoria própria.

A partir dos requisitos, foram desenvolvidos os diagramas de casos de uso, sequência e classe. Esses diagramas documentam os fluxos e entidades envolvidas nos processos exercidos pelos usuários.

Em relação a modelagem do banco de dados, a Figura 3 mostra o diagrama de classe da aplicação.

Diagrama 1 - Diagrama de classe



Fonte: Autoria própria

2.4 Casos de uso

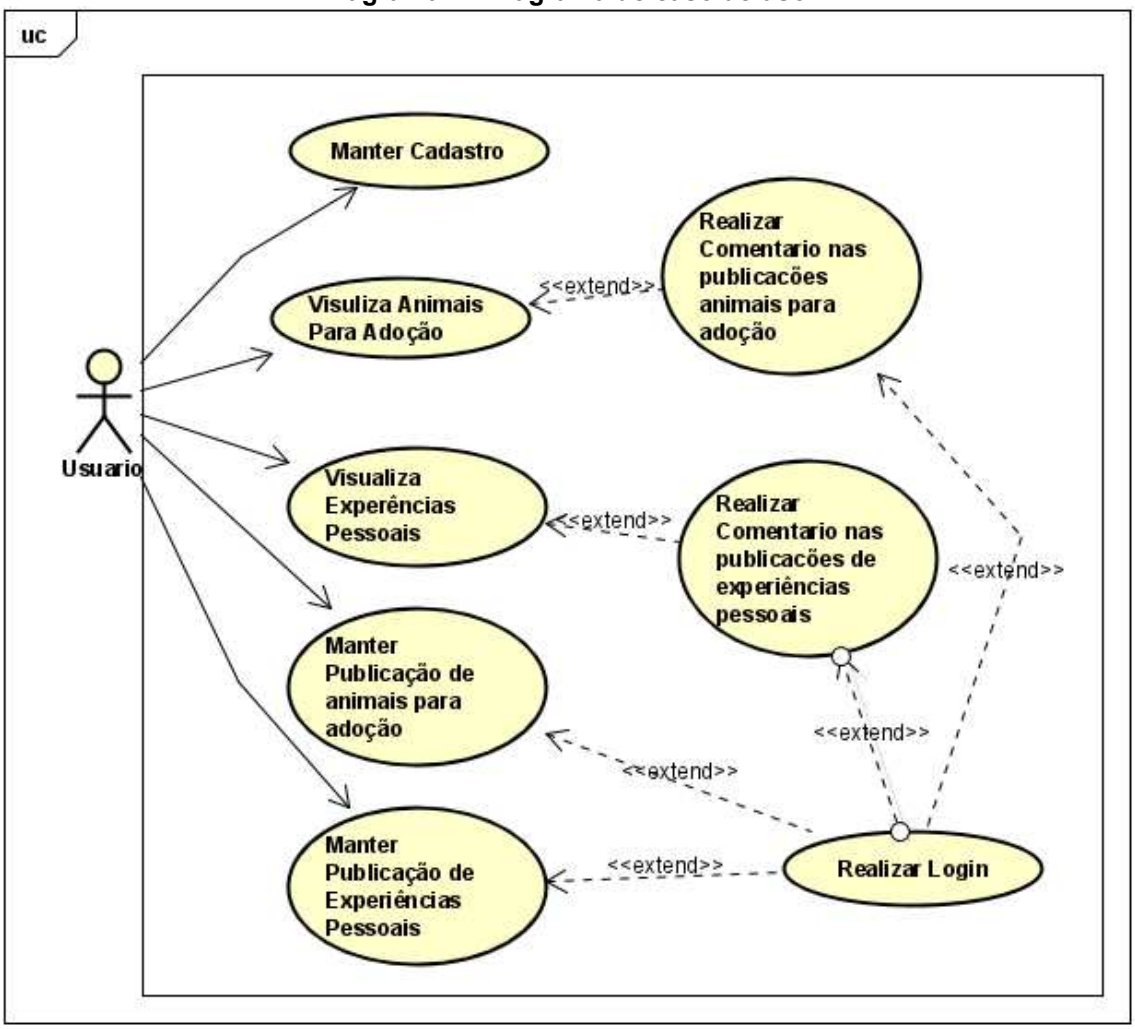
Um caso de uso é uma forma de especificar as funcionalidades de um software, ou seja, o que se pode fazer no sistema (ROCHA e MARTINS Jr., 2021). BOOCH, RUMBAUCH e JACOBSON (2005, p 227) explicam que, “um caso de uso especifica o comportamento de um sistema ou de parte de um sistema e é uma descrição de um conjunto de sequência de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator”.

Os casos de uso são criados a fim de facilitar o entendimento, para que os desenvolvedores e os usuários finais cheguem a uma compreensão comum, sem que haja algo efetivamente implementado (BOOCH, RUMBAUCH e

JACOBSON, 2005).

O diagrama de caso de uso a seguir, mostra o processo de funcionamento do aplicativo, em que os usuários podem acessar o aplicativo sem precisar realizar o login, poderá visualizar as postagens para adoção e experiências pessoais, a partir do momento que o usuário quiser interagir com essas publicações, como cadastrar e editar, bem como realizar comentário, será preciso criar uma conta e efetuar o login no aplicativo.

Diagrama 2 - Diagrama de caso de uso



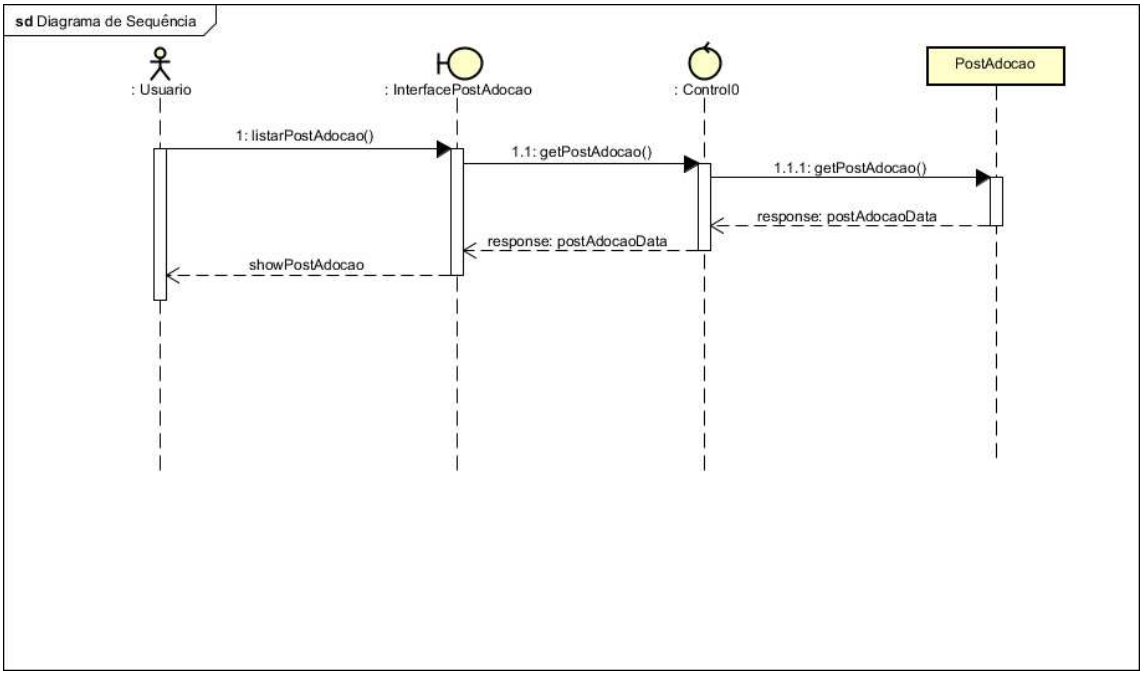
Fonte: Autoria própria

O usuário poderá acessar o aplicativo sem que seja necessário prévio cadastro, podendo visualizar o *feed* de postagens para adoção, detalhamentos dos dados e postagens pessoais. Mas, para poder cadastrar as postagens será necessário efetuar o cadastro no sistema (RF14), dessa

forma será liberado para qualquer usuário cadastrar e editar suas próprias postagens pessoais e adoções, tanto como curtir e comentar.

Visto o fluxo geral da aplicação, ao abrir o aplicativo a primeira tela que o usuário terá acesso será o feed de animais para doação. O processo de funcionamento está apresentado no diagrama de sequência a seguir.

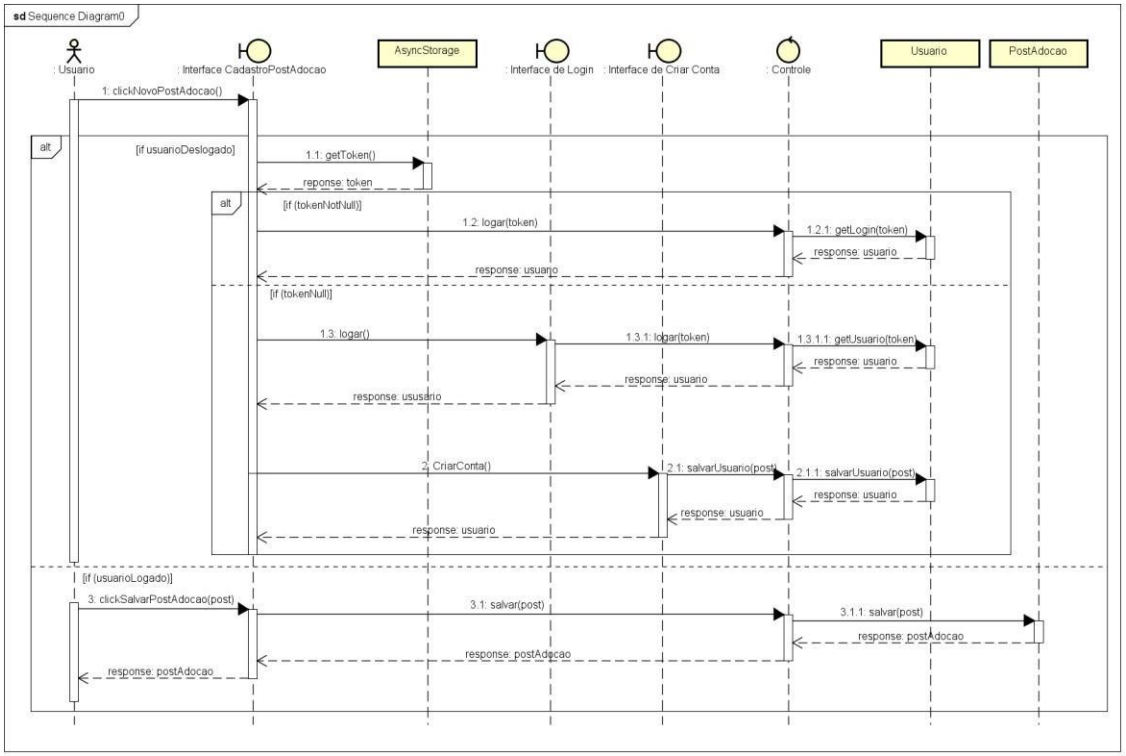
Diagrama 3 - Sequência listar postagem adoção



Fonte: Autoria própria

Para pode efetuar o cadastro de uma postagem para doação de animais (RF05), o usuário deve possuir um cadastro no sistema e efetuar login (RF14), usuários logados podem realizar cadastrar postagens para adoção e postagens pessoais, além de poder realizar comentários nas postagens, caso não tenha cadastro, ao acessar a página de cadastro, será verificado se o usuário está logado, caso não esteja, será aberto uma tela para logar ou realizar cadastro. Esse fluxo é exibido na imagem abaixo contendo o diagrama de sequência.

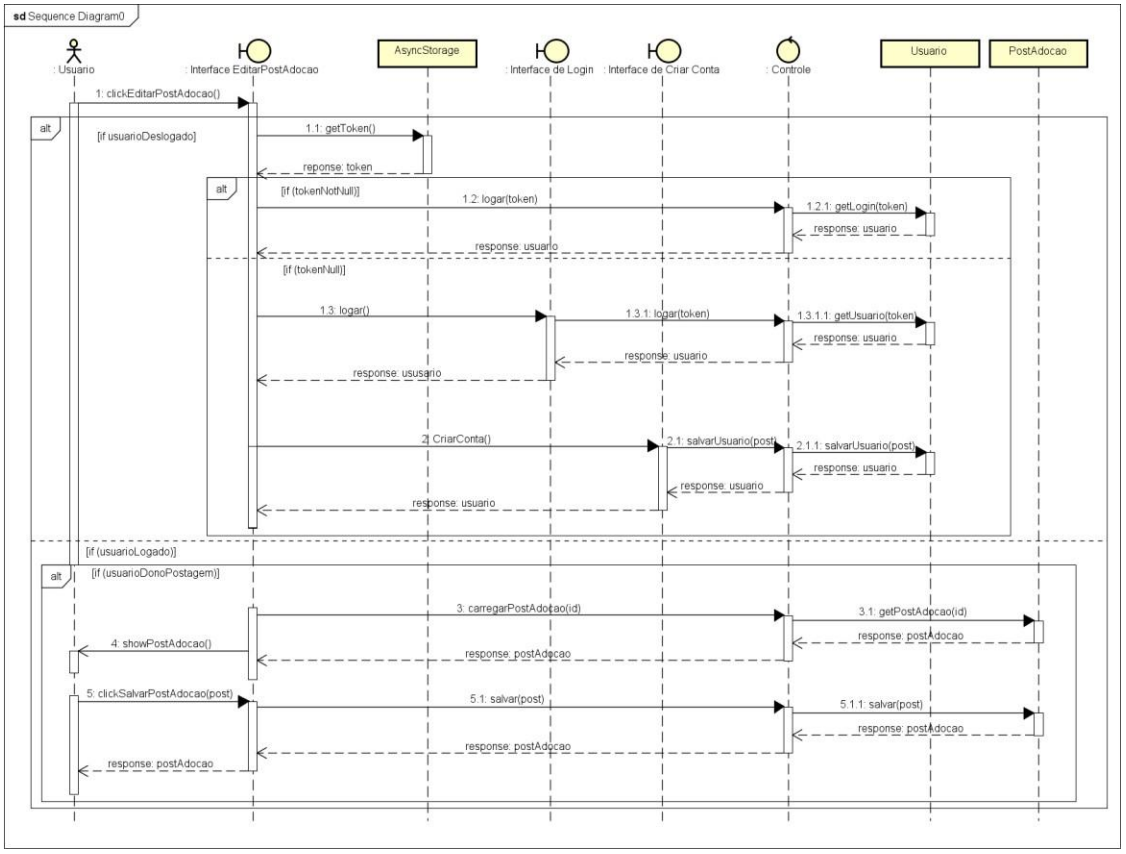
Diagrama 4 - Sequência cadastro postagem adoção



Fonte: Autoria própria

Existe também a possibilidade de o usuário editar suas postagens de animais para doação (RF05). Para isso, somente o usuário criador da postagem poderá editar e poderá fazer isso apenas se estiver logado no sistema (RF14), o processo de edição de uma postagem se dá pelo fluxo do diagrama 5 a seguir.

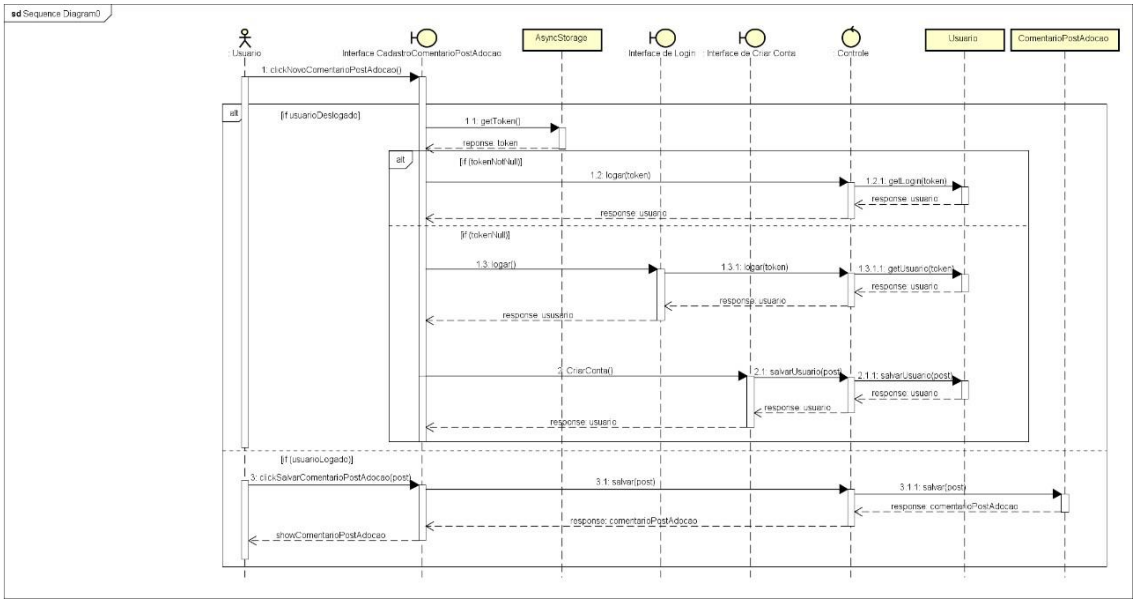
Diagrama 5 - Sequência edição postagem adoção



Fonte: Autoria própria

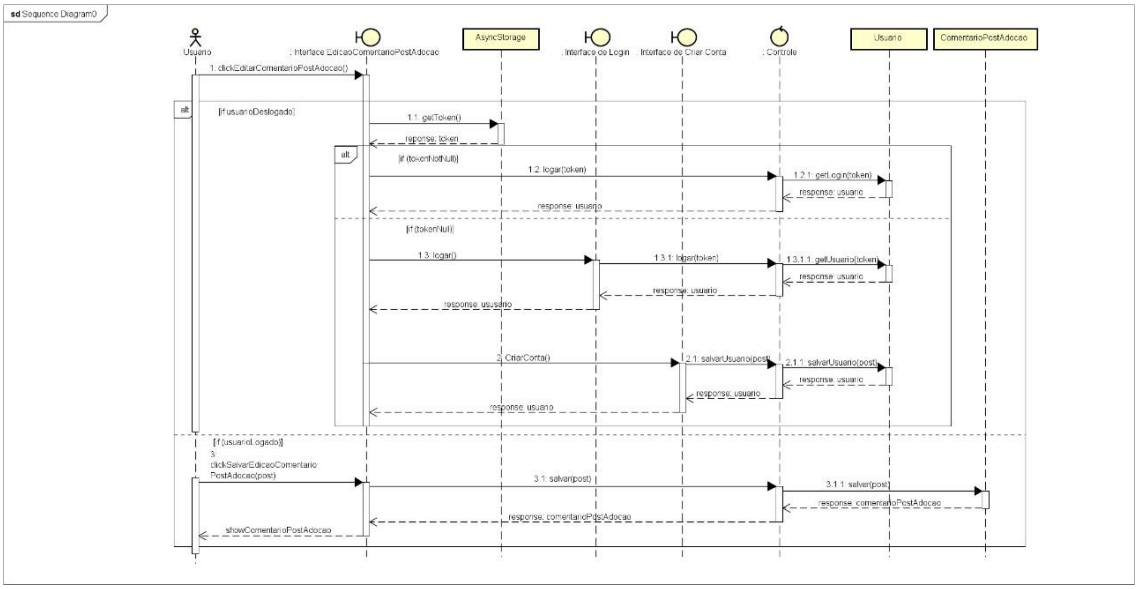
É possível ainda comentar nas postagens de animais para doação (RF09), qualquer usuário poderá comentar em qualquer postagem desde que logado no sistema (RF14). Uma vez logado poderá editar ou excluir o próprio comentário conformeapresentado nos diagramas 6 e diagrama 7 a seguir.

Diagrama 6 - Sequência cadastrar comentário postagem adoção



Fonte: Autoria própria

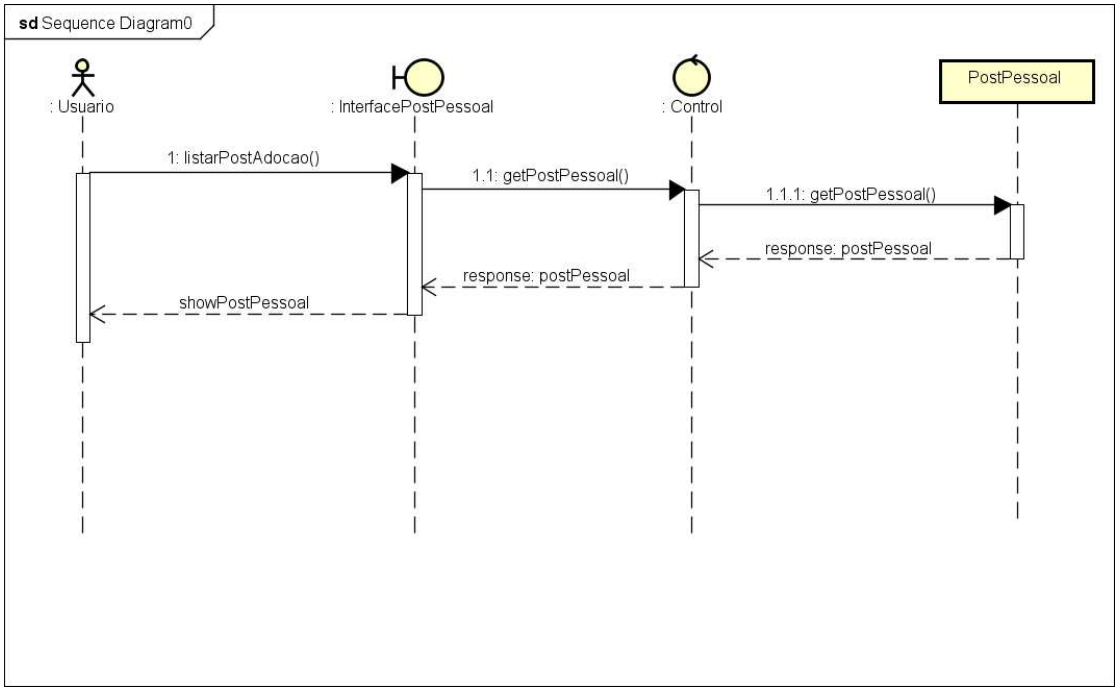
Diagrama 7 - Sequência edição comentário postagem adoção



Fonte: Autoria própria

Outra funcionalidade do aplicativo, é a possibilidade de realizar postagens de experiências pessoais (RF05), sendo necessário estar logado no sistema para realizar o cadastro (RF14). Este fluxo é análogo ao de postagens de animais paradoação, esse fluxo se dá conforme o diagrama 8.

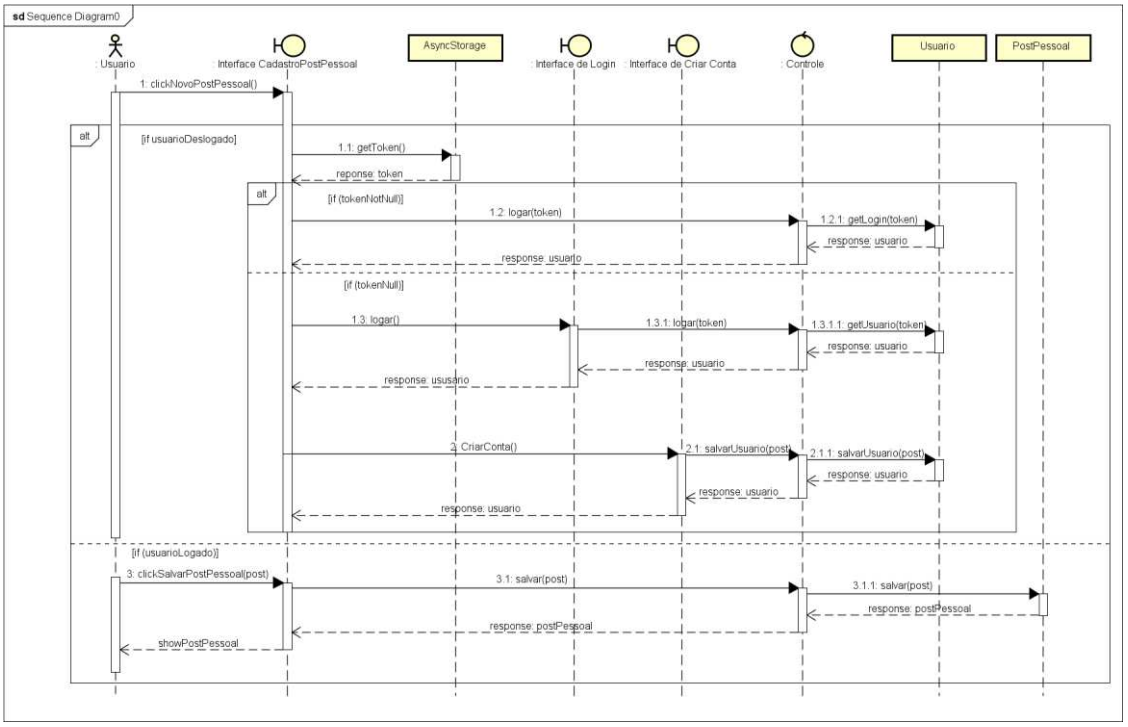
Diagrama 8 - Sequência listar postagem pessoal



Fonte: Autoria própria

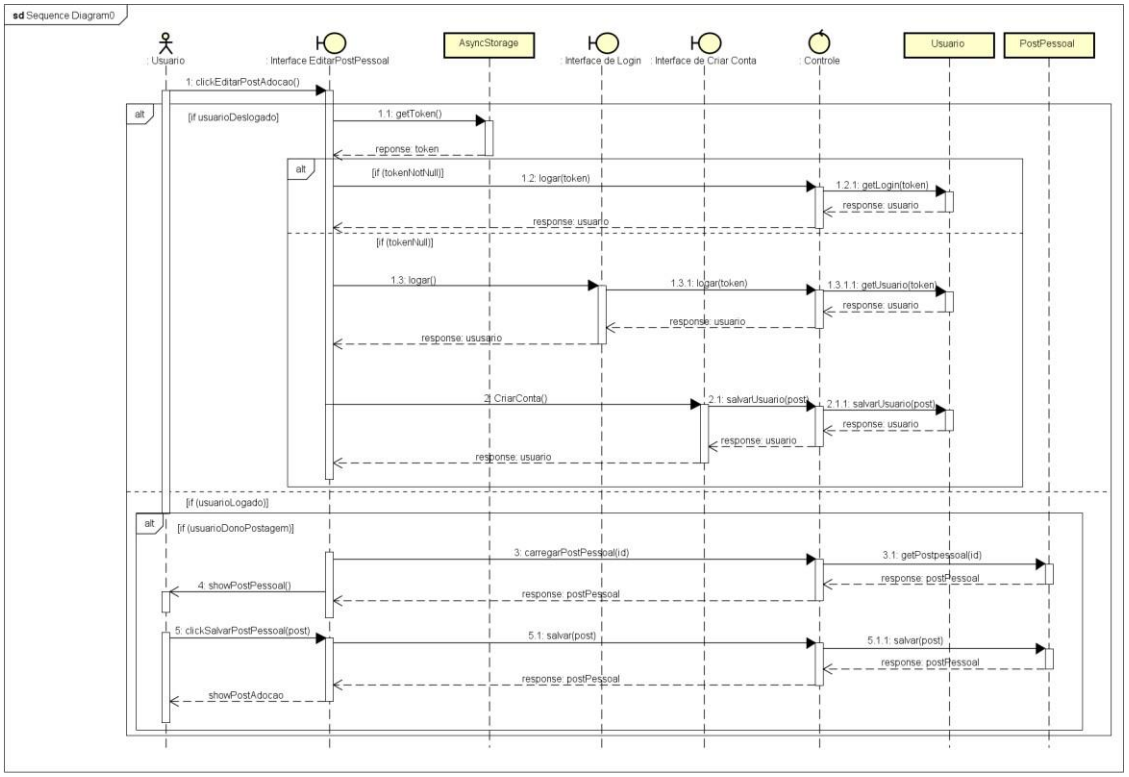
Nas postagens pessoais também é possível cadastrar e editar (RF06), desde que o usuário esteja logado no sistema (RF14). Também é possível comentar na postagem bem como curtir. O processo de cadastro e edição de postagens pessoais é representado nos diagramas 9 e 10.

Diagrama 9 - Sequência cadastrar postagem pessoal



Fonte: Autoria própria

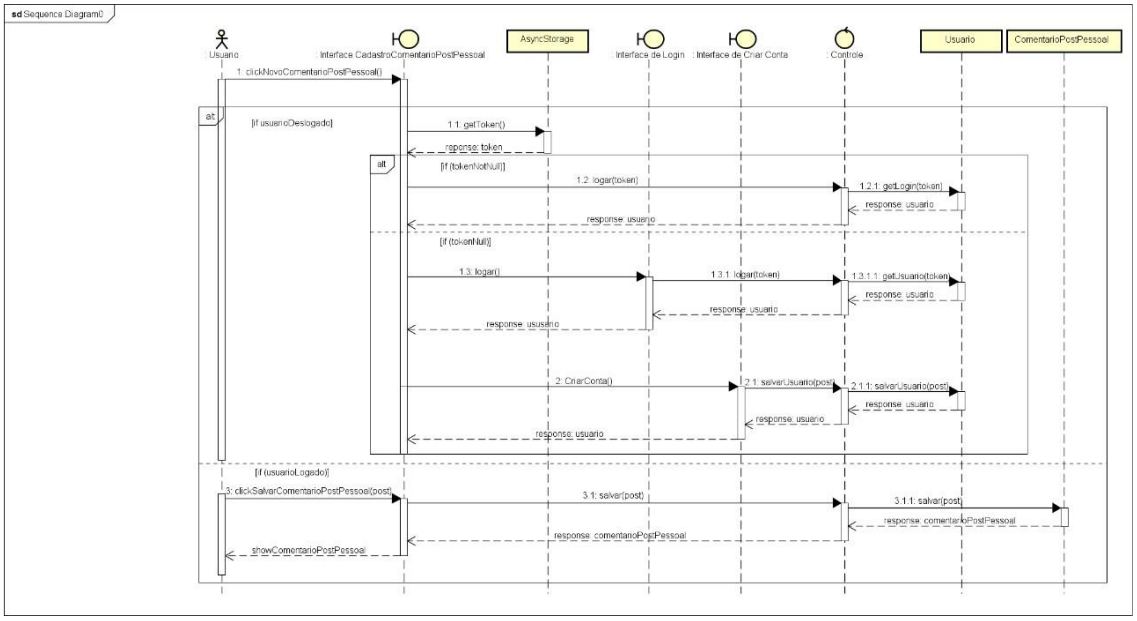
Diagrama 10 - Sequência editar postagem pessoal



Fonte: Autoria própria

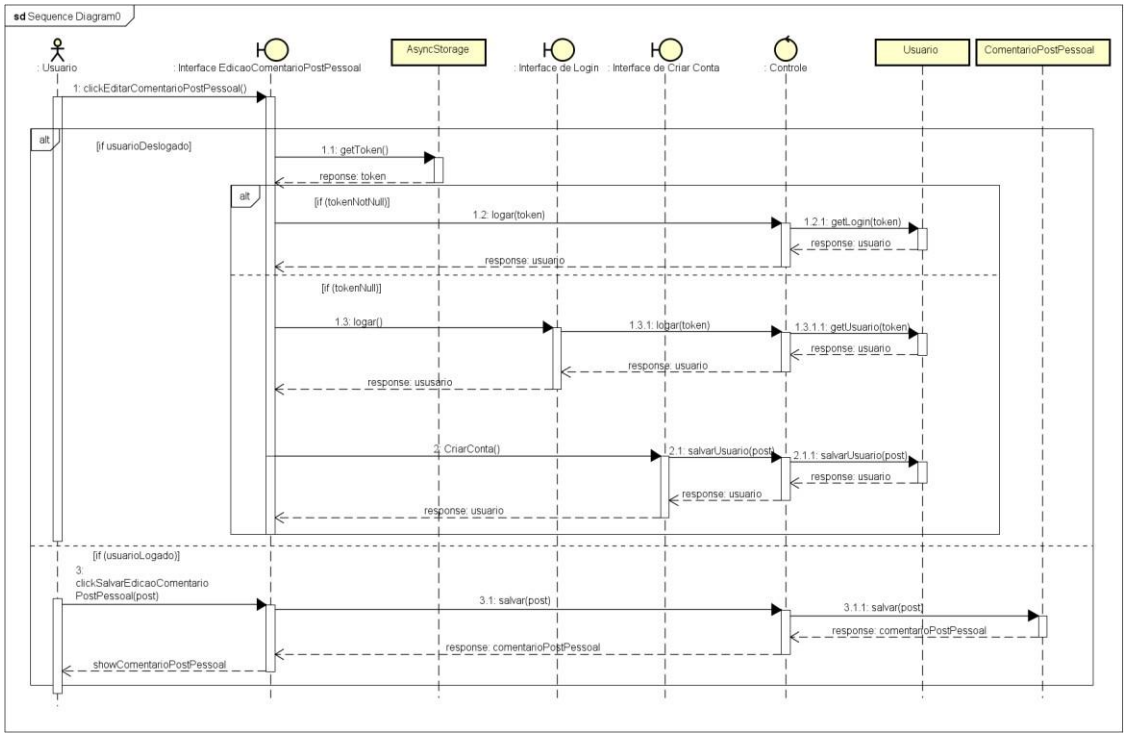
Ainda nas postagens pessoais, também é possível cadastrar e editar comentários (RF10), desde que o usuário esteja logado no sistema, semelhante aos comentários das postagens de doação. Os diagramas de sequência de cadastro e edição dos comentários das postagens pessoais são representados nos diagramas 11 e 12 a seguir.

Diagrama 11 - Sequência cadastrar comentário postagem pessoal



Fonte: Autoria própria

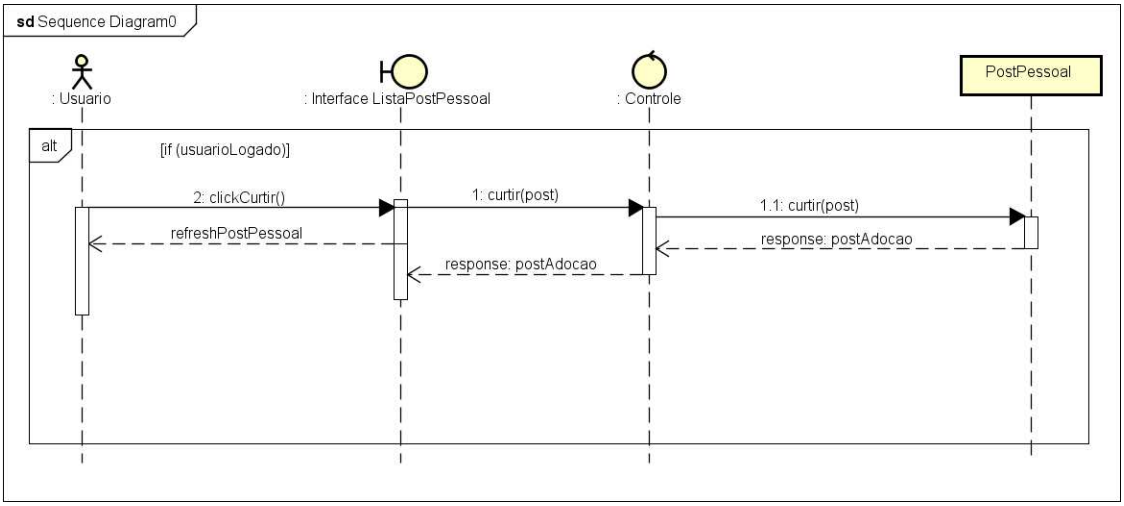
Diagrama 12 - Sequência editar comentário postagem pessoal



Fonte: Autoria própria

Nas postagens pessoais existe também a opção de curtir a postagem (RF08), para gerar um engajamento entre os usuários, da mesma forma como nos comentários, é necessário estar logado no sistema para poder curtir um post, o fluxo do diagrama de sequência é como exibido na imagem abaixo.

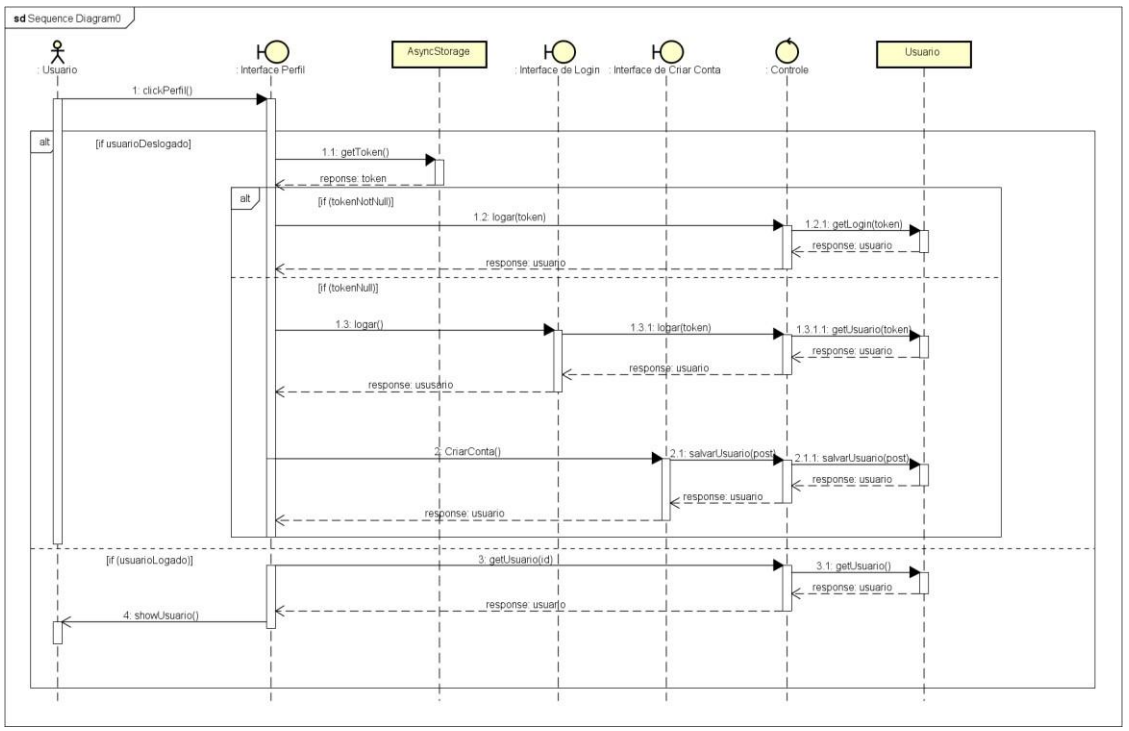
Diagrama 13 - Sequência curtir postagem pessoal



Fonte: Autoria própria

Por fim, o sistema permite criar ou editar um perfil de usuário (RF01), este é necessário quando o usuário quiser realizar cadastros e comentários no aplicativo. O diagrama 14 mostra como é feito o processo de listagem de perfil do usuário.

Diagrama 14 - Sequência visualizar perfil



Fonte: Autoria própria

3 DESENVOLVIMENTO

Para o desenvolvimento deste projeto foram estudadas algumas importantes tecnologias e foram criados, localmente, dois projetos, de acordo com as tecnologias relatadas neste documento. Para o método de desenvolvimento desses projetos, foi dividido em dois passos principais.

A primeira etapa é a construção de uma web API, contendo as classes e funções indispensáveis para a construção da camada model e controller. Ademais, esta camada inicial irá conter todos os métodos para as ações CRUD (*Create, Read, Update e Delete*) e ambas foram testadas com a ferramenta Postman para validar todos os endpoints.

A segunda etapa se deu no começo do desenvolvimento do aplicativo nativo, capaz de executar determinadas chamadas à web API. As partes foram chamadas de back-end e front-end, respectivamente.

3.1 Back-end

O back-end é baseado em uma web API que é a responsável pelas duas camadas da arquitetura MVC, a *model* e a *controller*. Com essas duas camadas sendo as principais da API, foram desenvolvidas classes auxiliares capazes de efetuar o roteamento de requisições e criptografia de senhas. O back-end foi construído de maneira simples, possuindo uma fácil manutenção e refatoração. O projeto em back-end, que é totalmente desacoplado do front-end, foi desenvolvido em Java com o framework Spring Boot. Modelado com o formato de API, quando uma rota é chamada, são feitas as instâncias de serviço necessárias para a consulta ao banco de dados.

A seguir serão mostradas as configurações necessárias para a criação da aplicação webservice, desenvolvida com Spring Boot, Mavem e MongoDB. Após a criação do projeto é necessário adicionar no arquivo pom.xml as dependências necessárias para o Maven construir o projeto. Em seguida é necessário especificar na figura 1 application.properties, os parâmetros de conexão para as credencias do MongoDB. A figura 1 apresenta o código para essas configurações.

Figura 1 - Arquivo application.properties

```
# server
server.port=8085

#mongodb
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=bd-tcc

```

Fonte: Autoria própria

É necessário configurar a classe principal (nesse caso, a classe ApiTccApplication), que será o ponto de partida do projeto Spring Boot. O método main() usa o Spring Boot SpringApplication() para iniciar o serviço, portanto, é necessário a anotação @SpringBootAplication. A figura 2 apresenta o código dessas configurações.

Figura 2 - Classe ApiTccApplication.java

```
@EnableMongoRepositories(basePackageClasses = UsuarioRepository.class)
@SpringBootApplication(exclude = {SecurityAutoConfiguration.class})
public class ApiTccApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(ApiTccApplication.class, args);
    }

    @Bean
    public PasswordEncoder getPasswordEncoder(){
        BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
        return encoder;
    }

}
```

Fonte: Autoria Própria.

Com o projeto configurado, podem ser criadas as classes de mapeamento das entidades de banco. Neste caso, é necessária a anotação *@Document* para saber qual entidade está sendo referenciada, a anotação @id é utilizada para informar ao spring-data qual campo/atributo de uma entidade estará relacionado à chave primária da respectiva collection no banco de dados, e em seguida

declarado os atributos conforme os nomes das colunas que serão salvas no banco de dados. Também foram criados os métodos *Getter* e *Setter* para os atributos da classe. A figura 3 apresenta a implementação da classe Usuario.java.

Figura 3 - Classe Usuario.java

```
16  @Document(collection = "usuario")
17  public class Usuario {
18
19      @Id
20      private String id;
21      private String nome;
22      private int idade;
23      private String telefone;
24      private int ddd;
25      private String cep;
26      private String apelido;
27      private String cpf;
28      private String foto;
29      private LocalDateTime data_cadastro = LocalDateTime.now();
30      private String tipo_usuario;
31      private String token;
32      @Field(name = "email")
33      private String email;
34      private String password;
35      @DBRef
36      private Endereco endereco;
```

Fonte: Autoria própria

O próximo passo é a criação da interface de acesso aos dados. Para isto, é necessário fazer a herança da interface MongoRepository, definindo também qual entidade está sendo gerenciada e o tipo do identificador dela. Nesse caso, o atributo da classe definido com a anotação *@Service*, que faz anotações de classes na camada de serviço e a anotação *@Query* serve para realizar uma busca personalizada nos dados do banco de dados, como apresentado no código da figura 4.

Figura 4 - Classe UsuarioRepository.java

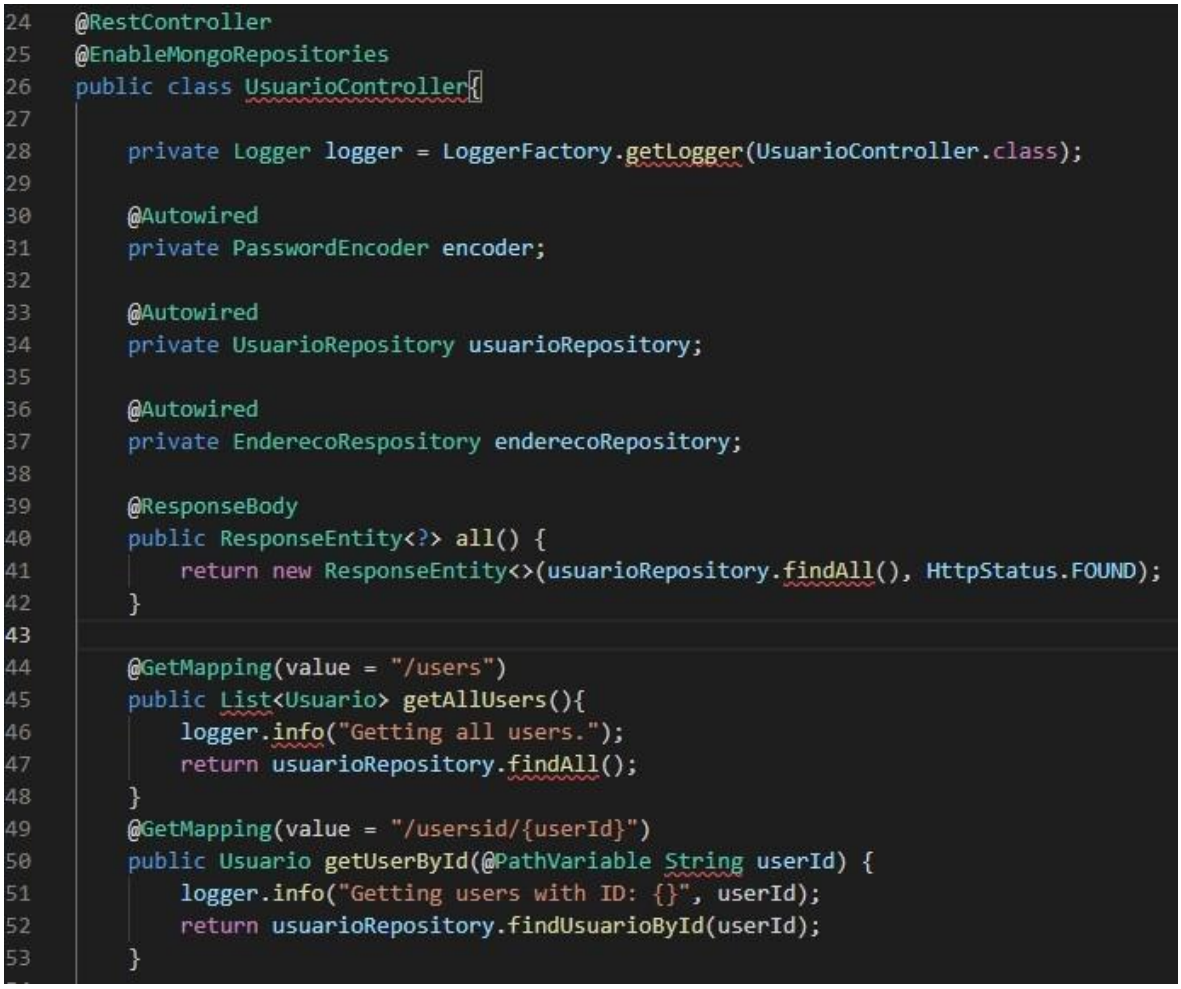
```
12  @Service
13  public interface UsuarioRepository extends MongoRepository<Usuario, String> {
14      Usuario findUsuarioById(String Id);
15
16      @Query("{'nome': {$regex: ?0 }})")
17      List<Usuario> findUsuarioByNome(String nome);
18
19      Usuario findUsuarioByEmail(String nome);
20
21
22      //Usuario createUser(Usuario user);
23  }
```

Fonte: Autoria própria

Para finalizar as implementações de gerenciamento de manipulação das entidades do banco de dados, é criada a classe controladora, que definirá as *Uniform Resource Locator* (URL) que serão consumidas na aplicação React Native. Neste momento, é necessário fazer uso da anotação @RestController que define o retorno REST dos serviços implementados.

Outra anotação necessária é a @GetMapping, que define o prefixo de todas as requisições realizadas pelo cliente. Para finalizar, é necessário fazer a injeção de dependência da interface repository, que é quem disponibiliza os métodos de manipulação dos registros conforme possível chamada da API. A figura 5 apresenta a classe UsuarioController.

Figura 5 - Classe UsuarioController.java

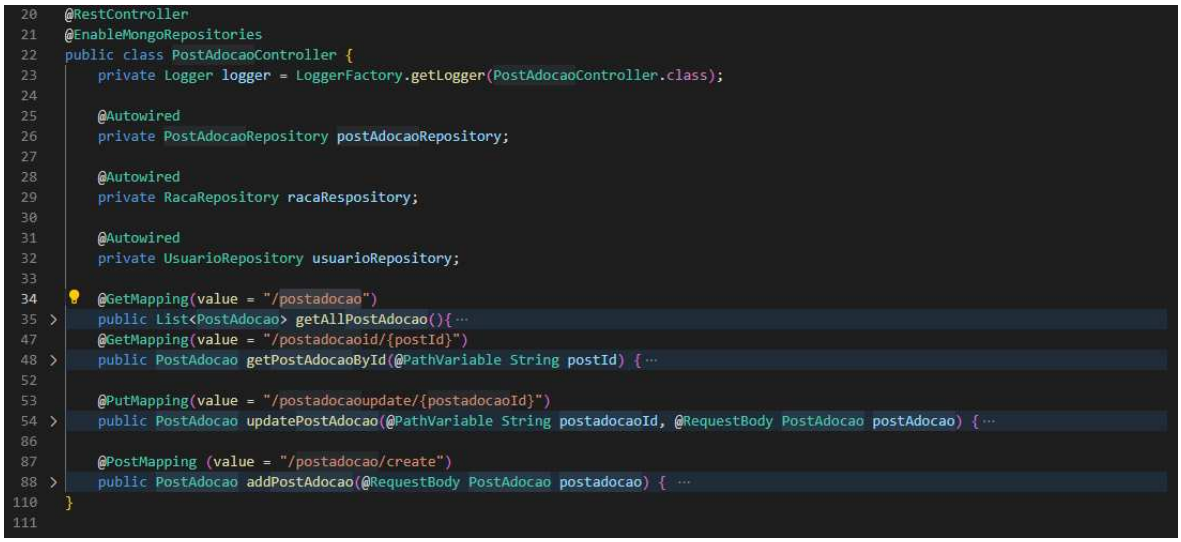


```
24  @RestController
25  @EnableMongoRepositories
26  public class UsuarioController{
27
28      private Logger logger = LoggerFactory.getLogger(UsuarioController.class);
29
30      @Autowired
31      private PasswordEncoder encoder;
32
33      @Autowired
34      private UsuarioRepository usuarioRepository;
35
36      @Autowired
37      private EnderecoRespository enderecoRepository;
38
39      @ResponseBody
40      public ResponseEntity<?> all() {
41          return new ResponseEntity<>(usuarioRepository.findAll(), HttpStatus.FOUND);
42      }
43
44      @GetMapping(value = "/users")
45      public List<Usuario> getAllUsers(){
46          logger.info("Getting all users.");
47          return usuarioRepository.findAll();
48      }
49      @GetMapping(value = "/usersid/{userId}")
50      public Usuario getUserById(@PathVariable String userId) {
51          logger.info("Getting users with ID: {}", userId);
52          return usuarioRepository.findUsuarioById(userId);
53      }
```

Fonte: Autoria própria

A *API* possui diversos *endpoints* de leitura, criação, alteração e delete em cada uma das classes *controller*, seguiu-se um padrão para todas. Os *endpoints* possuem o prefixo com o nome da classe e retorno no formato JSON, como mostrado na figura 6 abaixo, nas linhas 34, 47, 53 e 87, o prefixo do *endpoint* começa com postadocao que seria o nome da classe seguido da ação, podendo ser *update, create* ou *id* quando for uma pesquisa por determinada postagem. O *endpoint* da linha 34 retorna todas as postagens de animais para adoção, o endpoint da linha 47 obtem retorno das postagens de um determinado *id*, o *endpoint* da linha 53 atualiza as postagem de um *id* passado como referência, e o *endpoint* da linha 87 cria uma nova postagem para adoção passando as informaçoes em JSON. As demais classes *controller* seguem este mesmo padrão conforme as figuras de 7 a 12.

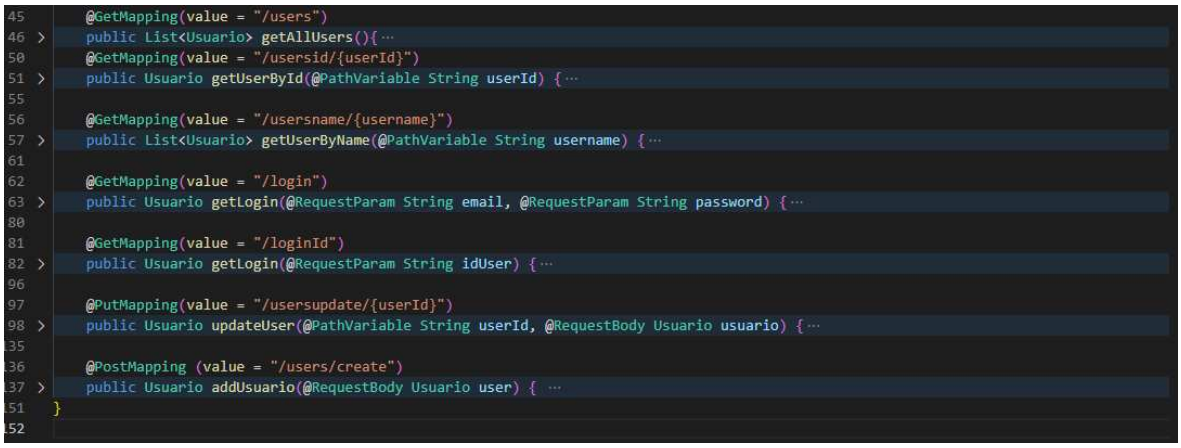
Figura 6 - Endpoints da classe PostAdocao



```
@RestController
@EnableMongoRepositories
public class PostAdocaoController {
    private Logger logger = LoggerFactory.getLogger(PostAdocaoController.class);

    @Autowired
    private PostAdocaoRepository postAdocaoRepository;

    @Autowired
    private RacaRepository racaRespository;

    @Autowired
    private UsuarioRepository usuarioRepository;

    @GetMapping(value = "/postadocao")
    public List<PostAdocao> getAllPostAdocao(){···
    @GetMapping(value = "/postadocaoid/{postId}")
    public PostAdocao getPostAdocaoById(@PathVariable String postId) {···

    @PutMapping(value = "/postadocaoupdate/{postadocaoId}")
    public PostAdocao updatePostAdocao(@PathVariable String postadocaoId, @RequestBody PostAdocao postAdocao) {···

    @PostMapping (value = "/postadocao/create")
    public PostAdocao addPostAdocao(@RequestBody PostAdocao postadocao) { ···
}
```

Fonte: Autoria própria

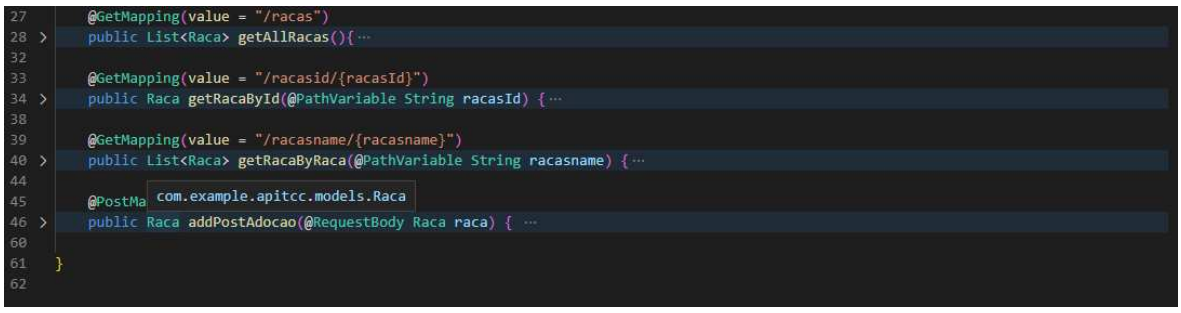
Figura 7 – Endpoints da classe Usuario



```
    @GetMapping(value = "/users")
    public List<Usuario> getAllUsers(){···
    @GetMapping(value = "/usersid/{userId}")
    public Usuario getUserById(@PathVariable String userId) {···

    @GetMapping(value = "/usersname/{username}")
    public List<Usuario> getUserByName(@PathVariable String username) {···

    @GetMapping(value = "/login")
    public Usuario getLogin(@RequestParam String email, @RequestParam String password) {···

    @GetMapping(value = "/loginId")
    public Usuario getLogin(@RequestParam String idUser) {···

    @PutMapping(value = "/usersupdate/{userId}")
    public Usuario updateUser(@PathVariable String userId, @RequestBody Usuario usuario) {···

    @PostMapping (value = "/users/create")
    public Usuario addUsuario(@RequestBody Usuario user) { ···
}
```

Fonte: Autoria própria

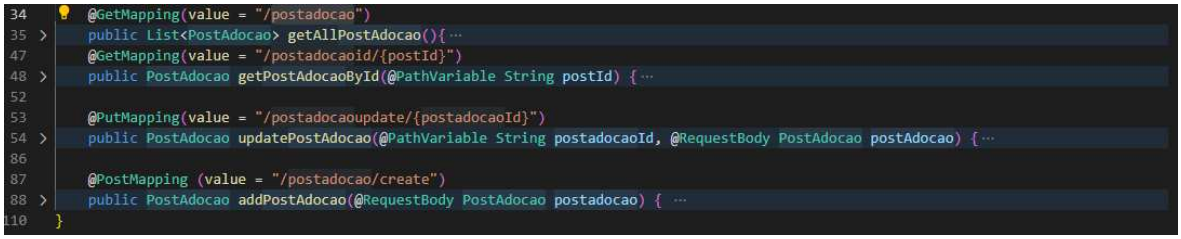
Figura 8 - Endpoints da classe Raca



```
    @GetMapping(value = "/racas")
    public List<Raca> getAllRacas(){···

    @GetMapping(value = "/racasid/{racasId}")
    public Raca getRacaById(@PathVariable String racasId) {···

    @GetMapping(value = "/racasname/{racasname}")
    public List<Raca> getRacaByRaca(@PathVariable String racasname) {···

    @PostMa  com.example.apitcc.models.Raca
    public Raca addPostAdocao(@RequestBody Raca raca) { ···

}
```

Fonte: Autoria própria

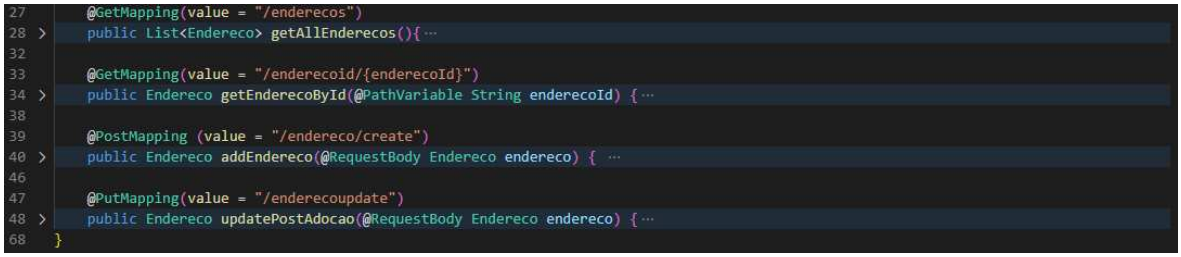
Figura 9 – Endpoints da classe PostAdocao



```
34    @GetMapping(value = "/postadocao")
35    public List<PostAdocao> getAllPostAdocao(){···
47    @GetMapping(value = "/postadocaoid/{postId}")
48    public PostAdocao getPostAdocaoById(@PathVariable String postId) {···

53    @PutMapping(value = "/postadocaoupdate/{postadocaoId}")
54    public PostAdocao updatePostAdocao(@PathVariable String postadocaoId, @RequestBody PostAdocao postAdocao) {···

87    @PostMapping (value = "/postadocao/create")
88    public PostAdocao addPostAdocao(@RequestBody PostAdocao postadocao) {  ···
110   }
```

Fonte: Autoria própria

Figura 10 – Endpoints da classe Endereco



```
27    @GetMapping(value = "/enderecos")
28    public List<Endereco> getAllEnderecos(){···

33    @GetMapping(value = "/enderecoid/{enderecoId}")
34    public Endereco getEnderecoById(@PathVariable String enderecoId) {···

39    @PostMapping (value = "/endereco/create")
40    public Endereco addEndereco(@RequestBody Endereco endereco) {  ···

47    @PutMapping(value = "/enderecoupdate")
48    public Endereco updatePostAdocao(@RequestBody Endereco endereco) {···
68    }
```

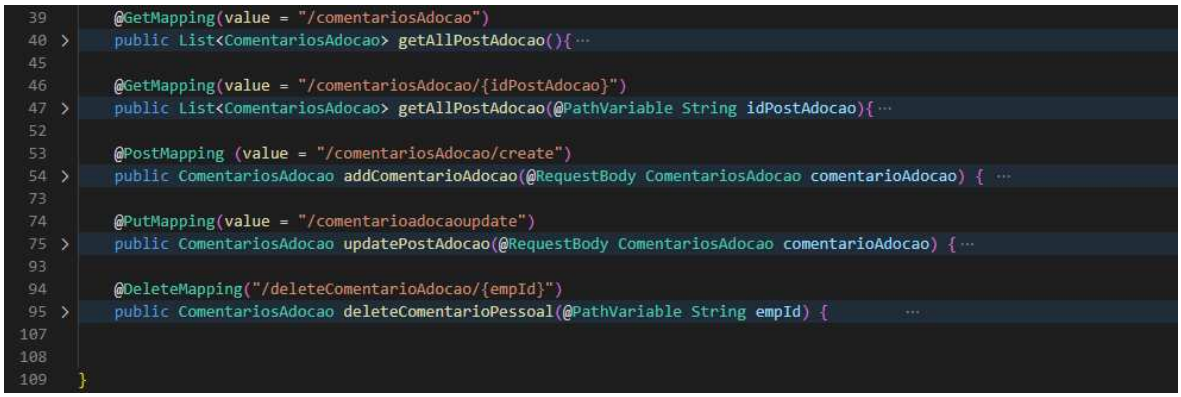
Fonte: Autoria própria

Figura 11 – Endpoints da classe ComentariosPostPessoal

```
40    @GetMapping(value = "/comentariosPessoal")
41    public List<ComentariosPessoal> getAllCommentPessoal(){···

47    @GetMapping(value = "/comentariosPessoal/{idPostPessoal}")
48    public List<ComentariosPessoal> getAllPostAdocao(@PathVariable String idPostPessoal){···

54    @PostMapping (value = "/comentariosPessoal/create")
55    public ComentariosPessoal addComentarioAdocao(@RequestBody ComentariosPessoal comentarioPessoal) {  ···

75    @PutMapping(value = "/comentariopessoalupdate")
76    public ComentariosPessoal updatePostAdocao(@RequestBody ComentariosPessoal comentarioPessoal) {···

95    @DeleteMapping("/deleteComentarioPessoal/{empId}")
96    public ComentariosPessoal deleteComentarioPessoal(@PathVariable String empId) {          ···
108
```

Fonte: Autoria própria

Figura 12 – Endpoints da classe ComentariosPostAdocao



```
39    @GetMapping(value = "/comentariosAdocao")
40    public List<ComentariosAdocao> getAllPostAdocao(){···

46    @GetMapping(value = "/comentariosAdocao/{idPostAdocao}")
47    public List<ComentariosAdocao> getAllPostAdocao(@PathVariable String idPostAdocao){···

53    @PostMapping (value = "/comentariosAdocao/create")
54    public ComentariosAdocao addComentarioAdocao(@RequestBody ComentariosAdocao comentarioAdocao) {  ···

74    @PutMapping(value = "/comentarioadocaoupdate")
75    public ComentariosAdocao updatePostAdocao(@RequestBody ComentariosAdocao comentarioAdocao) {···

94    @DeleteMapping("/deleteComentarioAdocao/{empId}")
95    public ComentariosAdocao deleteComentarioPessoal(@PathVariable String empId) {          ···
107
108
109   }
```

Fonte: Autoria própria

3.2 Front-end

Para realizar o desenvolvimento da aplicação front-end, foi utilizado o framework React Native, que é utilizado para o desenvolvimento de aplicações para dispositivos móveis. O React Native permite desenvolver na linguagem JavaScript ou TypeScript e, para este projeto, foi escolhido o JavaScript devido a familiaridade e conhecimento com a linguagem. O aplicativo foi dividido em pequenos componentes a fim de deixar o desenvolvimento mais dinâmico e possuir um maior reaproveitamento de código.

Primeiramente foi criada uma classe inicial que contém uma série de métodos que são executados após e durante a abertura do aplicativo. Essa classe é denominada App.js. Cada Hook feito em React Native deve ser exportada como um componente, para se exportar um Hook deve-se ter uma forma de renderizar aquele componente, então, para a classe App.js, renderizou-se em JSX, um componente de navegação de telas, conforme mostrado na figura a seguir.

Figura 13 - Estrutura da classe App.js

```
221 function MyTabs(){
222     return(
223         <Tab.Navigator screenOptions={screenOptionsBottonTabs}>
224             <Tab.Screen name='Adocão' component={AdocaoTabStack}
225             options={{tabBarIcon:({color, size}) => (<IconFeather name="github" size={25} color='#f4511e'/>)   }}  />
226             <Tab.Screen name='Blog' component={BlogTabStack}
227             options={{  tabBarIcon:({color, size}) => (<IconFeather name="smile" size={25} color='#f4511e'/>)  }} />
228             <Tab.Screen name='Perfil' component={PerfilTabStack}
229             options={{  tabBarIcon:({color, size}) => (<IconFeather name="user" size={25} color='#f4511e'/>)  }}/>
230         </Tab.Navigator>
231
232     )
233 }
234
235 export default props => {
236     return(
237         <UsersProvider>
238             <NavigationContainer>
239                 <MyTabs />
240             </NavigationContainer>
241         </UsersProvider>
242     )
243 }
244
```

Fonte: Autoria própria

Observa-se que na linha 236 da figura 13, é o momento em que a classe é renderizada a partir do método return(), e então é criado em formado JSX uma chamada a outro componente de uma classe capaz de iniciar o processo de navegação, encontrado na linha 239. Esse componente foi criado dentro da mesma classe App.js, nesta foram exportadas as funções representadas pelas

3 telas principais do aplicativo, e dentro de cada função, ela retorna todas as telas referentes a cada tela principal do aplicativo, utilizando uma dependência do React Native, chamada React Navigation. A dependência faz com que todas as telas iniciadas a partir dessa classe consigam navegar entre si, disponibilizando um fluxo de navegação simples através de camadas. A figura 14 exibida logo abaixo, mostra as telas de navegação do menu Perfil do aplicativo,somente elas poderão ser navegadas quando o usuário estiver neste menu.

Figura 14 - Função PerfilTabStack da classe App.js

```
187 function PerfilTabStack(){
188     return(
189         <StackAdocao.Navigator screenOptions={screenOptions}>
190 
191             <StackAdocao.Screen
192                 name='PerfilLogado'
193                 component={PerfilLogado}
194                 options={{
195                     title: "Perfil"
196                 }}
197             />
198             <StackAdocao.Screen
199                 name='PerfilEntrar'
200                 component={PerfilEntrar}
201                 options={{
202                     title: "Entrar"
203                 }}
204             />
205             <StackAdocao.Screen
206                 name='PerfilCadastrar'
207                 component={PerfilCadastrar}
208                 options={{
209                     title: "Cadastrar"
210                 }}
211             />
212         </StackAdocao.Navigator>
213     )
214 }
```

Fonte: Autoria própria

Para as demais telas do aplicativo, sempre que necessário, foi utilizada uma estrutura padrão orientada a eventos para a requisição de dados, para tratar dos eventos, existem os serviços. As requisições são realizadas via protocolo HTTPS. A biblioteca responsável por fazer esse procedimento é o serviço do Axios, que faz requisições assíncronas para a API. Previamente foi configurado uma classe api.js, contendo a chamada da biblioteca axios, retornando o baseURL conforme a linha 4 da figura 15, a base URL é o endereço ip no qual a API está localizada.

Figura 15 - Classe api.js

```
1   import axios from 'axios';
2
3   const api = axios.create({
4       baseURL: 'http://10.93.4.71:8085/'
5   });
6
7   export default api;
```

Fonte: Autoria própria

A figura 16 mostra como é feita a requisição para buscar as postagens de animais para doação.

Figura 16 - Requisição da biblioteca axios

```
19  useEffect(() => {
20      api.get("postadocao")
21        .then((response) => {
22            response.data.forEach(element => {
23              dispatch({
24                  type: 'createPostAdocao',
25                  payload: element,
26              })
27            }),
28            setCarregando(false)
29        })
30        .catch((err) => {
31          console.warn("ops! ocorreu um erro" + err);
32          console.log('items',response.data)
33        });
34
35        AsyncStorage.getItem("TOKEN").then((token) => {
36          console.log('token', token)
37          logarComToken(token)
38
39        })
40        .catch((err) => {
41          console.warn("ops! ocorreu um erro com o toekn" + err);
42
43        });
44
45    }, []);
```

Fonte: Autoria própria

Como é possível visualizar na linha 19 da figura 16, é utilizado o método useEffect importado da biblioteca react, ele é um hook de efeito, que permite

executar efeitos colaterais em componentes funcionais. Ou seja, a primeira coisa que é feita ao carregar essa tela, é executando o método useEffect, e neste caso será feito a chamada HTTPS para a API no endpoint/postadocao. Essa requisição irá retornar um JSON com os dados dessa rota.

Os dados de retorno dessa requisição serão armazenados no estado do aplicativo, como é possível visualizar na linha 22 a 27 da figura 16, existe uma estrutura de repetição for percorrendo os dados e armazenando no estado do aplicativo, neste projeto quem faz esse serviço é o gerenciador de estado Context API, com ele podemos compartilhar os dados com toda a aplicação. Para que o estado seja acessível em toda a aplicação, ele deve estar envolvido em todas as telas, e como neste projeto é utilizado o React Navigation, basta apenas envolver esse componente com a chama da classe StateContext.js, conforme a figura 13, linha 237. A figura 17 abaixo, ilustra o trecho de retorno da classe StateContext.js contendo o estado atual dos dados.

Figura 17 - Classe StateContext.js

```
189  export const UsersProvider = props => {
190      const [postAdocao, setPostAdocao] = useState([]);
191
192      function reducer(state, action){
193          const fn = actions[action.type]
194          return fn ? fn(state, action) : state
195      }
196
197      const [state, dispatch] = useReducer(reducer, initialState)
198
199      return(
200          <UsersContext.Provider value={{state, dispatch}} >
201              {props.children}
202          </UsersContext.Provider>
203      )
204  }
205
```

Fonte: Autoria própria

Para poder ter acesso aos dados do estado, é necessário declarar a classe StateContext.js em nosso código, conforme linha 5 da figura 18, e a partir disso poderá acessar os dados do estado conforme a linha 16 da figura 18.

Figura 18 - Declaração do estado StateContext.js

```
import { getActionFromState } from '@react-navigation/native';
import React, { useContext, useEffect, useState } from 'react';
import { View, Text, FlatList, Alert, TouchableOpacity, Image, ActivityIndicator } from 'react-native';
import { ListItem, Avatar, Button, Icon } from 'react-native-elements';
import UsersContext from '../../context/StateContext';
import api from '../../services/api';
import style from './styles'
import FeatherIcon from 'react-native-vector-icons/Feather';
import containerPadrao from '../../styles';
import AsyncStorage from "@react-native-async-storage/async-storage";

export default ({route, navigation}) => {
    //console.warn('props',props)
    const [postAdocao, setPostAdocao] = useState(route.params ? route.params : {});
    const [carregando, setCarregando] = useState(true);
    const {state, dispatch} = useContext(UsersContext);
```

Fonte: Autoria própria

Agora todo os dados da aplicação armazenados no estado estarão disponíveis, e então podem ser utilizados para exibir ao usuário. Para carregar as postagens de animais para doação, foi utilizado o componente FlatList, conforme figura 19 da biblioteca React Native, este por sua vez monta um array com as informações contidas no estado da aplicação.

Figura 19 - Componente FlatList

```
194     return(
195
196
197         <View style={containerPadrao.ContainerPadrao}>
198             {carregando &&
199                 <ActivityIndicator color="#fff" size={25} />
200             }
201             {!carregando &&
202                 <>
203                 <FlatList
204                     keyExtractor={({id},index)=>id}
205                     data={state.postAdocao}
206                     //renderItem={getUserItem}
207                     renderItem={getUserItem}
208                     extraData={state.postAdocao}
209                     //<Text key={item._id}>{item.nome},{item.idade}</Text>
210
211                 />
212                 </>
213             }
214         </View>
215     )
```

Fonte: Autoria própria

Também no FlatList é passado na opção renderItem uma constante getUserItem, este por fim contém os componentes que serão exibidos na tela, como textos, botões etc conforme a figura 20.

Figura 20 - Constante getUserItem

```
const getUserItem =({ item }) => {
    return (
        <View style={style.container}>
            <View style={style.productContainer}>
                <View style={style.product}>
                    <Image
                        style={style.productIamge}
                        source={{uri: item.fotos[0]}}
                    />
                    <Text style={style.productTile}>{item.nome}</Text>
                    <Text style={style.productSubTitle}>{item.raca.raca}</Text>
                    <View style={style.idadeContainer}>
                        <Text style={style.idade}></Text>
                    </View>

                    <View style={style.buttonsContainer}>
                        <TouchableOpacity style={style.buttonComentarios} onPress={
                            () => navigation.navigate('comentariosPostAdocao',item)  }>
                            <Text style={style.buttonTextComentarios}>COMENTARIOS</Text>
                        </TouchableOpacity>
                        <TouchableOpacity style={style.button} onPress={
                            () => navigation.navigate('PostAdocaoDetalhado',item)  }>
                                <Text style={style.buttonText}>MAIS INFORMAÇÕES</Text>
                        </TouchableOpacity>
                    </View>
                </View>
            </View>
        </View>
    )
}
```

Fonte: Autoria própria

O projeto ficará como código-aberto. O código fonte ficará em dois repositórios, um para o front-end e um para o back-end. Os links para esses repositórios são:

- Back-End: https://github.com/MarcosAlbrecht/api-tcc
- Front-End: https://github.com/MarcosAlbrecht/petsApp-V2

4 RESULTADOS

O objetivo do desenvolvimento deste trabalho foi a implementação de um sistema mobile que funcione como uma vitrine para exibir animais para adoção.

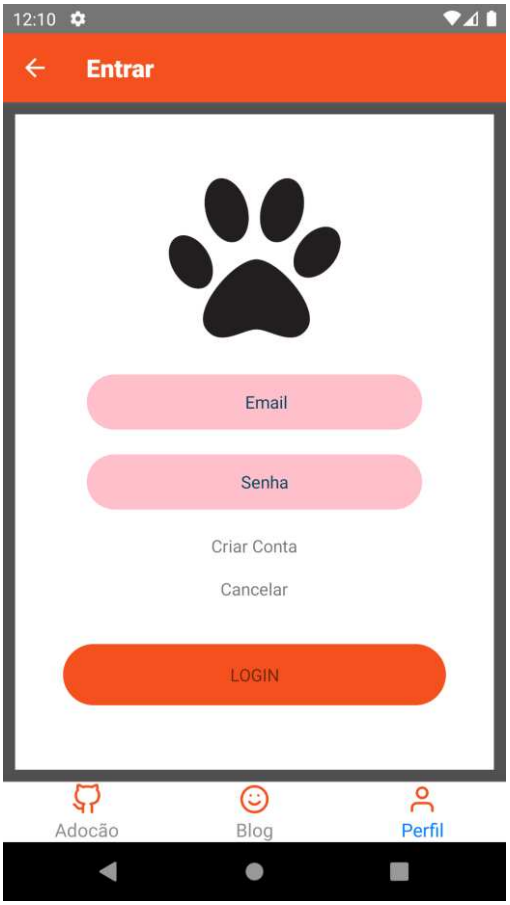
Na implementação do projeto back-end em Java, a utilização do Spring Boot e Maven apresentou benefícios e agilidade na configuração do projeto, o que anteriormente era uma das principais dificuldades entre os programadores que iniciavam com a tecnologia Java. Essas tecnologias apresentam, ainda, grande agilidade no desenvolvimento Java para web com a utilização do Spring Framework e seus projetos, como, por exemplo, o Spring Data e Spring Security.

Para o desenvolvimento *mobile* foi utilizada uma tecnologia híbrida (React Native) para aplicações empresariais. Como o seu consumo não atinge um grande volume de usuários, essa tecnologia se mostrou bastante eficiente, uma vez que o mesmo código pode ser compilado para diversas plataformas, permitindo atender maior
diversidade de mercado em termos de dispositivos móveis.

Como resultados parciais a criação de layouts para as telas que foram desenvolvidas, auxiliando o manuseio do aplicativo pelo usuário. As telas que foram desenvolvidas são as telas de postagens de animais para doação, cadastro, edição de postagens para doação e comentários de postagens de doação, tela de postagens pessoais, cadastro, edição de postagens pessoais e comentários de postagens pessoais, tela de perfil e tela de cadastro e edição do perfil, tela para login e tela para filtro de cadastro para doação. Essas telas foram desenvolvidas possuindo as interações e um fluxo de navegação entre as mesmas.

A Figura 21, baseia-se em uma tela de login contendo os campos de preenchimento de e-mail e senha, e dois botões para efetuar o login e para a criação de uma nova conta.

Figura 21 - Tela de login



Fonte: Autoria própria

A Figura 22, exibe as publicações das postagens de animais para adoção, onde o usuário poderá aplicar um filtro, cadastrar uma nova postagem, visualizar mais detalhes sobre a postagem e realizar um comentário.

Figura 22 - Tela de postagens de animais para adoção

Fonte: Autoria própria

A Figura 23 mostra a tela para realizar o cadastro ou edição de uma postagem de animal para adoção.

Figura 23 - Tela de cadastro e edição de postagem de animais para adoção

Fonte: Autoria própria

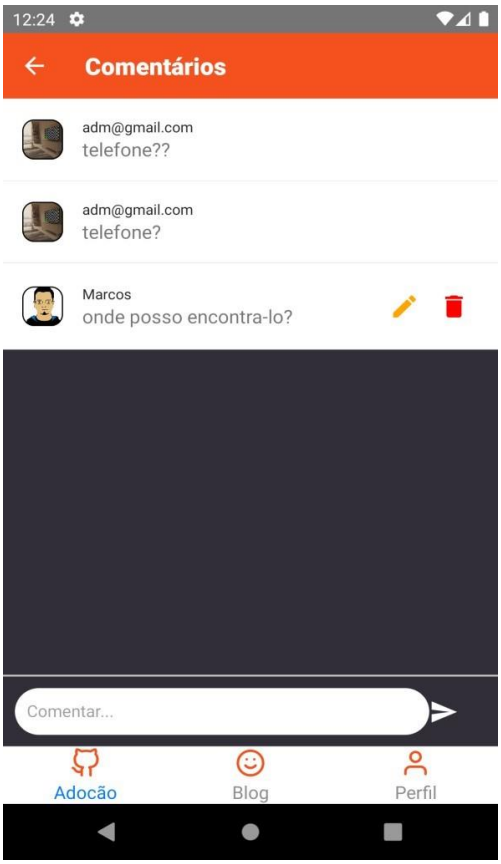
A figura 24 mostra a tela de informações detalhadas sobre o animal.

Figura 24 - Tela de informações detalhadas do animal

Fonte: Autoria própria

A figura 25, exibe os comentários de uma pastagem de animal para adoção, o intuito dessa tela é os usuários poderem obter mais informações.

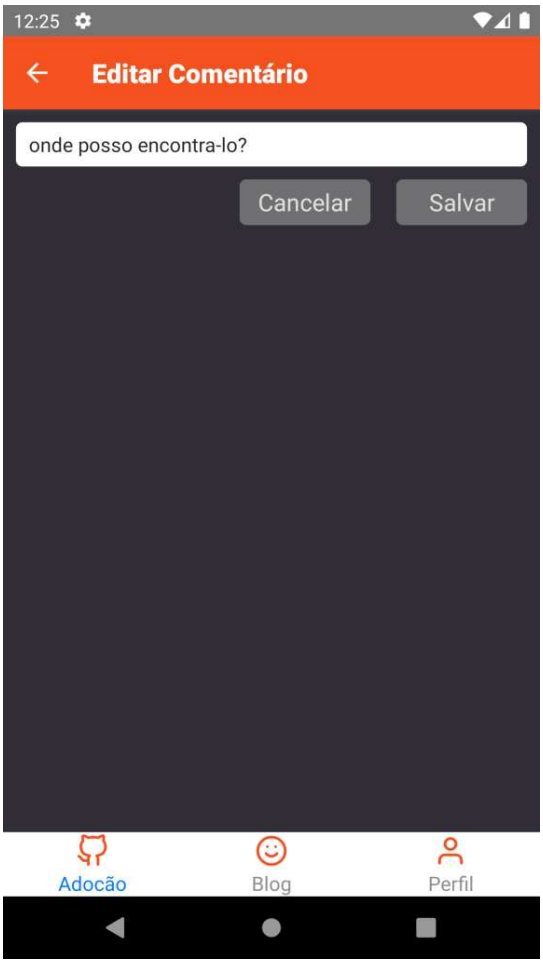
Figura 25 - Tela de comentários postagem para adoção



Fonte: Autoria própria

A figura 26 exibe a tela para edição de um comentário.

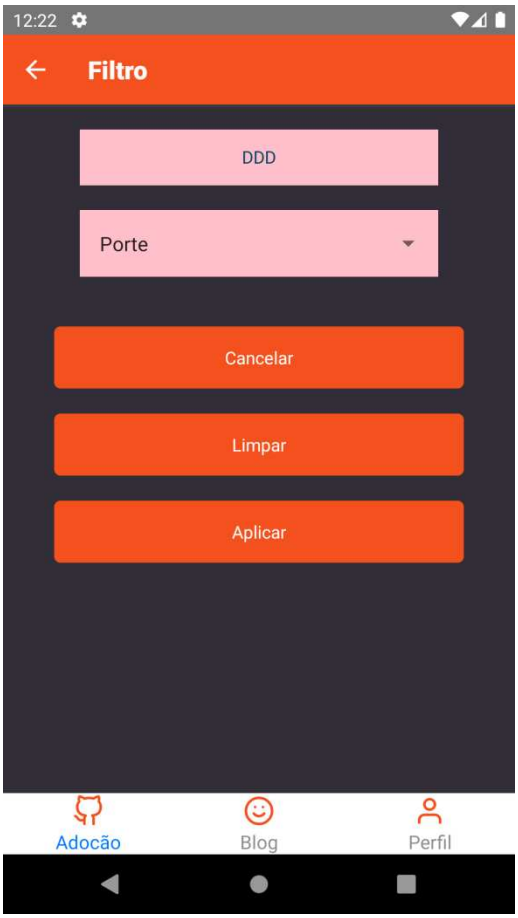
Figura 26 - Tela edição de comentário



Fonte: Autoria própria

A figura 27, mostra a tela para realizar um filtro nas postagens para adoção, visto que o usuário queira pesquisar por animais perto do seu local, ou ainda por tipo.

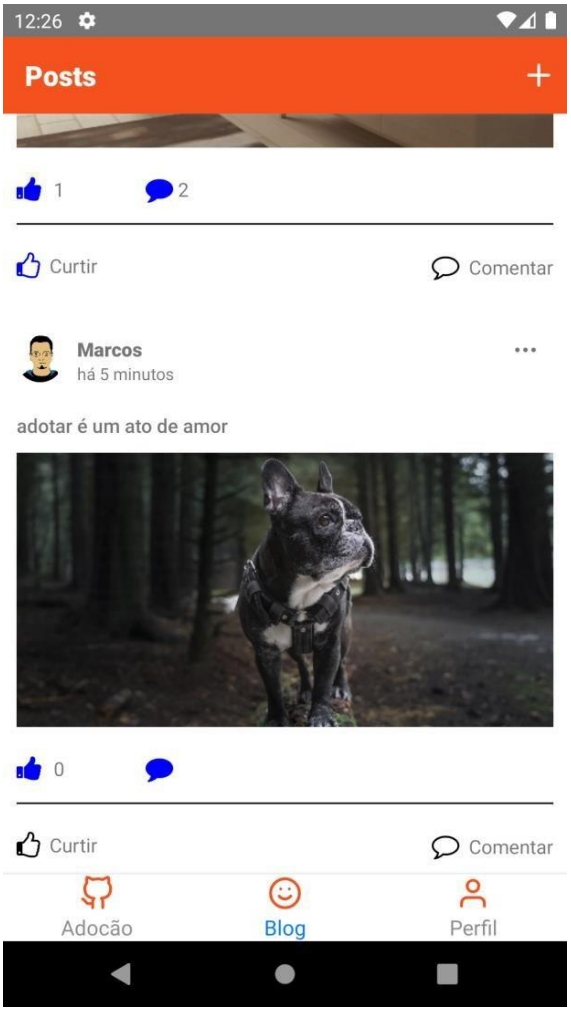
Figura 27 - Tela de filtro



Fonte: Autoria própria

A figura 28, mostra a tela principal do menu *Blog*, o qual contém as postagens de experiências pessoais com os animais, o intuito dessa tela é os usuários poderem ter uma interação com os demais usuários do aplicativo, além de poder contar suas histórias com seus pets.

Figura 28 - Tela principal de postagens pessoais



Fonte: Autoria própria

A figura 29, exibe a tela para cadastro ou edição de uma postagem pessoal.

Figura 29 - Tela cadastro e edição de postagem pessoal

Fonte: Autoria própria

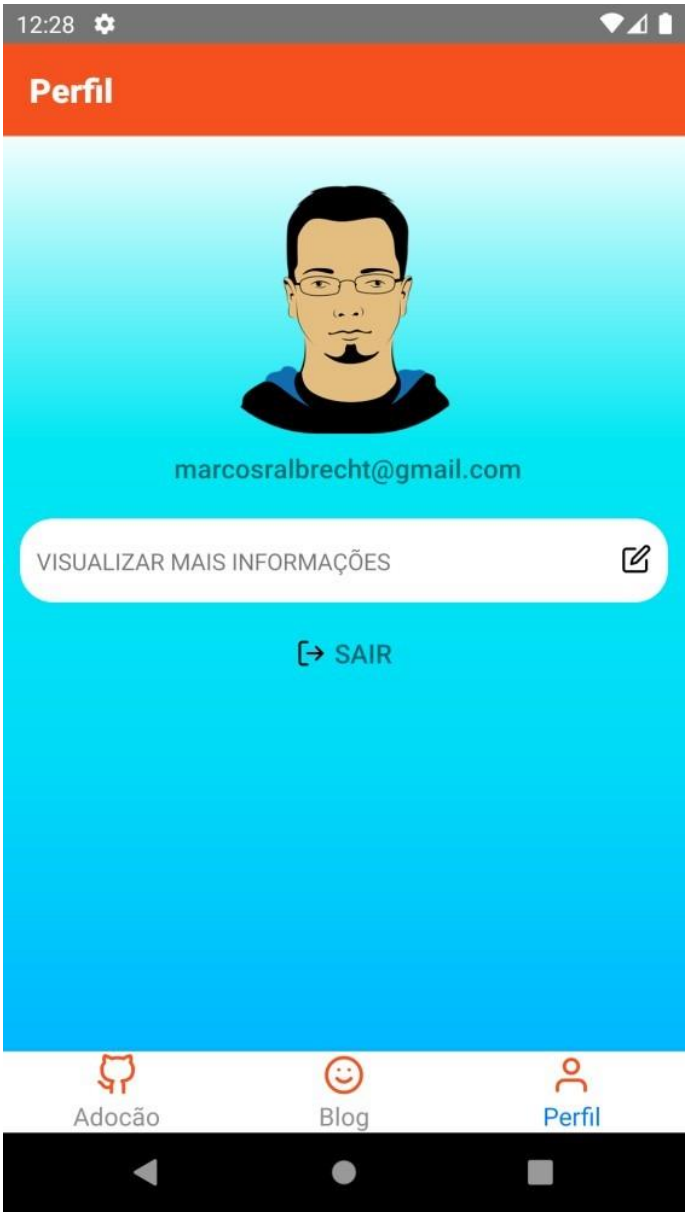
A figura 30 exibe a tela de comentários das postagens pessoais.

Figura 30 - Tela de comentários das postagens pessoais

Fonte: Autoria própria

A figura 31 mostra a tela de perfil do usuário, nela é possível visualizar os dados de cadastro bem como carregar uma foto do dispositivo.

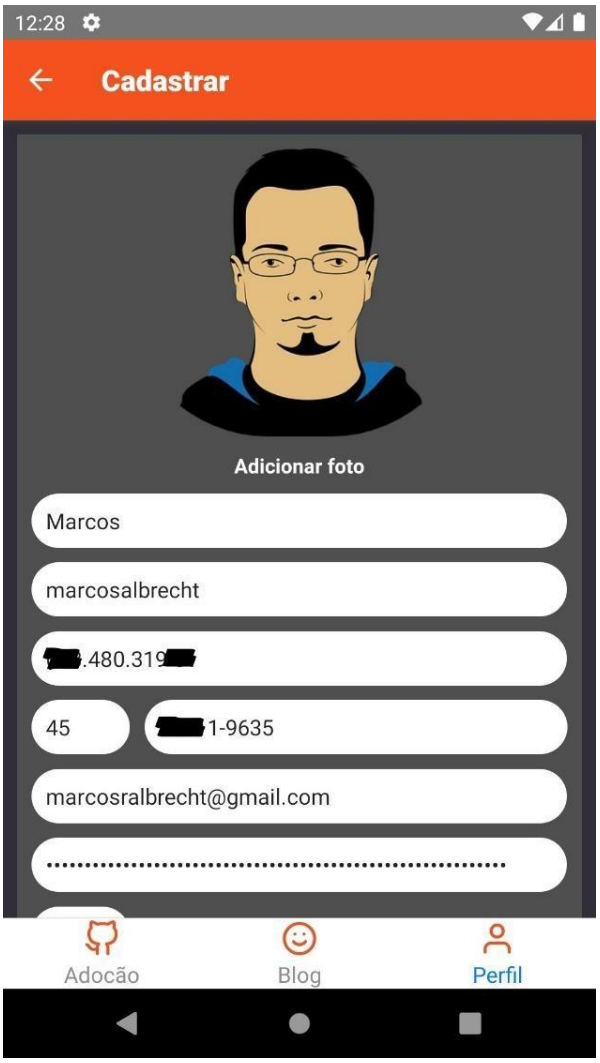
Figura 31 - Tela de perfil do usuário



Fonte: Autoria própria

E por fim, a figura 32 mostra a tela para realizar cadastro e edição do perfil de usuário.

Figura 32 - Tela de cadastro e edição de perfil



Fonte: Autoria própria

TRABALHOS FUTUROS

Existe ainda muito espaço para melhorias e adições de novos recursos ao aplicativo, bem como realizar modificações no layout do aplicativo, a fim de melhorar a experiência do usuário e deixá-lo mais moderno.

Algumas das mudanças/melhorias/adições são:

- Implementação de um usuário administrador para poder gerir todas as postagens e usuários.
- Função usuários premium ter a possibilidade de cadastrar mais fotos.
- Hospedar o serviço em nuvem, tanto o banco de dados MongoDB como o back-end Spring Boot.
- Adicionar funcionalidade de poder exibir todas as postagens do usuário.
- Adicionar funcionalidade de ao clicar sobre o nome de um usuário nas postagens, direcionar para o perfil do usuário.
- Adicionar paginação nas buscas back-end, para melhorar ainda mais o desempenho de visualizações no front-end.

CONSIDERAÇÕES FINAIS

Dado a problemática envolvendo os animais de rua e objetivando a destinação dos animais a tutores responsáveis, o presente trabalho de conclusão de curso teve como objetivo desenvolver uma aplicação para que ONGs, protetores independentes, centros de zoonoses e demais usuários possam utilizar para a divulgação de animais para adoção.

Com as funcionalidades de descrição das características dos animais, prospecta-se que a devolução de animais adotados também seja reduzida, uma vez que o perfil do animal adotado serão descritos informações como, idade, tamanho, raça, pelagem e fotos, o que auxilia o adotante a verificar se o animal em questão atende seus desejos e necessidades, e a funcionalidade de compartilhamento de experiências pessoais permitirá o acompanhamento da relação tutor - animal adotado.

Do ponto de vista acadêmico, o desenvolvimento deste trabalho contribui para a formação de expertise no desenvolvimento de aplicações mobile. Ainda, o desenvolvimento deste trabalho contribui para formação profissional do autor com a aquisição de novas habilidades, além de reforçar os conhecimentos e competências adquiridos durante a graduação em Tecnologia em Sistemas paraInternet.

REFERÊNCIAS BIBLIOGRÁFICAS

BANKER, K. MongoDB in Action. In: OLIVEIRA, R. da S. **Utilizando o algoritmo de levenshtein e mongodb em dados de licitações governamentais.** 2017. Monografia (Especialização em Banco de Dados) - Universidade Federal do Mato Grosso, Cuiabá, 2017. Disponível em: <https://bdm.ufmt.br/bitstream/1/376/1/TCCP_2016_Roberto%20da%20Silva%20Oliveira.pdf>. Acesso em: 14 mai. 2022.

BOAGLIO, F. **Spring Boot: Acelere o desenvolvimento de microsserviços.** Casa do Código. 2017 Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=GvYlEAAAQBAJ&oi=fnd&pg=PT2&dq=BOAGLIO,+2017&ots=VLgPpvymcJ&sig=ND1HkFW43KFPgTmYsJPwFwkDP-U#v=onepage&q=BOAGLIO%2C%202017&f=false>. Acesso em: 18 mai. 2022.

BOOCH, G; RUMBAUGH, J; JACOBSON, I. **UML: guia do usuário**. Elsevier, 2005.

CABRAL, C. **React Native: Construa aplicações móveis nativas com JavaScript**. 2016. Disponível em: <https://tableless.com.br/react-native-construa-aplicacoes-moveis-nativas-com-javascript/>. Acesso em: 13 mai. 2022.

CÂMARA, R. **O que você deve saber sobre o funcionamento do React Native**. 2018. Disponível em:< https://tableless.com.br/o-que-voce-deve-saber-sobre-funcionamento-react-native/>. Acesso em: 6 mai. 2022.

CAETANO, E. C. S. **As contribuições da TAA - Terapia Assistida por Animais à Psicologia**. 2010. Trabalho de Conclusão de Curso (Graduação em Psicologia). Universidade do Extremo Sul Catarinense - UNESC, Criciúma.

Disponível em: <https://silo.tips/download/as-contribuioes-da-taa-terapia-assistida-por-animais-a-psicologia>. Acesso em: 20 mai. 2022.

CARDOSO, Sandra P. D. **Causas de renúncia de cães e gatos nos concelhos de Cascais e Sintra**. 2013. Dissertação (Mestrado em Medicina Veterinária) - Universidade Lusófona de Humanidades e Tecnologias, Lisboa, 2013. Disponível em: <http://recil.grupolusofona.pt/bitstream/handle/10437/5353/Tese%20-%20Sandra% 20Cardoso .pdf? sequence=1 >. Acesso em: 04 jun. 2022.

CARVALHO, L. 9 benefícios que bichos de estimação trazem à saúde. **Exame**. 2016. Disponível em: <https://exame.com/casual/9-beneficios-que-bichos-de-estimacao-trazem-a-saude/>. Acesso em: 05 jun. 2022.

CENSO PET: 139,3 milhões de animais de estimação no Brasil. **Instituto pet brasil.** 2019. Disponível em <http://institutopetbrasil.com/imprensa/censo-pet-1393-milhoes-de-animais-de-estimacao-no-brasil/>. Acesso em: 25 mai. 2022.

FUNDAÇÃO GETÚLIO VARGAS. **Pandemia acelerou processo de transformação digital das empresas no Brasil, revela pesquisa**. Disponível em: < https://portal.fgv.br/noticias/pandemia-acelerou-processo-transformacao-digital-empresas-brasil-revela-pesquisa?utm_source=portal-fgv&utm_medium=fgvnoticias&utm_campaign=fgvnoticias-2021-05-26>. Acesso em: 05 mai. 2022.

CORONATO, M. 3 comportamentos péssimos que levam ao abandono de animais, medidos pelo Ibope. **Época**. 2016. Disponível em: <https://epoca.oglobo.globo.com/vida/noticia/2016/06/3-comportamentos-pessimos-que-levam-ao-abandono-de-animais-segundo-o-ibope.html>. Acesso em: 16 mai. 2022.

DELABARY, B. F. Aspectos que influenciam os maus tratos contra animais no meio urbano. **Revista Eletrônica em Gestão, Educação e Tecnologia**

Ambiental. Santa Maria, v.5, n.5, p. 835 - 840, 2012. Disponível em: <https://periodicos.ufsm.br/reget/article/view/4245>. Acesso em: 13 mai. 2022.

EIS, D. **Flexbox – Organizando seu layout**. 2012. Disponível em: < https://tableless.com.br/flexbox-organizando-seu-layout/>. Acesso em: 20 mai. 2022.

EISENMAN, B. **Writing Cross-Platform Apps with React Native**, 2016. Disponível em: <https://www.infoq.com/articles/react-native-introduction/>. Acesso em: 1 mai. 2022.

FARACO, C. B. **Interação humano-cão: o social constituído pela relação interespécie**. 2008. Tese (Doutorado em Psicologia) - Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2008. Disponível em:<http://livros01.livrosgratis.com.br/cp052930.pdf>. Acesso em: 5 mai. 2022.

FILHO, G. L. Da S. **Desenvolvimento de aplicativo para adoção de animais abandonados utilizando a linguagem de programação Kotlin e programação reativa**. 2017. Trabalho de Conclusão de Curso (Graduação em Engenharia da Computação) - Universidade Tecnológica Federal do Paraná, Curitiba, 2017. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/handle/1/8462>. Acesso em: 27 mai. 2022.

FONTES, H. **Mercado de aplicativos cresce no Brasil e alunos da USP em São Carlos conquistam espaço no cenário**. Jornal da USP. São Carlos, 2016. Disponível em: <http://jornal.usp.br/universidade/mercado-de-aplicativos-cresce-no-brasil-e-alunos-da-usp-em-sao-carlos-conquistam-espaco-no-cenario/>. Acesso em: 22 mai. 2022.

GARCIA, R. C. M. Normas e políticas públicas para controle populacional de cães e gatos. In: **Congresso Brasileiro de Bioética e Bem-Estar Animal**, 3., 2014, Curitiba: Anais...Curitiba: UFPR/LABEA, 2014. p. 149.

GONÇALVES, A. M. **Abandono de animais bate recorde na pandemia e problema não é só brasileiro**. 2021. Disponível em <https://www.uol.com.br/nossa/colunas/coluna-doveterinario/2021/03/11/abandono-de-animais-bate-recorde-na-pandemia-e-problema-nao-e-so-brasileiro.htm>. Acesso em: 16 mai. 2022.

GUERIN, K. **Programa Permanente de Controle Reprodutivo de Cães e Gatos Relacionando o Impacto na Sociedade**. 2009. Trabalho de Conclusão de Curso (Graduação em Medicina Veterinária) - Faculdades Metropolitanas Unidas, São Paulo, 2009. Disponível em: <https://arquivo.fmu.br/prodisc/medvet/kg.pdf>. Acesso em: 15 mai. 2022.

KUPKA, F. **O que é React Native?**. 2017. Disponível em: < https://www.organicadigital.com/blog/o-que-e-react-native/>. Acesso em: 25 mai. 2022.

MEIER, R. Professional Android™ Application Development. In: BELO, J. D. L. M. R. **SYPEC: Desenvolvimento de uma aplicação Android para controlo e avaliação postural.** 2012. Dissertação (Mestrado em Engenharia Biomédica) - Universidade Nova de Lisboa, Lisboa, 2012. Disponível em: <https://run.unl.pt/bitstream/10362/8873/1/Belo_2012.pdf>. Acesso em: 15 mai. 2022.

MILFONT, C. **JSX, a resposta do React pra resolver definitivamente um problema**. 2016. Disponível em: < https://medium.com/@milfont/jsx-a-resposta-do-react-pra-resolver-definitivamente-um-problema-99f572316eb5> . Acesso em: 20 mai. 2022.

MOL, S. Mesmo sem transmitir o coronavírus, cães e gatos têm sido alvo deabandono. **Semad**. 2020. Disponível em: <http://www.meioambiente.mg.gov.br/noticias/4135-mesmo-sem-transmitir-o-coronavirus-caes-e-gatos-tem-sido-alvo-de-abandono>. Acesso em: 20 mai. 2022.

MONTEIRO, F. **React Native: Webview ou realmente nativo?**. 2017. Disponível em: <https://medium.com/nutripad/react-native-webview-ou-realmente-nativo-4e30a37ae020 >. Acesso em: 6 mai. 2022.

MULESOFT. **What is a REST API design?** 2020. Disponível em: < https://www.mulesoft.com/resources/api/what-is-rest-api-design>. Acesso em: 13 mai. 2022.

OLIVEIRA, R. da S. **Utilizando o algoritmo de levenshtein e mongodb em dados de licitações governamentais.** 2017. Monografia (Especialização em Banco de Dados) - Universidade Federal do Mato Grosso, Cuiabá, 2017. Disponível em: <https://bdm.ufmt.br/bitstream/1/376/1/TCCP_2016_Roberto%20da%20Silva%20Oliveira.pdf>. Acesso em: 14 mai. 2022.

OLIVEIRA, I. **React Native é mesmo nativo?**. 2017. Disponível em: < https://programadorbr.com/blog/react-native-e-mesmo-nativo/>. Acesso em: 13 mai. 2022.

ORACLE. **Obtenha informações sobre a Tecnologia Java**. 2017. Disponível em: <https://www.java.com/pt-BR/about/>. Acesso em 14 mai. 2022.

PAULA, S. A. de. **Política pública de esterilização cirúrgica de animais domésticos, como estratégia de saúde e de educação**. 2012. Monografia (Especialização em Gestão Pública Municipal) - Universidade Tecnológica Federal do Paraná, Curitiba, 2012. Disponível em: < http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/1495/4/CT_GPM_II_2012_32.pdf >. Acesso em: 14 mai. 2022.

PEDRASSANI, C. E. **Uma solução em nodejs e react native para busca e oferta de emprego.** 2018. 90f. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) — Faculdade Antonio Menegheti-AMF, Restinga Sêea,2018. Disponível em: < http://repositorio.faculdadeam.edu.br/xmlui/bitstream/handle/123456789/309/TC

C_SI_CARLOS_EDUARDO_PEDRASSANI_AMF_2018.pdf?sequence=1&isAllowed=y>. Acesso em 14 mai. 2022.

SADALAGE, P. J.; FOWLER, M. **NoSQL Essencial: Um guia conciso para o Mundo emergente da persistência poliglota.** [S.l.]: Novatec Editora, 2013.

SANTOS, P. Algumas questões relativas ao encaminhamento de cães e gatos para adoção. **Revista de Antropologia da UFSCar**, São Carlos, v.7, n. 1, p. 230-247, jan-jun. 2015. Disponível em: <http://www.rau.ufscar.br/wp-content/uploads/2016/09/12_rau07104.pdf>. Acesso em: 14 mai. 2022.

SANTOS, R. S. Dos. **Medmob – aplicativo de consultas médicas utilizando react native e node.js.** 2021. Monografia (Engenheiro da Computação) - Universidade Federal do Amazonas, Manaus, 2021. Disponível em: <https://riu.ufam.edu.br/bitstream/prefix/5982/8/TCC_RodrigoSantos.pdf>. Acesso em: 05 jun. 2022.

SILVA, D. A. da.; SOUZA, C. S. de. Construção de app com react native. **RevistaTecnologias em Projeção**, v. 10, n. 1, 2019. Disponível em: < http://revista.faculdadeprojecao.edu.br/index.php/Projecao4/article/view/1316/1059 >. Acesso em: 03 jun. 2022.

SOMMERVILLE, I. **Engenharia de software, 10ª ed.** Editora Pearson, 2018.

STROZZI, C. NoSQL: A non-SQL RDBMS. In: NOGUERA, V. E. R. **Extensão de uma álgebra ER para execução de consultas em bancos de dados NoSQL orientados a documentos.** 2018. Dissertação (Mestrado em Ciência da Computação, área de concentração: Engenharia de Software) - Universidade Federal de São Carlos, São Carlos, 2018. Disponível em: <https://repositorio.ufscar.br/bitstream/handle/ufscar/9716/NOGUERA_Viviana_2018.pdf?sequence=4&isAllowed=y>. Acesso em: 03 jun. 2022.

TEIXEIRA, P. H. F. **Uma API REST para a contratação de profissionais na aplicação Severino**. 2021. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Federal de Uberlândia, Uberlândia,

2021. Disponível em: <https://repositorio.ufu.br/handle/123456789/32435>. Acesso em: 03 jun. 2022.

VENTEU, K. C.; PINTO, G. S. **Desenvolvimento Móvel Híbrido**. Revista Interface Tecnológica, *[S. l.]*, v. 15, n. 1, p. 86-96, 2018. Disponível em: <https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/337>. Acesso em: 03 jun. 2022.