UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

TORTUGA: APLICATIVO PARA IDENTIFICAÇÃO DE CÁGADOS DA ESPÉCIE PHRYNOPS WHILLIAMSI

GABRIEL HENRIQUE BIZ

BLUMENAU
2017

GABRIEL HENRIQUE BIZ

TORTUGA: APLICATIVO PARA IDENTIFICAÇÃO DE CÁGADOS DA ESPÉCIE PHRYNOPS WILLIAMSI

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Aurélio Faustino Hoppe, Mestre - Orientador

BLUMENAU
2017

TORTUGA: APLICATIVO PARA IDENTIFICAÇÃO DE CÁGADOS DA ESPÉCIE PHRYNOPS WILLIAMSI

Por

GABRIEL HENRIQUE BIZ

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Daniel Theisges dos Santos, Mestre – FURB

Blumenau, 03 de julho de 2017

Dedico este trabalho aos meus pais, que sempre me apoiaram e incentivaram na conclusão deste curso de graduação.

AGRADECIMENTOS

À minha família que sempre apoiou e incentivou os meus estudos.

À minha namorada, por me ajudar nos momentos difíceis.

Ao meu orientador, Aurélio Hoppe, que sugeriu o tema do trabalho, e prestou todo o seu apoio para a conclusão do mesmo.

E aos meus amigos e colegas do curso, que sempre me apoiaram e ajudaram durante esses anos.

Conhecimento não é aquilo que você sabe, mas o que você faz com aquilo que você sabe.

Aldous Huxley

RESUMO

Este trabalho apresenta o desenvolvimento de um protótipo para auxiliar biólogos no monitoramento e acompanhamento de cágados da espécie Phrynops Williamsi. O protótipo desenvolvido é uma extensão do trabalho Tortuga (BERTOLDI, 2016), que tem como objetivo o desenvolvimento de um aplicativo Android capaz de realizar a identificação do cágado a partir de uma imagem da listra em formatado de ferradura e da listra circular localizadas na parte inferior da cabeça dessa espécie. Os dados são armazenados no próprio dispositivo, com a possibilidade de exporta-los para o Google Drive e para a aplicação web desenvolvida. A aplicação web permite ao usuário visualizar os cágados que foram catalogados por outros usuários. A visualização do cágado apresenta os locais da captura no Google Maps, cada marcador apresentado no mapa possui a data e hora da captura. Os resultados obtidos através de testes de usabilidade demonstraram que o protótipo desenvolvido mostrou-se eficaz no monitoramento de cágados e a solução foi bem aceita pelos participantes do experimento.

Palavras-chave: Monitoramento e acompanhamento. Cágados. Android.

ABSTRACT

This work presents the development of a prototype to assist biologists in the monitoring and overseeing of tortoises of the Phrynops Williamsi species. The prototype developed is an extension of the Tortuga work (BERTOLDI, 2016), which aims to develop an Android application capable of performing the identification of the tortoise from an image of the horseshoe formatted stripe and the circular stripe located in the part The head of this species. The data is stored on the device itself, with the possibility of exporting it to Google Drive and to the developed web application. The web application allows the user to view the turtles that have been cataloged by other users. The tortoise display shows the capture locations in Google Maps, each marker displayed on the map has the capture date and time. The results obtained through usability tests demonstrated that the prototype developed proved to be effective in the monitoring of tortoises and the solution was well accepted by the participants of the experiment.

Key-words: Monitoring and overseeing. Tortoises. Android.

LISTA DE FIGURAS

Figura 1 – Marcação de quelônios 18
Figura 2 – Esquema de codificação numérica para marcação de quelônios 18
Figura 3 – Biometria de quelônios 19
Figura 4 – Partes de um SIG 21
Figura 5 – Aplicação Plantarum 23
Figura 6 – Aplicação Pic4Turtle 24
Figura 7 – Aplicação Leafsnap 25
Figura 8 – Aplicação LikeThat Garden 26
Figura 9 – Tela para identificação de cágados do protótipo Tortuga 28
Figura 10 – Diagrama de casos de uso da aplicação Android 30
Figura 11 – Diagrama de casos de uso da aplicação web 31
Figura 12 – Arquitetura da solução 32
Figura 13 – Diagrama de pacotes da aplicação Android 33
Figura 14 – Pacotes do pacote `ui` 34
Figura 15 – Pacotes do pacote `data` 35
Figura 16 – Diagrama de pacotes da aplicação Android 37
Figura 17 – Diagrama de atividades da aplicação 39
Figura 18 – Tela principal da aplicação para seleção da imagem 40
Figura 19 – Tela de seleção das listras e resultado 41
Figura 20 – Tela para cadastro do cágado e acompanhamento 44
Figura 21 – Telas da galeria de imagens do acompanhamento 46
Figura 22 – Tela para visualização dos locais de captura 47
Figura 23 – Tela principal da aplicação / seleção da conta 49
Figura 24 – Tela para seleção de pasta do Google Drive 51
Figura 25 – Tela que apresenta o conteúdo da pasta criada ao realizar a exportação dos dados 54
Figura 26 – Tela para visualização dos cágados da aplicação web 60
Figura 27 – Tela para visualização do perfil do cágado da aplicação web 60

LISTA DE QUADROS

Quadro 1 – Requisitos funcionais da aplicação Android 29
Quadro 2 – Requisitos não funcionais da aplicação Android 29
Quadro 3 – Requisitos funcionais da aplicação web 29
Quadro 4 – Requisitos não funcionais da aplicação web 29
Quadro 5 – Código do método `getCurrentLocation` da classe `LocationService`. 42
Quadro 6 – Código do método `loadResults` da classe `ResultActivity` 43
Quadro 7 – Código do método `find` da classe `SearchService` 43
Quadro 8 – Código do método `loadAnnotation` da classe `CreateAnnotationActivity` 45
Quadro 9 – Código do método `getAddress` da classe `LocationService` 45
Quadro 10 – Código do método `onCreate` da classe `MapsActivity` 48
Quadro 11 – Código do método `onMapReady` da classe `MapsActivity` 48
Quadro 12 – Código do método `onGoogleApiClienteConnected` da classe `ExportActivity` 50
Quadro 13 – Código do método `run` da classe `DriveDataExporterRunnable` 51
Quadro 14 – Código do método `exportJson` da classe `DriveDataExporterRunnable` 52
Quadro 15 – Trecho de código do método `exportData` da classe `BackupDataExporter` responsável por exportar os dados da base 52
Quadro 16 – Trecho de código do método `exportData` da classe `BackupDataExporter` responsável por exportar as imagens da aplicação 53
Quadro 17 – Código do método `run` da classe `DriveDataImporterRunnable` 55
Quadro 18 – Código do método `importJson` da classe `DriveDataImporterRunnable` 55
Quadro 19 – Código do método `importData` da classe `BackupDataImporter` 56
Quadro 20 – Código do método `importData` da classe `BackupDataImporter` 57
Quadro 21 –Código do método `onCreate` da classe `MainActivity` 58
Quadro 22 – Código do método `sendTortoise` da classe `TortugaIntegration` 59
Quadro 23 – Perfil dos usuários que realizaram o teste de usabilidade 62
Quadro 24 – Smartphones utilizados nos testes 67

Quadro 25 – Questionário de perfil de usuário..... 72
Quadro 26 – Lista de tarefas..... 73
Quadro 27 – Questionário de usabilidade 76

LISTA DE TABELAS

Tabela 1– Respostas das atividades de acompanhamento 63
Tabela 2 – Respostas das perguntas referentes a usabilidade do protótipo..... 65
Tabela 3 – Respostas referentes as funcionalidades do protótipo 66

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

GPS – Global Positioning System

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

SIG – Sistema de Informação Geográfica

UC – Use Case

UML – Unified Modeling Language

UUID – Universally Unique Identifier

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVOS 16
1.2 ESTRUTURA 16
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 MONITORAMENTO DE QUELÔNIOS 17
2.2 SIG – SISTEMA DE INFORMAÇÃO GEOGRÁFICA 20
2.3 TRABALHOS CORRELATOS 22
2.3.1 PLANTARUM: UMA APLICAÇÃO ANDROID PARA CONSULTAS DE PLANTAS 22
2.3.2 Pic4turtle 23
2.3.3 Leafsnap 25
2.3.4 LikeThat Garden 25
3 DESENVOLVIMENTO 27
3.1 PROTÓTIPO ATUAL 27
3.2 REQUISITOS 28
3.3 ESPECIFICAÇÃO 29
3.3.1 Casos de uso 30
3.3.2 Arquitetura 31
3.3.3 Diagramas de pacotes da aplicação Android 32
3.3.4 Diagrama de pacotes da aplicação web 37
3.4 IMPLEMENTAÇÃO 38
3.4.1 Técnicas e ferramentas utilizadas 38
3.4.2 Diagrama de atividades 39
3.4.3 Seleção da imagem 40
3.4.4 Cadastro do cágado e acompanhamento 44
3.4.5 Histórico de imagens 46
3.4.6 Visualização dos locais 47
3.4.7 Importação e exportação dos dados 49
3.4.8 Aplicação web 57
3.5 ANÁLISE DOS RESULTADOS 61
3.5.1 Teste de usabilidade 61

3.5.2 Teste de backup dos dados e compatibilidade 67
4 CONCLUSÕES 68
4.1 EXTENSÕES 69
REFERÊNCIAS 70
APÊNDICE A – QUESTIONÁRIO DO PERFIL DE USUÁRIO, LISTA DE TAREFAS E AVALIÇÃO DE USABILIDADE 72

1 INTRODUÇÃO

O número estimado de espécies vivas varia entre 10 e 15 milhões, porém apenas dois milhões delas foram nomeadas. Desta forma, a pergunta básica sobre o número de espécies que habitam o planeta está longe de ser respondida. Além disso, um nome é apenas um rótulo, nomeá-las não significa que realmente tem-se conhecimento sobre a espécie. Uma vez que uma espécie é nomeada e seu fenótipo descrito, deve-se saber sobre sua variabilidade e seu ciclo de vida, definir o seu nicho ecológico, compreender o seu papel nas comunidades e nos ecossistemas (BOERO, 2010, p. 116, tradução nossa).

Costa et al. (2013, p. 5) afirmam que a biodiversidade tem papel central para a espécie humana e, que animais, plantas e microrganismos fornecem alimentos, medicamentos e matérias-primas e, são nossa conexão mais evidente com a natureza. Porém, segundo os autores, esta conexão vem sendo afetada pela diminuição da biodiversidade.

Vié et al. (2008, p. 1, tradução nossa) apontam que a diminuição da biodiversidade é uma das crises mundiais mais urgentes, com muitas espécies decaindo para baixos níveis populacionais e um número significativo de espécies em extinção. Diante dessas dificuldades, é essencial desenvolver estratégias de inventário e monitoramento rápido da diversidade biológica, assim como criar a infraestrutura necessária para gerar, armazenar e utilizar dados sobre biodiversidade (CULLEN et al., 2012 p. 19).

Segundo Burghardt (2008, p. 4, tradução nossa), quando pesquisadores estudam grandes populações de animais baseadas em informações de avistamentos, eles comumente adotam uma abordagem de amostragem e análise conhecida como captura-marcação-recaptura. Este método e seus derivados são largamente usados para se obter parâmetros vitais da população, como comportamento, movimentação, sobrevivência, crescimento, tamanho aproximado da população e formas de manejo.

Edwards (2012, p. 1, tradução nossa) garante que a partir do uso de tecnologias adequadas é possível medir o comportamento das espécies em seu ambiente natural e sem perturbações mesmo onde a manipulação experimental é impraticável, indesejáveis ou mesmo antiético. O autor também afirma que utilizando essas tecnologias para o desenvolvimento de indicadores comportamentais pode-se detectar o início precoce de problemas de saúde nos espécimes, assim, diminuindo a probabilidade da extinção de alguns grupos de animais.

Diante do exposto, este trabalho apresenta uma extensão para dispositivos móveis do protótipo Tortuga (BERTOLDI, 2016), que tinha por objetivo criar um identificador único para cágados da espécie Phrynops Williamsi de forma não invasiva. A migração realizada

permite que biólogos possam acompanhar os parâmetros vitais da população de cágados da espécie Phrynops Williamsi.

1.1 OBJETIVOS

O objetivo deste trabalho é estender o protótipo Tortuga para a plataforma Android, migrando o método de identificação e, incluindo recursos para facilitar o acompanhamento e monitoramento de cágados da espécie Phrynops Williamsi.

Os objetivos específicos são:

a) migrar o método de identificação de cágados para dispositivos móveis;
b) disponibilizar um mecanismo para armazenar de maneira *off-line* as informações, imagens e coordenadas de Global Positioning System (GPS) de onde o cágado foi localizado;
c) disponibilizar um mecanismo para visualização dos parâmetros vitais da espécie (características, comportamento, movimentação e localização, sobrevivência e formas de manejo).

1.2 ESTRUTURA

A presente monografia encontra-se dividida em quatro capítulos: introdução, fundamentação teórica, desenvolvimento e conclusões. O capítulo 2 apresenta um embasamento teórico a respeito do monitoramento de quelônios. Na sequência, o capítulo 3 apresenta a abordagem utilizada no desenvolvimento do protótipo, como: os requisitos principais, diagrama de casos de uso, diagrama de pacotes, arquitetura da solução e diagrama de atividades. Além disso, também são destacadas as ferramentas utilizadas e a implementação do protótipo desenvolvido. Por fim, o capítulo 4 apresenta as conclusões obtidas no desenvolvimento do trabalho e as sugestões para extensões que podem ser realizadas no protótipo atual.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em duas seções. A seção 2.1 apresenta algumas informações sobre o monitoramento de quelônios. A seção 2.2 aborda Sistemas de Informação Geográfica descrevendo suas características e vantagens. Na seção 2.3 são apresentados alguns trabalhos correlatos que possuem objetivos semelhantes aos do trabalho desenvolvido.

2.1 MONITORAMENTO DE QUELÔNIOS

O monitoramento de quelônios permite determinar a variação do número de indivíduos ao longo do tempo, bem como compreender os seus processos ecológicos, sendo eles, basicamente sua faixa etária, densidade, razão sexual, taxas de sobrevivência e recrutamento. Para isso é necessário realizar um estudo de longa duração, tendo em vista que quelônios são seres de vida longa, crescimento lento e maturação sexual tardia (BALESTRA et al., 2015, p. 116).

O monitoramento dos quelônios é realizado por meio da marcação dos espécimes capturados. Existe uma grande variedade de técnicas de marcação de quelônios. Sendo que a técnica mais comum utilizada entre os biólogos é realização de furos ou cortes nos escudos marginais da carapaça do animal. Para a realização desse procedimento, recomenda-se que sejam utilizados materiais esterilizados, presando pela biossegurança do animal (BALESTRA et al., 2015, p. 128).

O método de marcação, onde são realizados cortes nos escudos marginais da carapaça do animal, possui baixo custo e, ao mesmo tempo, a desvantagem do possível desaparecimento das marcas devido a regeneração natural. Esta desvantagem é mais evidente em indivíduos muito jovens ou recém-nascidos. Em adultos o corte ou furo geralmente constitui uma marca permanente na carapaça. Comumente o furo é feito com uma furadeira elétrica conforme mostra a Figura 1a e o corte realizado através de pequenas serras ou seguetas conforme exibe a Figura 1b (BALESTRA et al., 2015, p. 128).

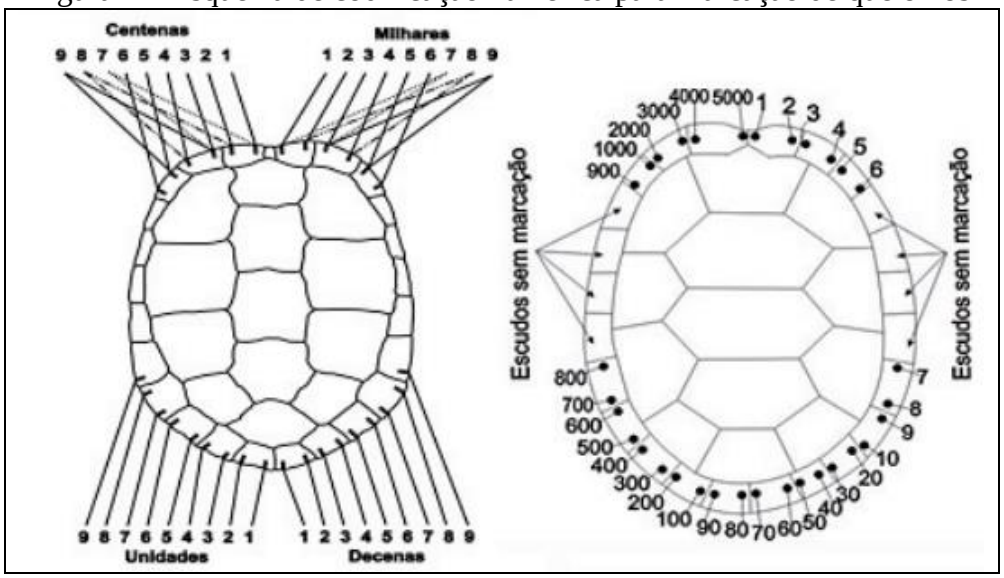
Figura 1 – Marcação de quelônios

(a) marcação com furadeira (b) marcação com serra

Fonte: Balestra et al. (2016, p. 129).

Quando o objetivo do estudo é apenas averiguar captura e recaptura não é necessário individualizar os espécimes. Porém, quando se deseja avaliar crescimento e movimentação é necessário realizar a individualização dos espécimes. Para isso é criada uma codificação numérica, com as unidades, dezenas, centenas e milhares, estabelecida pela disposição dos cortes ou furos nos escudos marginais da carapaça, sendo que cada escudo pode receber até duas numerações (BALESTRA et al., 2015, p. 128). A Figura 2 demonstra dois esquemas de codificação numérica utilizados na marcação de quelônios.

Figura 2 – Esquema de codificação numérica para marcação de quelônios



Fonte: Balestra et al. (2016, p. 129).

No monitoramento individual dos espécimes Balestra et al. (2015, p. 134) destaca as medidas básicas a serem tomadas para os espécimes capturados, sendo elas, comprimentos máximos retilíneos da carapaça (Figura 3a) e plastrão (Figura 3b), largura máxima retilínea da carapaça (Figura 3c) e plastrão (Figura 3d), altura máxima do casco (Figura 3e) e largura cefálica (Figura 3f). Entretanto, em estudos taxonômicos clássicos com base em parâmetros morfométricos, dimorfismo e taxas de crescimento, que objetivam uma avaliação mais

acurada dos caracteres morfométricos da carapaça e plastrão, outras medidas podem ser necessárias (BALESTRA et al., 2015, p. 134).

Figura 3 – Biometria de quelônios

(a) Comprimento carapaça

(b) Comprimento plastrão

(c) Largura carapaça

(d) Largura plastrão

(e) Altura máxima do casco

(f) Largura cefálica

Fonte: Balestra et al. (2016, p. 135).

Para Balestra et al. (2015, p. 145) é muito importante fazer bons registros de imagens dos espécimes capturados e dos seus hábitats. Para isso, recomenda-se o uso de equipamentos fotográficos cuja capacidade resolutiva, regulagem de foco e compensação de excesso ou carência de luz permita analisar apuradamente os caracteres morfológicos de interesse no espécime, bem como características relevantes do ambiente no qual foi capturado. Desta forma, para cada exemplar amostrado, deve-se usar um padrão com escala que permita expor a carapaça, plastrão, a região lateral, membros e cabeça (vista dorsal, ventral e lateral).

Segundo Yoccoz, Nichols e Boulinier (2001, p. 446), os programas de monitoramento voltados para auxiliar a gestão das espécies proveem informações úteis que auxiliam no

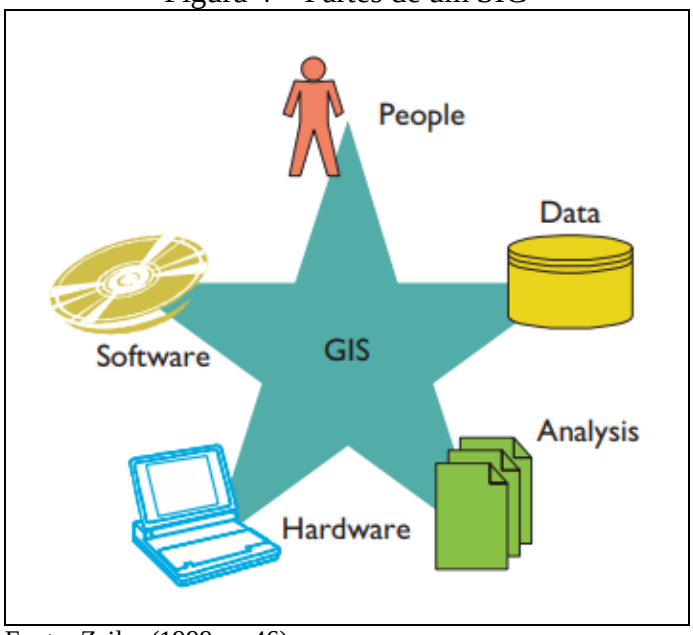
momento em que são tomadas decisões de gestão da espécie. Os resultados de monitoramentos bem delineados permitem avaliar as respostas de uma população às práticas de manejo, a programação de conservação, bem como aos impactos de fatores externos, tais como doenças, caça e conversão de hábitat (BALESTRA et al., 2015, p. 116).

Um dos mais antigos programas de monitoramento em execução no Brasil é o Programa Quelônios da Amazônia (LUSTOSA et al., 2016, p. 14). Segundo o autor, o programa foi responsável por realizar o manejo de mais 65 milhões de quelônios nos estados das regiões Norte e Centro-Oeste do Brasil, destacando a tartaruga-da-amazônia e o tracajá. Essas ações têm proporcionado conservar e recuperar as populações naturais dessas espécies. É atribuído aos esforços desse programa, o fato de que nenhuma dessas espécies aparece em lista de ameaça de extinção no território brasileiro. Porém, Lustosa et al. (2016, p. 14) destacam que a manutenção de índices populacionais desejáveis dessas espécies depende da continuidade dos trabalhos de proteção, manejo e monitoramento.

2.2 SIG – SISTEMA DE INFORMAÇÃO GEOGRÁFICA

Segundo Lisboa Filho e Iochpe (1996, p. 1) Sistema de Informação Geográfica (SIG) é um conjunto de programas, equipamentos e metodologias, perfeitamente integrados, de forma a tornar possível a coleta, o armazenamento, o processamento e a análise de dados georreferenciados, bem como a produção de informação derivada de sua aplicação. Para Zeiler (1999, p. 46, tradução nossa) SIG é a combinação de pessoas qualificadas, dados espaciais e descritivos, métodos analíticos, software e hardware. Tudo organizado para automatizar, gerenciar e entregar informações através de uma apresentação geográfica. A Figura 4 apresenta as partes de um SIG segundo Zeiler.

Figura 4 – Partes de um SIG



Fonte: Zeiler (1999, p. 46).

Um sistema de geoprocessamento tem por objetivo o processamento de dados referenciados geograficamente, desde a coleta até a geração e a exibição das informações por meio de mapas convencionais, relatórios, arquivos digitais e gráficos, entre outros (SILVA, 2006, p. 16). Para isso, a utilização dos SIGs vem crescendo rapidamente em todo o mundo, uma vez que possibilita um melhor gerenciamento de informações e consequente melhoria nos processos de tomada de decisões em áreas de grande complexidade (LISBOA FILHO; IOCHPE, 1996, p. 1).

Os SIGs podem ser utilizados em diversas aplicações: planejamento e gestão urbana e regional, meio ambiente, infraestrutura, agricultura, segurança, transportes, educação e marketing (SILVA, 2006, p. 23). Estas aplicações ilustram a diversidade de soluções de SIG, e, segundo Zeiler (1999, p. 50, tradução nossa), as suas características mais comuns são:

a) comumente um SIG está integrado com outras aplicações para execução de análise geográfica e científica. Desta maneira, é importante que os dados do SIG estejam estruturados e armazenados de modo a permitir o acesso aos dados distribuídos;
b) uma arquitetura de informação aberta é fundamental, pois facilita a integração de dados geográficos com outros dados, como: dados em tempo real, imagens e bancos de dados corporativos;
c) enquanto o mapa impresso ainda é uma forma comum de apresentação de dados geográficos, aplicações de mapas dinâmicos e acesso a mapas na Internet estão se tornando cada vez mais importantes para auxiliar na tomada de decisão. O acesso interativo proporciona modelos de dados mais sofisticados que permitem consultas

mais ricas e melhores análises;

d) é importante selecionar a estrutura de dados adequada para o tipo de análise que se deseja executar. Algumas aplicações tornam-se eficientes na modelagem do mundo como uma superfície contínua, como em uma imagem ou como conjunto de feições discretas em formato vetor.

Para Zeiler (1999), os SIGs melhoraram a forma como as pessoas interagem com mapas. Você pode facilmente alterar a maneira como uma informação é apresentada e também pode selecionar locais ou objetos para iniciar uma consulta ou análise. A tecnologia SIG ampliou nossa visão de um mapa. Ao invés de uma entidade estática, um mapa é agora uma apresentação dinâmica de dados geográficos. Mapas podem integrar dados de diversas fontes em uma referência geográfica comum. A utilização de mapas permite identificar distribuições, relacionamentos e tendências que não são discerníveis. Um demógrafo pode comparar mapas de áreas urbanas compilados no passado com mapas atuais para orientar a política pública. Um epidemiologista pode correlacionar os locais de surtos de doenças raras com fatores ambientais para encontrar possíveis causas (ZEILER, 1999, p. 24, tradução nossa).

2.3 TRABALHOS CORRELATOS

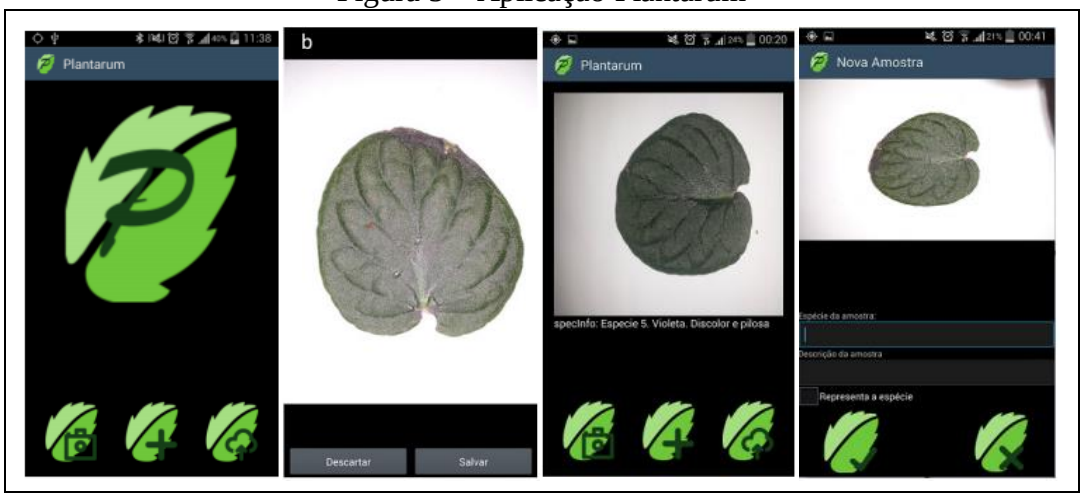
Não foram encontrados trabalhos diretamente relacionados ao objetivo de estudo deste tema. Desta forma, são apresentados trabalhos com objetivos semelhantes ao tema proposto. O primeiro descreve o trabalho de conclusão de curso de Bortolon (2014) que desenvolveu uma aplicação que realiza o cadastro e classificação de espécies de plantas via smartphones para a plataforma Android. O segundo é a aplicação para dispositivos móveis Pic4Turtle (PIC4TURTLE, 2016) que busca identificar a espécie de tartarugas marinhas. O terceiro é a aplicação Leafsnap (LEAFSNAP, 2011) que identifica a espécie das árvores através das suas folhas. E, por fim, a aplicação LikeThat Garden (JUSTVISUAL, 2015) busca identificar a espécie de flores.

2.3.1 PLANTARUM: UMA APLICAÇÃO ANDROID PARA CONSULTAS DE PLANTAS

Bortolon (2014) desenvolveu uma aplicação Android que realiza o cadastro e a classificação de espécies de plantas. Inicialmente, o usuário captura uma imagem da folha através da câmera do dispositivo móvel. Onde, esta imagem precisa estar sobre uma superfície branca e ter boas condições de iluminação. Para realizar uma consulta, além da imagem, é

solicitado ao usuário algumas informações da folha, como tipo, pilosidade, discoloridade e os pontos que representam o pecíolo e ponta da folha. Estas informações são enviadas para um servidor que responde se existe uma espécie correspondente na base de dados. Caso não exista, o usuário pode cadastrá-la. Outra alternativa, é vincular a imagem a uma espécie já cadastrada previamente. A Figura 5 apresenta algumas telas da aplicação Plantarum, como a tela principal da aplicação, tela para selecionar imagem para reconhecimento, tela que apresenta a espécie da folha e a tela para cadastro de uma nova amostra.

Figura 5 – Aplicação Plantarum



Fonte: Bortolon (2014).

A aplicação desenvolvida por Bortolon (2014) possui uma arquitetura cliente-servidor. O cliente foi desenvolvido em Java. Sendo que, para a gravação dos arquivos de imagem foi utilizado a biblioteca Exifdriver e, para efetuar a comunicação via HyperText Transfer Protocol (HTTP) com o servidor, foram utilizadas as bibliotecas Httpclient-4.3, Httpcore-4.3 e Httpmime-4.3 disponibilizadas pela Apache Software Foundation. No desenvolvimento do servidor foi utilizado a linguagem C# (C-Sharp) disponibilizada pela plataforma Microsoft .Net.

Bortolon (2014) indica que os resultados obtidos foram satisfatórios. Onde, testes efetuados com plantas reais (Figura 5) obtiveram uma precisão média de 97.94%. Ele também ressalta que foram encontrados alguns problemas de iluminação e contraste nas imagens capturadas. Para resolver esses problemas, optou-se por fotografar às folhas utilizando um vidro sobre uma superfície branca e com o flash do dispositivo ativado.

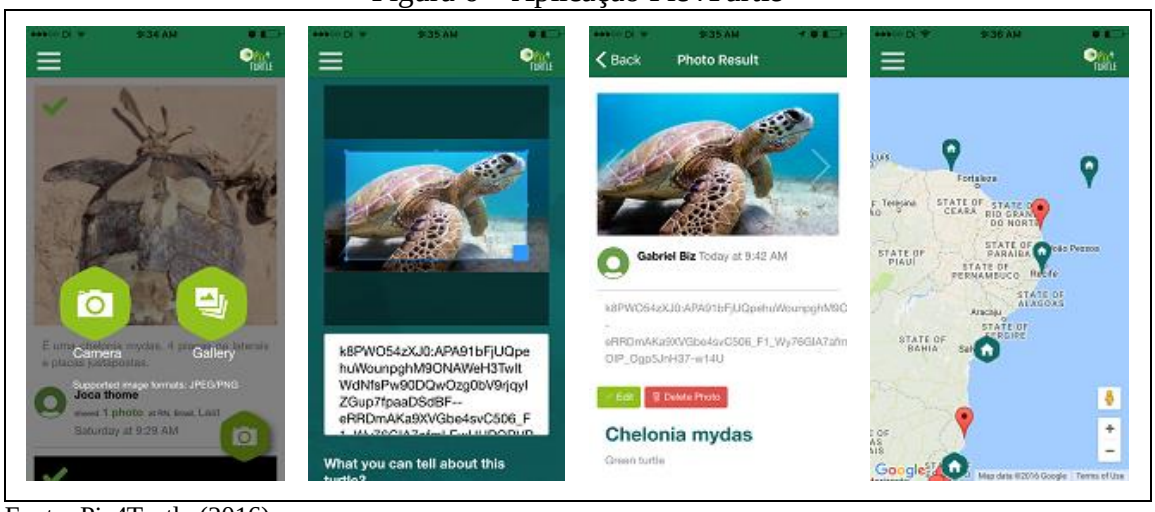
2.3.2 Pic4turtle

O Pic4Turtle (PIC4TURTLE, 2016) é uma aplicação disponibilizada para as plataformas Android e iOS, que permite ao usuário identificar a espécie de uma tartaruga

marinha através de um dispositivo móvel. Para isso, o usuário precisa enviar uma imagem da tartaruga obtida através da câmera do aparelho ou da galeria do próprio dispositivo. O aplicativo suporta dois tipos de formatos de imagem: Joint Photographic Experts Group (JPEG) e Portable Network Graphics (PNG). Após informar a imagem, o aplicativo permite ao usuário selecionar a área ao qual a tartaruga se encontra. Nessa etapa, também são realizadas algumas perguntas sobre as características da tartaruga. Se a espécie for identificada, são retornadas algumas informações tais como: se está em extinção, sua alimentação, onde vivem, tamanho, peso e algumas curiosidades.

A Figura 6 apresenta algumas telas da aplicação Pic4Turtle, onde é possível ver a tela de seleção da imagem, a marcação da tartaruga na imagem selecionada, o reconhecimento da espécie da tartaruga, locais onde outros usuários encontraram tartarugas e unidades de conservação.

Figura 6 – Aplicação Pic4Turtle



Fonte: Pic4Turtle (2016).

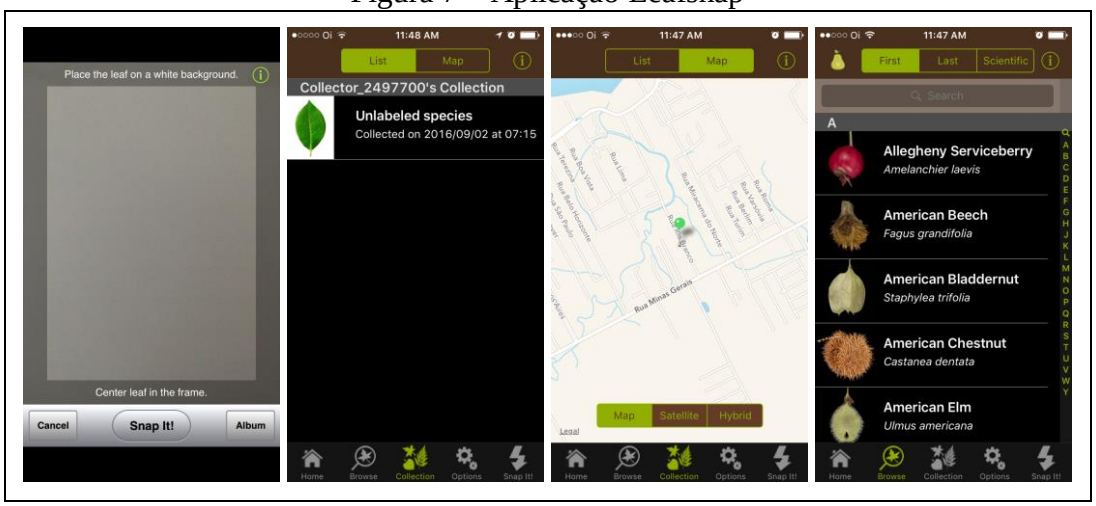
A aplicação permite que os usuários compartilhem ou adicionem comentários em fotos de tartarugas. Quando a foto é compartilhada, especialistas podem confirmar se a espécie da tartaruga está correta. Ao realizar o compartilhamento de uma foto, o aplicativo utiliza o sistema de GPS do dispositivo para apresentar a localização de onde a tartaruga foi encontrada.

O Pic4Turtle (PIC4TURTLE, 2016) também permite que o usuário visualize fotos de tartarugas enviadas por outros usuários ou encontre unidades de conservação espalhadas pelo mundo através do Google Maps. Neste último caso, as unidades de conservação são apresentadas com um ponto verde no mapa, e as tartarugas são marcadas com um ponto vermelho.

2.3.3 Leafsnap

O Leafsnap (LEAFSNAP, 2011) é uma aplicação disponibilizada para a plataforma iOS que permite ao usuário identificar espécies de árvores através de imagens de suas folhas fotografadas sobre um fundo de cor sólida e clara. A foto da folha pode ser obtida através da própria câmera ou da galeria de imagens do dispositivo móvel. Após selecionar a imagem, ela é enviada para o servidor que realiza o reconhecimento da folha e retorna uma lista com as espécies de árvores que mais combinam com a folha. Ao qual, o usuário pode selecionar qual é a espécie correta para a folha em questão. A partir da espécie retornada, o usuário pode visualizar algumas informações tais como habitat, tempo de vida, países em que essa espécie se encontra e uma galeria completa de imagens, com foto de sua folha, fruto, pecíolo, casco e semente. A Figura 7 apresenta algumas telas da aplicação Leafsnap, como a tela para captura de uma foto, onde também é possível obter uma foto através da galeria, tela com os registros de folhas reconhecidas, tela com a localização de onde a folha foi encontrada e a tela para consulta de espécies cadastradas.

Figura 7 – Aplicação Leafsnap



Fonte: Leafsnap (2011).

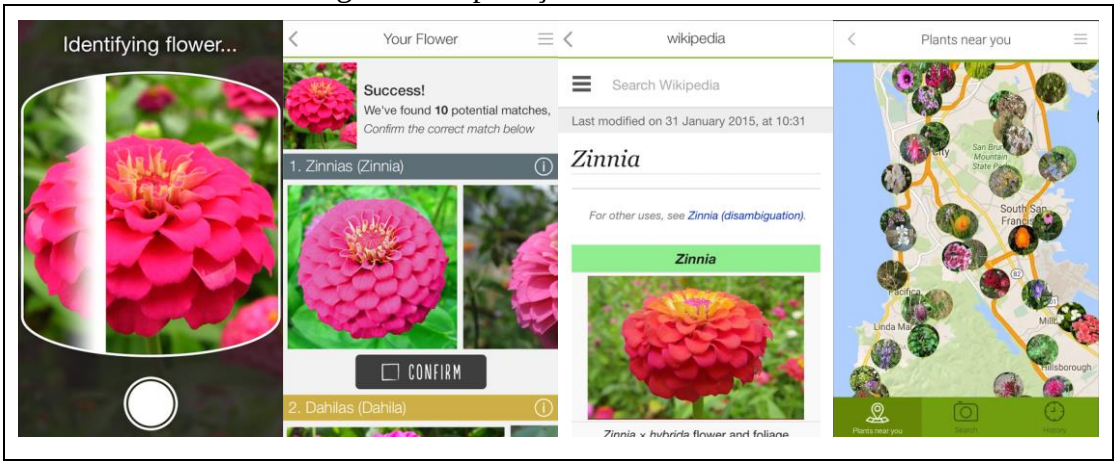
A aplicação permite ao usuário manter um histórico de todas as suas folhas reconhecidas junto ao local de onde a imagem foi obtida através do GPS do dispositivo, que pode ser visualizado no Google Maps. A aplicação também conta com uma base de dados de espécies com mais de 185 espécies de árvores cadastradas.

2.3.4 LikeThat Garden

O LikeThat Garden (JUSTVISUAL, 2015) é uma aplicação disponibilizada para as plataformas Android e iOS, que permite ao usuário identificar a espécie de uma flor via dispositivo móvel. Para isso, é necessário enviar uma imagem da flor a ser identificada. Esta

imagem pode ser obtida através da câmera ou através da galeria de imagens do dispositivo móvel. Após selecionar a imagem, ela é enviada para o servidor que realiza o reconhecimento da flor e retorna as espécies que possuem a maior quantidade de características semelhantes com a imagem enviada, e então o usuário pode selecionar qual é a espécie correta da flor. Para cada espécie retornada, o usuário pode abrir a página do Wikipedia para ver algumas informações da espécie, como reino, clado, ordem, família e gênero. A Figura 8 apresenta algumas telas da aplicação LikeThat Garden, como a tela para captura da flor, tela com resultado das espécies que tem similaridade com a imagem da flor enviada, tela com informações da espécie no Wikipedia e a tela que apresenta as espécies de flores que foram encontradas próximas ao local que o usuário se encontra.

Figura 8 – Aplicação LikeThat Garden



Fonte: JustVisual (2015).

A aplicação salva no próprio dispositivo um histórico com todas as imagens de flores que foram selecionadas para o reconhecimento. Quando o dispositivo não possuir conexão com a internet, a imagem selecionada pelo usuário também é salva nesse histórico, permitindo que o usuário realize o reconhecimento da espécie quando o dispositivo reestabelecer a conexão com a internet.

O LikeThat Garden (JUSTVISUAL, 2015) também possui uma funcionalidade que apresenta espécies de flores que foram encontradas próximas ao seu local atual. Essa funcionalidade utiliza o GPS do dispositivo para apresentar no Google Maps o local onde as espécies de flores foram encontradas.

3 DESENVOLVIMENTO

Este capítulo apresenta o protótipo atual e as etapas do desenvolvimento da solução proposta. A seção 3.1 apresenta o protótipo para identificação de cágados da espécie Phrynops Williamsi desenvolvido por Bertoldi (2016). Na seção 3.2 são apresentados os principais requisitos do protótipo desenvolvido. A seção 3.3 apresenta a especificação da solução, com diagramas de caso de uso e de pacotes e a sua arquitetura. A seção 3.4 detalha a implementação do protótipo, destacando as suas principais funcionalidades. E, por fim, a seção 3.5 apresenta os experimentos realizados e resultados obtidos.

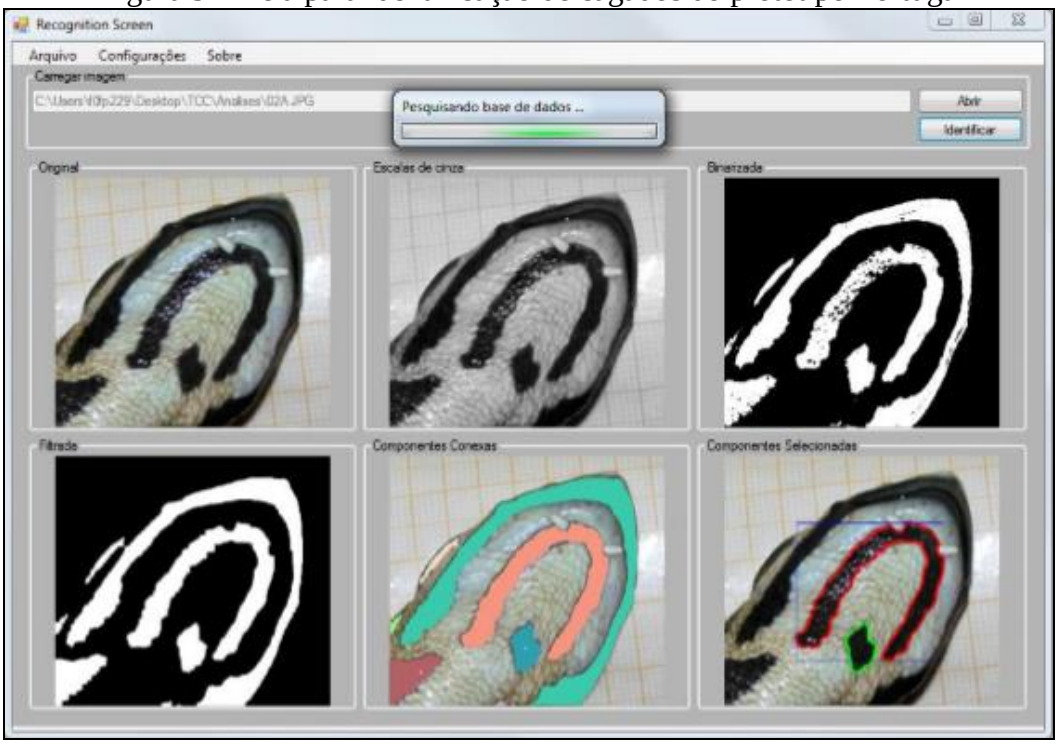
3.1 PROTÓTIPO ATUAL

O protótipo para identificação de cágados da espécie Phrynops Williamsi desenvolvido por Bertoldi (2016) tinha por objetivo criar um identificador único através de características extraídas de uma foto da listra em formato de ferradura e a listra circular localizadas na parte inferior da cabeça dos cágados dessa espécie. Foram necessárias algumas etapas para a geração do identificador único do cágado, sendo eles:

a) pré-processamento da imagem: inicialmente são aplicados alguns filtros na imagem submetida pelo usuário, tendo como objetivo descartar informações que não tem relevância para a geração do identificador e a remoção de possíveis imperfeições que dificultariam a identificação correta do cágado;
b) extração das componentes: nesta etapa é realizada a extração das componentes da imagem. Onde, cada elemento na cor preta é identificado como um componente;
c) seleção das listras: a partir das componentes identificadas na etapa anterior são feitas várias verificações, como comparação de tamanho, posição na imagem, circularidade e distância entre as listras. Essas verificações são feitas a fim de identificar a listra em formato de ferradura e a listra circular;
d) caracterização da forma: encontradas as listras, são calculados os seus descritores de Fourer. No cálculo dos descritores foram levadas em consideração as seguintes características: *bounding box* das listras, centroide, dispersão e circularidade. A partir dos testes realizados, foram estabelecidos vinte e três (23) descritores para as listras em formato de ferradura e seis (6) descritores para a listra em formato circular. É importante ressaltar que estes descritores são utilizados como identificador único e, são invariantes a rotação, escala e translação.

A Figura 9 apresenta a tela para identificação de cágados do protótipo Tortuga. Essa tela apresenta a imagem selecionada pelo usuário e os resultados da aplicação dos filtros para efetuar o reconhecimento das listras.

Figura 9 – Tela para identificação de cágados do protótipo Tortuga



Fonte: Bertoldi (2016, p. 65).

O protótipo foi desenvolvido na linguagem C# (C-Sharp), utilizando a biblioteca EmguCV para o processamento de imagens. Ele alcançou uma taxa de 85,71% de acerto para comparações intra-classe e 85,17% em comparações inter-classe. Entretanto, Bertoldi (2016), destaca que alguns problemas contribuíram para a diminuição das taxas de acerto, sendo eles: falta de nitidez nas fotos, o ângulo em que as listras se encontram na foto e a presença de reflexo na listra em formato de ferradura. E como extensão, Bertoldi (2016) propôs transformar o protótipo desenvolvido em uma aplicação para dispositivos móveis, permitir ao usuário selecionar a região da imagem que contém as listras e também retornar os 10 cágados mais semelhantes ao cágado informado, permitindo ao usuário realizar a classificação final.

3.2 REQUISITOS

Como a solução desenvolvida possui duas aplicações (Android e web) os requisitos foram separados por aplicação para facilitar a compreensão dos mesmos. A matriz de rastreabilidade dos Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) da aplicação Android são apresentados no Quadro 1 e Quadro 2 respectivamente e os da

aplicação web são apresentados no Quadro 3 e Quadro 4. Eles estão relacionados com os casos de uso apresentados na Figura 10 e Figura 11.

Quadro 1 – Requisitos funcionais da aplicação Android

Requisitos funcionais (RF)	**Casos de uso (UC)**
RF 01: permitir ao usuário capturar imagens das listras da cabeça do cágado a partir de um dispositivo Android	UC01
RF 02: permitir ao usuário selecionar a região da imagem que contém as listras da cabeça do cágado	UC01
RF 03: retornar ao usuário os cágados que possuem a maior quantidade de características semelhantes ao da imagem informada	UC01 e UC02
RF 04: salvar a posição geográfica de onde as imagens foram capturadas para manter o histórico do local ao qual o cágado foi encontrado	UC01, UC02, UC03 e UC04
RF 05: permitir ao usuário visualizar no mapa os locais onde os cágados foram localizados/identificados	UC06
RF 06: permitir ao usuário adicionar observações ao perfil do cágado	UC01, UC02, UC04 e UC05
RF 07: permitir a exportação e importação dos dados para realização de backup	UC07

Fonte: elaborado pelo autor.

Quadro 2 – Requisitos não funcionais da aplicação Android

Requisitos não funcionais (RNF)
RNF 01: utilizar método para reconhecimento de cágados desenvolvido por Bertoldi (2016)
RNF 02: ser desenvolvida para a plataforma Android
RNF 03: utilizar o ambiente de desenvolvimento Android Studio
RNF 04: utilizar o banco de dados SQLite para persistir os dados

Fonte: elaborado pelo autor.

Quadro 3 – Requisitos funcionais da aplicação web

Requisitos funcionais (RF)	**Casos de uso (UC)**
RF 01: permitir ao usuário visualizar os cágados reconhecidos pelas aplicações Android	UC01
RF 02: permitir ao usuário visualizar no mapa os locais onde os cágados foram localizados/identificados	UC3
RF 03: permitir ao usuário filtrar os cágados pelo endereço	UC01 e UC02

Fonte: elaborado pelo autor.

Quadro 4 – Requisitos não funcionais da aplicação web

Requisitos não funcionais (RNF)
RNF 01: utilizar Node.js como servidor web
RNF 02: utilizar o banco de dados MongoDB para persistir os dados
RNF 03: utilizar o ambiente de desenvolvimento Atom

Fonte: elaborado pelo autor.

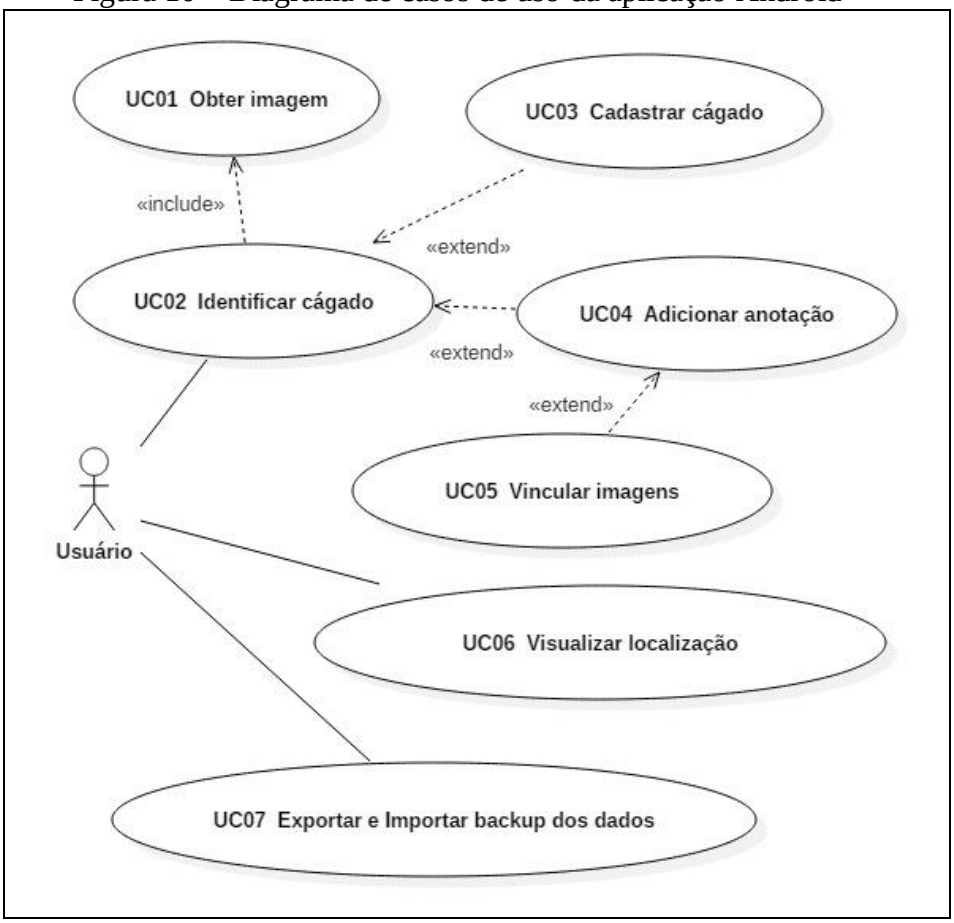
3.3 ESPECIFICAÇÃO

Nesta seção é descrita a especificação da solução. Inicialmente são apresentados os diagramas de caso de uso, seguido pela apresentação da arquitetura da solução. Por fim, são apresentados os diagramas de pacotes das aplicações Android e web. Para o desenvolvimento dos diagramas de caso de uso foi utilizada a ferramenta StarUML, e para os diagramas de pacotes o Enterprise Architect.

3.3.1 Casos de uso

O protótipo contém dez casos de uso, que representam as principais funcionalidades das aplicações desenvolvidas, e um ator denominado `usuário`. Para facilitar a visualização dos casos de uso, eles foram separados por aplicação. O diagrama da Figura 10 apresenta os casos de uso da aplicação Android, e a Figura 11 apresenta da aplicação web. Os diagramas foram feitos utilizando a Unified Modeling Language (UML).

Figura 10 – Diagrama de casos de uso da aplicação Android

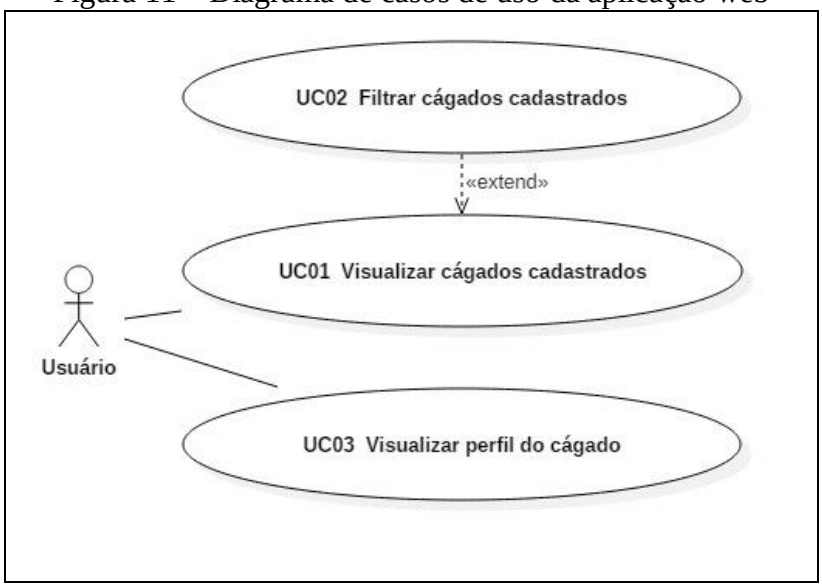


Fonte: elaborado pelo autor.

No caso de uso `UC01 Obter imagem` o usuário deve informar a imagem das listras do cágado. No caso de uso `UC02 Identificar cágado` são apresentados os cágados que possuem a maior quantidade de semelhanças com o cágado da imagem. Nesse momento, o usuário pode visualizar o perfil dos cágados retornados pela aplicação para ajudar na identificação. Caso nenhum dos cágados retornados seja correspondente ao cágado informado, o usuário poderá cadastrá-lo como um novo indivíduo. Este cadastro é realizado no `UC03 Cadastrar cágado`. No caso de uso `UC04 Adicionar anotação` é possível adicionar uma anotação sobre a captura do cágado. No caso de uso `UC05 Vincular imagens` o usuário pode vincular imagens do cágado em uma anotação. No caso de uso `UC06`

`Visualizar localização` o usuário pode ver no Google Maps os locais onde o cágado já foi reconhecido. No caso de uso `UC07 Exportar e importar backup dos dados` o usuário pode criar um backup dos dados exportando-os para o Google Drive e, caso for necessário é possível importá-los novamente. Na Figura 11 encontra-se o diagrama de casos de uso da aplicação web que tem por objetivo o compartilhamento dos dados coletados pela aplicação Android.

Figura 11 – Diagrama de casos de uso da aplicação web



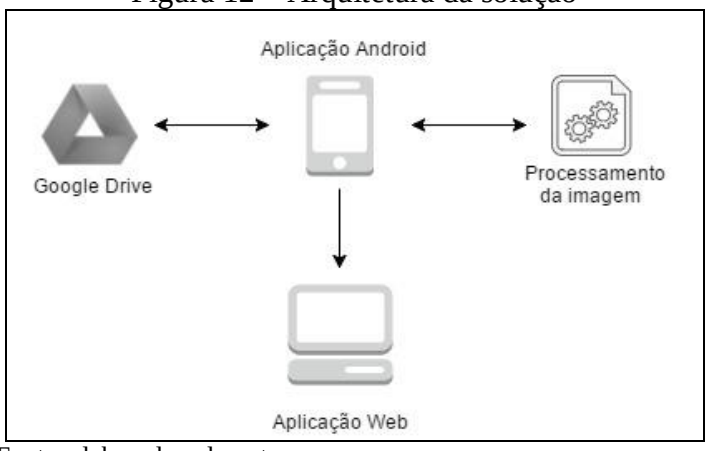
Fonte: elaborado pelo autor.

No caso de uso `UC01 Visualizar cágados cadastrados` o usuário pode visualizar todos os cágados reconhecidos em aplicações Android pela aplicação web. O caso de uso `UC02 Filtrar cágados cadastrados` o usuário pode aplicar um filtro por endereço nos cágados apresentados pela aplicação, facilitando assim a busca por algum local específico. No `UC03 Visualizar perfil do cágado` o usuário pode visualizar os detalhes de cada cágado existente na aplicação.

3.3.2 Arquitetura

A partir dos requisitos e casos de uso levantados, estabeleceu-se a seguinte arquitetura. A Figura 12 apresenta a arquitetura da solução desenvolvida.

Figura 12 – Arquitetura da solução



Fonte: elaborado pelo autor.

A aplicação Android é o ponto central entre os componentes utilizados. É através dela que o usuário persiste os dados na solução. Quando o usuário informar uma imagem do cágado na aplicação Android, ela utilizará a Application Programmin Interface (API) de processamento de imagens desenvolvida por Bertoldi (2016) para realizar a extração das características da imagem. Essa API retorna uma lista de identificadores (descritores), que são armazenados pelo aplicativo. Sempre que o usuário incluir um novo cágado, será feito a comparação a partir desses descritores. Além disso, ele também poderá incluir informações referentes ao perfil do cágado, como suas medidas corporais e um comentário com as anotações da captura. Esses dados ficam salvos no aplicativo, permitindo que o usuário os consulte quando for necessário.

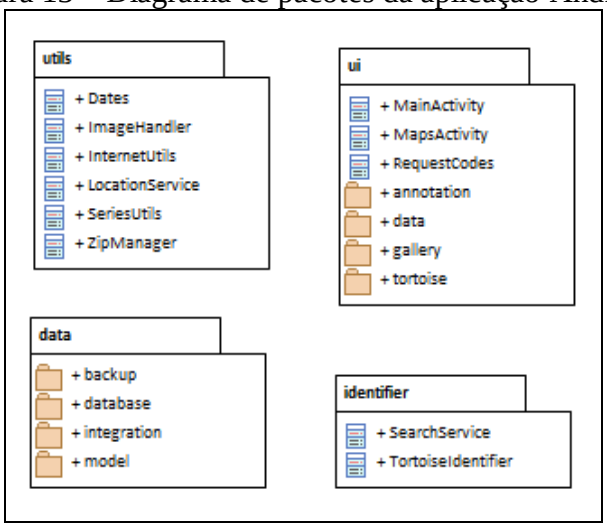
Os dados salvos no aplicativo também podem ser exportados para uma conta do Google Drive, criando um backup das informações. E, caso necessário, esses dados podem ser importados no aplicativo. Em paralelo, os dados salvos no aplicativo são enviados para uma aplicação web. No site, são compartilhadas as informações de todos os cágados que foram identificados por usuários do aplicativo.

A partir da arquitetura desenvolvida e para melhor organização do projeto, optou-se por separar as classes em pacotes, cada pacote agrupa classes com propósitos semelhantes que serão descritos nas próximas seções.

3.3.3 Diagramas de pacotes da aplicação Android

A Figura 13 apresenta o diagrama de pacotes referentes a aplicação Android que é responsável pelo cadastro, identificação, integração e backup dos dados coletados pelo usuário.

Figura 13 – Diagrama de pacotes da aplicação Android



Fonte: elaborado pelo autor.

Como pode ser observado no diagrama acima, alguns pacotes possuem outros pacotes. Estes pacotes agrupam as classes conforme sua especialidade. As próximas seções serão detalhadas as classes de cada pacote. A seção 3.3.3.1 descreve a função das classes dos pacotes `utils` e `identifier`. A seção 3.3.3.2 apresenta o pacote `ui` que contém as classes da interface de usuário. Na seção 3.3.3.3 é apresentada cada classe dos pacotes do pacote `data`.

3.3.3.1 Pacote utilitários e de identificação

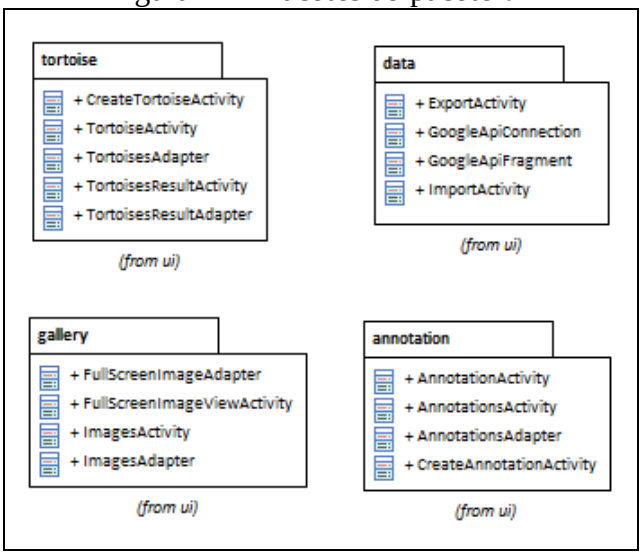
O pacote `util` agrupa classes com funções auxiliares, essas funções são utilizadas pelas outras classes do aplicativo. A classe `Dates` possui funções para conversão de objetos da classe `Data` para `String` e vice-versa. A classe `ImageHandler` possui funções para manipulação de arquivos de imagens, como criar uma nova imagem ou obter todas as imagens associadas a uma anotação. A classe `InternetUtils` possui uma função para verificar se o dispositivo móvel possui conexão com a internet. A classe `LocationService` possui funções para obter a localização atual do dispositivo ou obter o endereço de uma localização conforme sua latitude e longitude. A classe `SeriesUtils` possui funções para converter um *array* de `Double` para `String` e vice-versa. A classe `ZipManager` contém um método que compacta uma lista de arquivos em um único arquivo `zip`.

O pacote `identifier` é responsável pelas classes que realizam a identificação dos cágados com base na imagem das listras que estão da cabeça dos mesmos. A classe `TortoiseIdentifier` possui os métodos para realizar a identificação dos cágados e para realizar a comparação de dois identificadores afim de definir a sua semelhança. A classe `SearchService` possui um método que encontra os cágados que possuem a maior semelhança com o cágado que foi selecionado pelo usuário.

3.3.3.2 Pacote de interface de usuário

O pacote `ui` possui classes do tipo *activity*, que são as telas apresentadas pelo aplicativo. Optou-se por organizar as classes em pacotes conforme suas funcionalidades. A Figura 14 apresenta as classes de cada pacote.

Figura 14 – Pacotes do pacote `ui`



Fonte: elaborado pelo autor.

O pacote `tortoise` possui as telas que estão relacionadas as informações dos cágados, como cadastro e edição do cágado, perfil do cágado e a tela que apresenta o percentual de semelhança entre os cágados. A classe `CreateTortoiseActivity` permite ao usuário realizar o cadastro de um novo cágado, ou editar as informações de um cágado já cadastrado. A classe `TortoiseActivity` apresenta as informações de um cágado cadastrado. A classe `TortoiseAdapter` apresenta todos os cágados cadastrados em um componente `ListView` e é utilizada pela classe `MainActivity`. A classe `TortoisesResultActivity` apresenta os cágados que tem o maior percentual de semelhança com o cágado selecionado pelo usuário. Através dessa tela também é possível ir para a tela de cadastro de cágados. A classe `TortoisesResultAdapter` apresenta o cágado e o seu percentual de semelhança com o cágado selecionado pelo usuário, os cágados são apresentados em um componente `ListView` na classe `TortoisesResultActivity`.

O pacote `gallery` possui as telas relacionadas as imagens da galeria de um acompanhamento, como a tela que apresenta todos as imagens e a tela que apresenta a imagem em tela cheia. A classe `FullScreenImageAdapter` é responsável por permitir o usuário navegar entre as imagens da galeria em modo tela cheia. A classe `FullScreenImageViewActivity` apresenta uma imagem em tela cheia. A classe `ImagesActivity` apresenta todas as imagens da galeria ordenadas por data de criação. A

classe `ImagesAdapter` apresenta as imagens com a sua data de criação em um componente `ListView`, esse componente é apresentado na classe `ImagesActivity`.

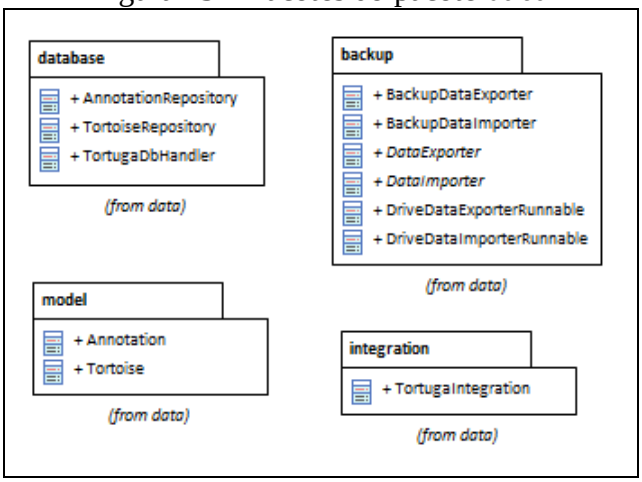
O pacote `data` possui as telas para exportação e importação dos dados da aplicação. Quando o usuário clica no botão para exportar os dados da aplicação é a classe `ExportActivity` que inicia o processo para exportar os dados para o Google Drive. A classe `ImportActivity` é responsável por iniciar o processo de importação de dados do Google Drive. As classes `GoogleApiFragment` e `GoogleApiConnection` são utilizadas pelas classes `ExportActivity` e `ImportActivity` para apresentar as telas do Google Drive e para realizar a conexão com os serviços do Google.

O pacote `annotation` possui as telas relacionadas aos acompanhamentos dos cágados como cadastro de acompanhamento e visualização dos acompanhamentos. A classe `AnnotationActivity` apresenta as informações de um acompanhamento cadastrado. A classe `AnnotationsActivity` apresenta os acompanhamentos de um cágado ordenados por data. A classe `AnnotationsAdapter` apresenta a data de criação do acompanhamento em um componente `ListView` e é utilizada pela classe `AnnotationsActivity`. A classe `CreateAnnotationActivity` permite ao usuário realizar o cadastro de um acompanhamento ou editar um acompanhamento já cadastrado.

3.3.3.3 Pacote de dados

O pacote `data` é responsável pelas classes de acesso a base de dados, classes de modelos de dados, classes de exportação e importação de backup e a classe para integração com a aplicação web. Na Figura 15 é possível observar as classes de cada pacote do pacote data.

Figura 15 – Pacotes do pacote `data`



Fonte: elaborado pelo autor.

O pacote `database` é responsável pelas classes de acesso a base de dados. As classes `AnnotationRepository` e `TortoiseRepository` são responsáveis por inserir, alterar, obter e excluir os acompanhamentos e cágados da base de dados. A classe `TortugaDbHandler` estende a classe `SQLiteOpenHelper` do Android, ela é responsável pela criação e versionamento da base de dados.

O pacote `model` contém as classes que representam as estruturas de dados do aplicativo. A classe `Annotation` representa o acompanhamento de um cágado, essa classe possui os campos relevantes a serem anotados em cada acompanhamento realizado. A classe `Tortoise` representa um cágado, essa classe possui um campo que é o identificador do cágado e outro que armazena o seu sexo.

O pacote `backup` contém as classes que realizam a exportação e importação dos dados da aplicação para o Google Drive. A classe `BackupDataExporter` implementa a interface `DataExporter` e é responsável por exportar os arquivos da base de dados em formato `json`, compactar as imagens em um único arquivo `zip` e escreves os *bytes* desse arquivo no `OutputStream` recebido pela classe `DriveDataExporterRunnable`. A classe `DriveDataExporter` implementa a interface `Runnable` do Java, isso permite que o processo para exportação seja executado em uma *thread* secundaria, não travando a interface do aplicativo. Essa classe chama a classe `BackupDataExporter` passando o `OutputStream` onde os *bytes* dos arquivos devem ser escritos para serem enviados para o Google Drive.

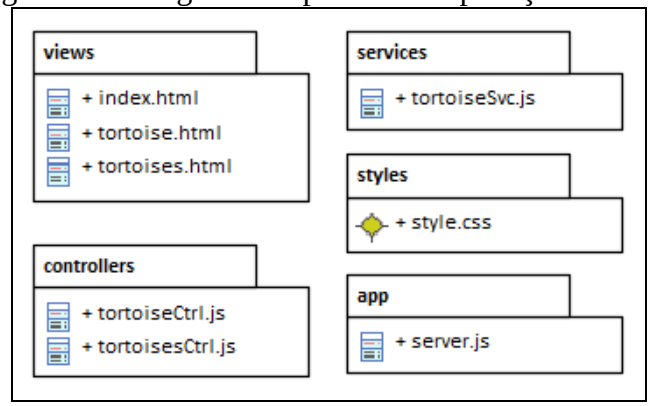
A classe `BackupDataImporter` implementa a interface `DataImporter` que é responsável por ler o `InputStream` que contém os *bytes* dos arquivos `json` e `zip` que foram enviados para o Google Drive pela rotina de exportação. A classe `BackupDataImporter` é chamada pela classe `DriveDataImporterRunnable` que implementa a interface `Runnable` do Java, ela é responsável por criar o `InputStream` que possui os *bytes* dos arquivos no Google Drive.

O pacote `integration` possui uma única classe que é responsável por enviar os cágados para a aplicação web. A classe `TortugaIntegration` envia os dados da base de dados para a aplicação web que armazena os dados dos cágados de todos dispositivos que utilizam o aplicativo.

3.3.4 Diagrama de pacotes da aplicação web

A Figura 16 apresenta o diagrama de pacotes referentes a aplicação web que é responsável pelo compartilhamento das informações dos cágados identificados pelos aplicativos.

Figura 16 – Diagrama de pacotes da aplicação Android



Fonte: elaborado pelo autor.

O pacote `views` contém os arquivos HyperText Markup Language (HTML) com as telas da aplicação web. O arquivo `index.html` possui o conteúdo comum entre as telas, contendo o cabeçalho da aplicação web. O arquivo `tortoise.html` é responsável por apresentar a tela com as informações do perfil do cágado, apresentando os locais em que o cágado foi reconhecido e o seu identificador e sexo cadastrado pelo usuário. O arquivo `tortoises.html` é responsável por apresentar todos os cágados cadastrados em cartões, cada cartão apresenta a imagem do perfil do cágado e o seu endereço.

O pacote `controllers` possui os arquivos JavaScript que são responsáveis por carregar os dados das telas da aplicação. O arquivo `tortoiseCtrl.js` é responsável por carregar as informações do perfil do cágado, nele é criada a *view* do Google Maps e adicionado os marcadores com os locais em que o cágado já foi reconhecido. O arquivo `tortoisesCtrl.js` é responsável por carregar todos os cágados cadastrados na solução. Ele também é responsável por filtrar os cágados pelo endereço informado pelo usuário no campo do cabeçalho da aplicação.

O pacote `services` contém um único arquivo JavaScript. O arquivo `tortoiseSvc.js` é utilizado pelas classes do pacote controllers para obter os dados dos cágados cadastrados. Esse arquivo realiza a integração entre os serviços do *back-end* e *front-end* da aplicação. Ele possui os métodos para obter todos os cágados, onde é possível passar o filtro de endereço, e o método para obter os dados de um único cágado pelo seu Universally Unique Identifier (UUID).

O pacote `styles` contém um único arquivo CSS, o `style.css`. Esse arquivo contém todos os estilos CSS utilizados nas telas da aplicação web.

O pacote `app` contém o arquivo `server.js`. Esse arquivo é o servidor do *back-end* da aplicação. Ele possui os serviços para cadastrar, excluir, atualizar e obter os cágados na aplicação web. Ele também é responsável por retornar o conteúdo HTML das telas da aplicação web. Esse arquivo também é utilizado pela aplicação Android, para realização do cadastro dos cágados, e, pelo arquivo `tortoiseSvc.js` do *front-end*, onde é realizada a busca e filtragem dos cágados.

3.4 IMPLEMENTAÇÃO

Nesta seção são apresentadas as técnicas e ferramentas utilizadas para implementar o presente projeto. Também é apresentado um diagrama de atividades com os passos necessários para realizar o acompanhamento dos cágados através da solução desenvolvida.

3.4.1 Técnicas e ferramentas utilizadas

No desenvolvimento do aplicativo foi utilizada a linguagem de programação Java, no Integrated Development Environment (IDE) Android Studio na versão 2.3.1. O Android Studio é a IDE oficial do Google para desenvolvimento de aplicativos Android. Para a implementação das funcionalidades que utilizam o Google Maps foi utilizado o Google Places API na versão 10.2.1.

Na implementação das funcionalidades de exportação e importação dos dados da aplicação foram utilizados o Google Actions, Base Client Library API e Google Drive API, ambos na versão 10.2.1. Para a conversão dos objetos Java para `json` foi utilizada a biblioteca `Google Gson` na versão 2.2.4.

Para o envio dos dados dos cágados para a aplicação web foi utilizada a biblioteca `Volley` na versão 1.0.0. Essa biblioteca também é disponibilizada pelo Google e é indicada na documentação oficial do Android quando se é necessário realizar o envio de dados pela internet.

Para realizar a migração do protótipo para identificação de cágados da espécie Phrynops Williamsi desenvolvido por Bertoldi (2016) na linguagem C# foi utilizada a biblioteca `OpenCV4Android` na versão 3.2.0.

No desenvolvimento da aplicação web foi utilizado Node.js no *back-end* e o *framework* AngularJS na versão 1.6.4 para o *front-end*. A aplicação web foi desenvolvida no IDE Atom na versão 1.16.0.

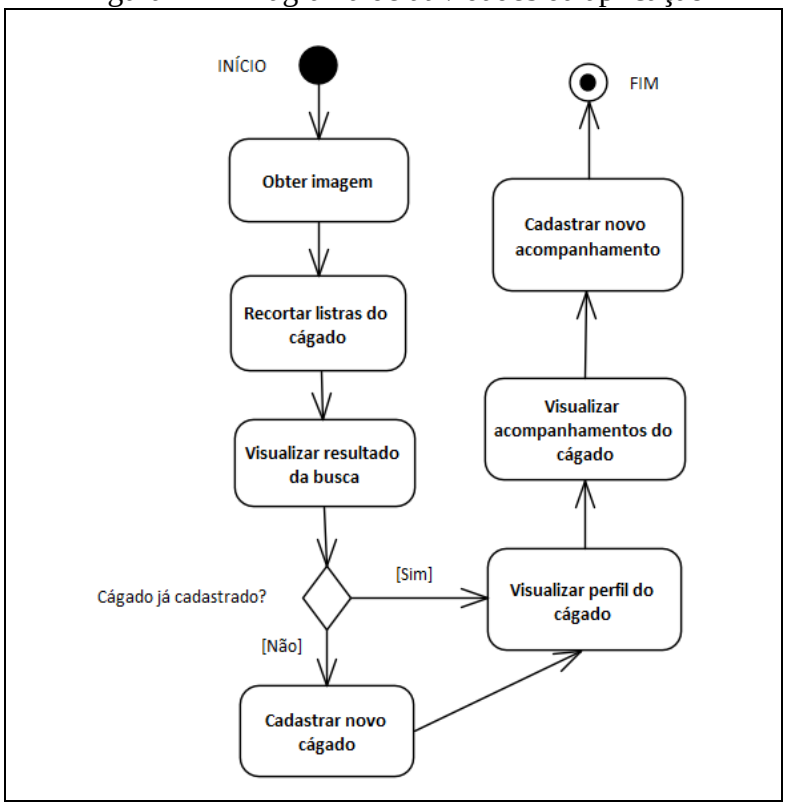
No desenvolvimento do *front-end* também foram utilizados HTML, CSS e o framework Bootstrap na versão 4.0.0-alpha.

No *back-end* foi criada uma REST API utilizando a biblioteca `Express` na versão 4.13.2. Para armazenar os dados foi utilizado o banco de dados NoSQL MongoDB. Para utilização do banco de dados foi utilizada a biblioteca `Mongojs` na versão 2.4.0.

3.4.2 Diagrama de atividades

Para melhor compreensão do fluxo da aplicação, o diagrama de atividades da Figura 17 demonstra os passos necessários para o usuário realizar o acompanhamento/monitoramento de um cágado.

Figura 17 – Diagrama de atividades da aplicação



Fonte: elaborado pelo autor.

A primeira atividade é a obtenção da imagem das listras do cágado. Posteriormente o usuário deve recortar a área da imagem que possui as listras da cabeça do cágado. Nesse momento, a aplicação realizará uma busca na base de dados afim de encontrar os cágados que possuem a maior semelhança com o cágado da imagem selecionada. Ao final da busca, os cágados são apresentados ao usuário. Nesse momento é possível selecionar um cágado já cadastrado na aplicação ou realizar o cadastro de um novo indivíduo.

Após a realização do cadastro é apresentada a tela com o perfil do cágado. Nela, é possível ver o histórico de captura/recaptura do cágado (histórico de acompanhamento).

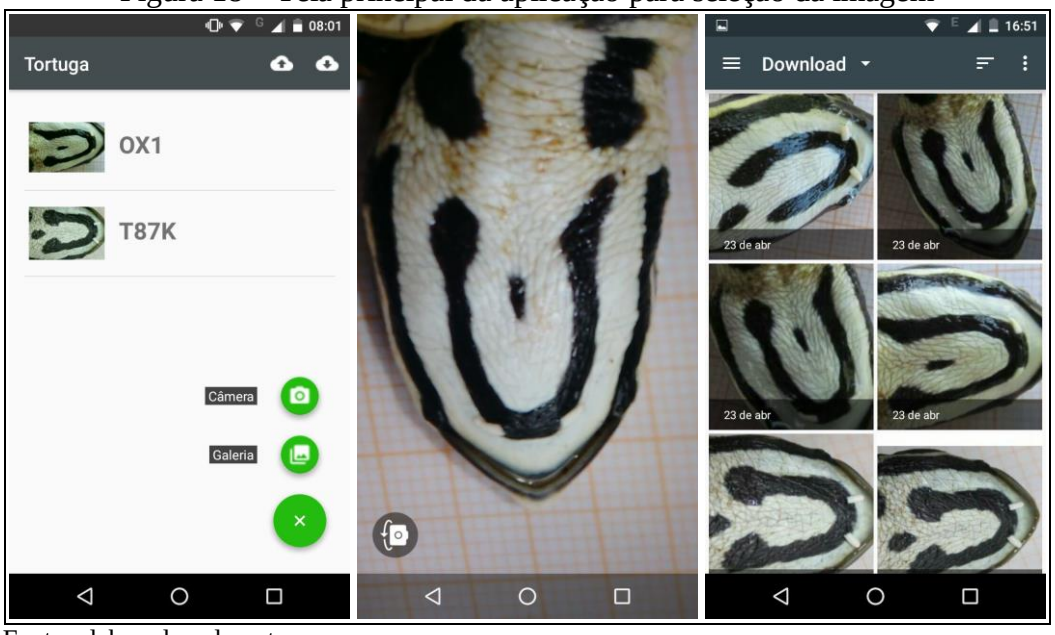
Na tela que mostra o histórico de acompanhamento é apresentada a opção para adicionar um novo registro de captura. No cadastro de acompanhamento é apresentada a localização em que o dispositivo móvel estava quando o usuário visualizou a tela com os resultados da busca e os campos para realizar o acompanhamento do cágado, sendo eles campos para as suas medidas corporais e um campo para observações.

Ao finalizar o cadastro de acompanhamento, ele é salvo com data e hora em que o usuário reconheceu o cágado e a localização do dispositivo móvel. Nas próximas seções, cada uma das etapas citadas acima será detalhada através de telas e trechos de códigos.

3.4.3 Seleção da imagem

O primeiro passo realizado para a identificação de um cágado é selecionar a imagem das listras encontradas abaixo da cabeça, conforme mostra a Figura 18. Essa imagem pode ser obtida de duas maneiras, através da câmera do dispositivo móvel ou da galeria de imagens.

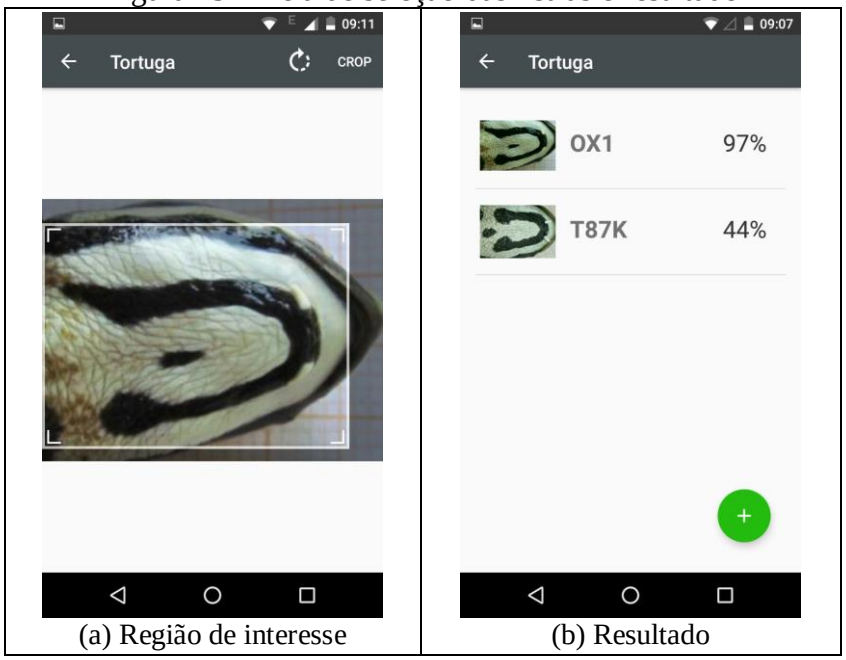
Figura 18 – Tela principal da aplicação para seleção da imagem



Fonte: elaborado pelo autor.

Após selecionar a imagem, o usuário pode recortar a área da imagem que possui as duas listras localizadas abaixo da cabeça do cágado. Para facilitar a seleção das listras é apresentado um quadrado que permite ajustar a região que será recortada da imagem. Também é possível ampliar e girar a imagem. Este processo é demonstrado na Figura 19a.

Figura 19 – Tela de seleção das listras e resultado



(a) Região de interesse (b) Resultado

Fonte: elaborado pelo autor.

Após a seleção das listras é apresentada uma tela com os cágados já reconhecidos pelo aplicativo (Figura 19b) que possuem a maior quantidade de características semelhantes ao cágado da imagem selecionada. O usuário pode optar por vincular ou por cadastrar um novo cágado. Isto dependerá da sua percepção, ou seja, se ele entender que nenhum dos cágados apresentados é semelhante ao cágado da imagem selecionada, optará por um novo cadastro.

O percentual de semelhança entre os cágados é obtido através do método para identificação de cágados desenvolvido por Bertoldi (2016). Como descrito na seção 3.1 o método foi desenvolvido na linguagem de programação C# e utilizou a biblioteca EmguCV para realizar o processamento da imagem do cágado. Para permitir a identificação dos cágados pela aplicação Android foi necessário realizar a migração do método desenvolvido por Bertoldi (2016) para a linguagem Java. Para realizar o processamento da imagem das listras do cágado foi utilizada a biblioteca OpenCV4Android. Como ambas bibliotecas são um *wrapper* da biblioteca OpenCV não ouve grandes adaptações no código migrado, apenas algumas particularidades de cada linguagem de programação.

No método `onCreate` da tela `ResultActivity` é executado o método `getCurrentLocation` da classe `LocationService`. Esse método é o responsável por obter a localização atual do dispositivo móvel. Essa localização é salva na base de dados quando o usuário cadastra um novo acompanhamento para um dos cágados apresentados na tela de resultado da busca, ou quando o usuário opta por cadastrar um novo cágado. No Quadro 5 é possível observar a implementação do método que obtém a localização atual do dispositivo.

Quadro 5 – Código do método `getCurrentLocation` da classe `LocationService`

```
1     public Location getCurrentLocation() {
2         if (ContextCompat.checkSelfPermission(ctx,
3 android.Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
4                 ContextCompat.checkSelfPermission(ctx,
5 android.Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
6             Toast.makeText(ctx, "First enable LOCATION ACCESS in settings.",
7 Toast.LENGTH_LONG).show();
8         }
9
10        LocationManager service = (LocationManager)
11 ctx.getSystemService(Context.LOCATION_SERVICE);
12        List<String> providers = service.getProviders(true);
13
14        Location bestLocation = null;
15        for (String provider : providers) {
16            Location l = service.getLastKnownLocation(provider);
17            if (l == null) {
18                continue;
19            }
20            if (bestLocation == null || l.getAccuracy() < bestLocation.getAccuracy()) {
21                bestLocation = l;
22            }
23        }
24        return bestLocation;
25    }
```

Fonte: elaborado pelo autor.

Na linha 12 é obtida a lista de serviços que podem prover a localização do dispositivo móvel, é informado o parâmetro `true` para o método `getProviders` para indicar que devem ser apenas retornados serviços que estão ativos no momento. Logo abaixo é iterado em cada serviço retornado pelo método anterior, para cada serviço é executado o método `getLastKnownLocation`. Esse método obtém a última localização conhecida (linha 16) pelo serviço. Caso o serviço retorne uma localização, na linha 20 ela é comparada com a melhor localização obtida até o momento. Essa comparação é realizada através do método `getAccuracy` que retorna a precisão da localização em metros, então caso a precisão da localização atual seja melhor ela substitui a antiga localização. Após iterar sobre todos os serviços então a linha 24 retorna à localização que possui a melhor precisão.

No método `onCreate` da tela `ResultActivity` também são calculados os descritores utilizados para realizar a identificação do cágado. Para isso é executado o método `getFourierSeries` da classe `TortoiseImageProcessing` que recebe por parâmetro o caminho da imagem selecionada pelo usuário. Após calcular os descritores do cágado selecionado pelo usuário é executado o método `loadResults` da classe `ResultActivity`. Esse método é responsável por obter os cágados que possuem a maior quantidade de semelhanças com o cágado informado e apresenta-los para o usuário. O código do método `loadResults` pode ser observado no Quadro 6.

Quadro 6 – Código do método `loadResults` da classe `ResultActivity`

```
1     private void loadResults() {
2         List<Pair<Tortoise, Double>> result = SearchService.find(this, series, 3);
3         ListAdapter adapter = new TortoisesResultAdapter(this, result);
4
5         ListView listView = (ListView) findViewById(R.id.mainListView);
6         listView.setAdapter(adapter);
7     }
```

Fonte: elaborado pelo autor.

Na linha 2 é executado o método `find` da classe `SearchService`. Esse método recebe como parâmetro os descritores do cágado que foi selecionado pelo usuário e a quantidade de cágados que devem ser retornados. Na linha 3 é criado um `TortoisesResultAdapter` que recebe em seu construtor a lista de cágados retornados pelo método `find`. Esse adapter é responsável por apresentar ao usuário os cágados reconhecidos pela aplicação e seu percentual de semelhança com o cágado selecionado. No Quadro 7 é possível ver o código do método `find`.

Quadro 7 – Código do método `find` da classe `SearchService`

```
1  public static List<Pair<Tortoise, Double>> find(Context ctx, double[] series, long size) {
2      TortoiseRepository repository = TortoiseRepository.getInstance(ctx);
3
4      List<Pair<Long, double[]>> all = repository.findSeries();
5      List<Pair<Long, Double>> diffs = new ArrayList<>(all.size());
6
7      for (Pair<Long, double[]> pair : all) {
8          diffs.add(new Pair<>(pair.first, TortoiseImageProcessing.diff(series, pair.second)));
9      }
10
11     Collections.sort(diffs, ascending);
12
13     if (diffs.size() > size) {
14         diffs = diffs.subList(0, (int) size);
15     }
16
17     List<Pair<Tortoise, Double>> tortoises = new ArrayList<>();
18     for (int i = 0; i < diffs.size(); i++) {
19         tortoises.add(new Pair<>(repository.find(diffs.get(i).first), diffs.get(i).second));
20     }
21     return tortoises;
22 }
```

Fonte: elaborado pelo autor.

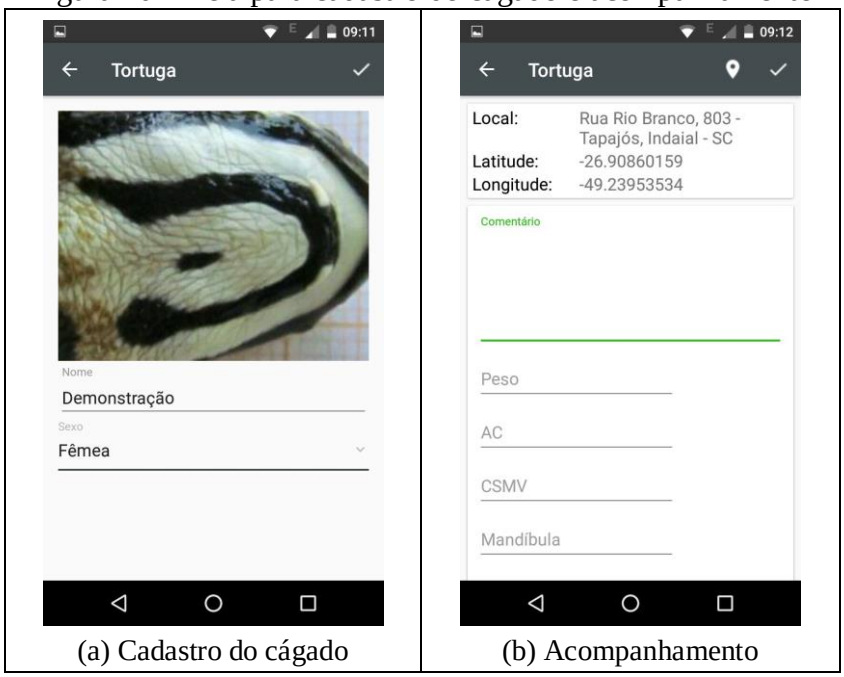
Na linha 4 é executado o método `findSeries` da classe `TortoiseRepository`. Ele é responsável por retornar um `Pair` que possui o identificador do cágado na base de dados e seus descritores. Na linha 7 é iterado sobre o resultado do método `findSeries` e na linha 8 comparado os seus descritores com os do cágado selecionado pelo usuário. Para cada registro da base é calculada sua semelhança com o cágado selecionado pelo usuário e salvo esse valor na lista `diffs` criada na linha 5. Essa lista armazena um `Pair` com o identificador do cágado e a sua semelhança com o cágado selecionado pelo usuário. Na linha 11 é realizada a ordenação da lista `diffs` deixando os cágados com maior semelhança no início da lista. Na linha 13 é verificado se a lista possui mais cágados do que a quantidade solicitada por parâmetro, caso existam mais cágados é criada uma sublista (linha 14) com a quantidade de cágados que foi

solicitado. Nas próximas linhas é obtido o objeto `Tortoise` dos cágados que possuem a maior quantidade de semelhanças com o cágado selecionado pelo usuário e então na linha 21 é retornada a lista de `Pair` que possui o `Tortoise` e um `Double` com a sua semelhança com o cágado selecionado pelo usuário.

3.4.4 Cadastro do cágado e acompanhamento

O cadastro do cágado apresenta a imagem que foi selecionada pelo usuário, um campo para o usuário informar um identificador para o cágado e um campo para o sexo. Neste último é possível optar entre os valores: Macho, Fêmea e Juvenil. A Figura 20a apresenta a tela para cadastro do cágado.

Figura 20 – Tela para cadastro do cágado e acompanhamento



(a) Cadastro do cágado (b) Acompanhamento

Fonte: elaborado pelo autor.

O cadastro de acompanhamento (Figura 20b) abrange as informações de: local, comentário, peso, comprimento curvilíneo da carapaça, largura curvilíneo da carapaça, comprimento do plastrão, largura do plastrão, comprimento retilíneo da carapaça, largura retilínea da carapaça, altura do casco, comprimento sutura média ventre, comprimento da primeira cauda, comprimento da segunda cauda, largura da mandíbula e número do tombo.

No método `onCreate` da tela `CreateAnnotationActivity` é executado o método `loadAnnotation`. Esse é o método responsável por apresentar a localização de onde o cágado foi encontrado. O trecho de código que obtém e apresenta a localização do dispositivo móvel pode ser visto no Quadro 8.

Quadro 8 – Código do método `loadAnnotation` da classe `CreateAnnotationActivity`

```
1   private void loadAnnotation() {
2       long id = getIntent().getLongExtra("annotationId", -1);
3       if (id < 0) {
4           double latitude = getIntent().getDoubleExtra("latitude", 0);
5           double longitude = getIntent().getDoubleExtra("longitude", 0);
6           if (latitude != 0 && longitude != 0) {
7               lblLatitude.setText(String.valueOf(latitude));
8               lblLongitude.setText(String.valueOf(longitude));
9
10              if (InternetUtils.hasConnection(this)) {
11                  LocationService locationService = LocationService.getInstance(this);
12                  String address = locationService.getAddress(latitude, longitude);
13                  if (address != null) {
14                      lblAddress.setText(address);
15                  }
16              }
17          }
18          return;
19      }
20
21      //Código
22  }
```

Fonte: elaborado pelo autor.

Na linha 3 é verificado se o usuário está realizando o cadastro de um novo acompanhamento, pois se o acompanhamento já estiver cadastrado não é necessário obter a localização do dispositivo. Caso o acompanhamento ainda não esteja cadastrado então é obtida a latitude (linha 4) e longitude (linha 5), essas informações são obtidas no momento em que o usuário visualiza os resultados de uma busca, conforme descrito na seção 3.4.3.

Nas linhas 7 e 8 são apresentadas a latitude e a longitude para o usuário. Na linha 10 é verificado se o dispositivo móvel possui conexão com a internet, caso o dispositivo esteja conectado na internet no momento então é executado o método `getAddress` da classe `LocationService`. Esse método é o responsável por obter o endereço do local onde o cágado foi reconhecido. Caso o endereço seja retornado então ele é apresentado para o usuário (linha 14). O código do método `getAddress` é apresentado no Quadro 9.

Quadro 9 – Código do método `getAddress` da classe `LocationService`

```
1   public String getAddress(double latitude, double longitude) {
2       Geocoder geocoder = new Geocoder(ctx);
3
4       try {
5           List<Address> adr = geocoder.getFromLocation(latitude, longitude, 1);
6           if (adr.isEmpty()) {
7               return null;
8           }
9           return adr.get(0).getAddressLine(0) + ", " + adr.get(0).getAddressLine(1);
10      } catch (Exception e) {
11          e.printStackTrace();
12      }
13      return null;
14  }
```

Fonte: elaborado pelo autor.

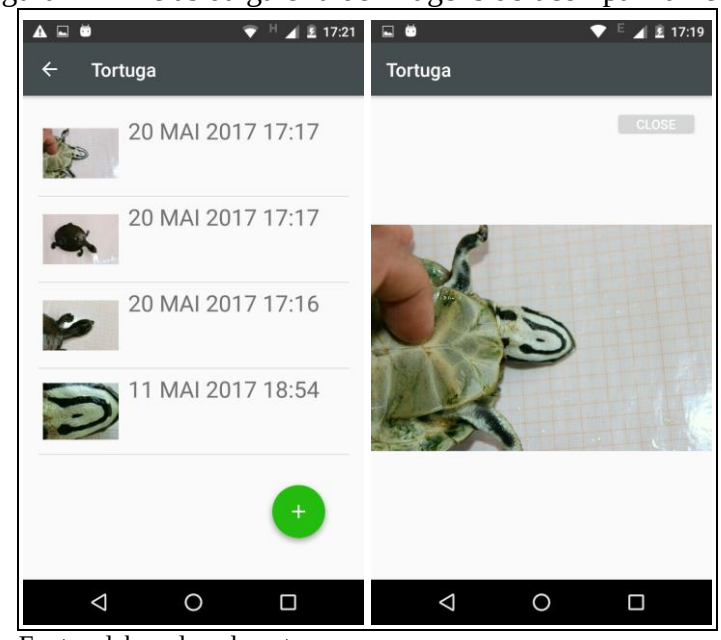
Na linha 2 é criada uma instância da classe `Geocoder`, essa classe é disponibilizada pelo Android e serve para transformar o endereço de uma rua em uma localização (latitude e longitude) ou vice-versa. Na linha 5 é executado o método `getFromLocation`, da classe

`Geocoder`. Esse método recebe a latitude e a longitude do local e a quantidade de endereços que devem ser retornados. Nesse caso, apenas um. Caso seja retornado um endereço então o método `getAddress` retorna as duas primeiras linhas desse endereço.

3.4.5 Histórico de imagens

Cada acompanhamento possui uma galeria de imagens, para que o usuário possa salvar todas as imagens do cágado que foram registradas durante o acompanhamento. Dessa forma é criado um histórico de imagens por acompanhamento, permitindo ao usuário identificar as mudanças que ocorreram com o cágado no intervalo entre os acompanhamentos. Para adicionar uma imagem na galeria o usuário pode utilizar à câmera do dispositivo móvel ou selecionar direto da galeria de imagens do dispositivo, da mesma forma que é obtida a imagem para a identificação de um cágado. Na Figura 21 é possível ver as telas da galeria de imagens do acompanhamento.

Figura 21 – Telas da galeria de imagens do acompanhamento



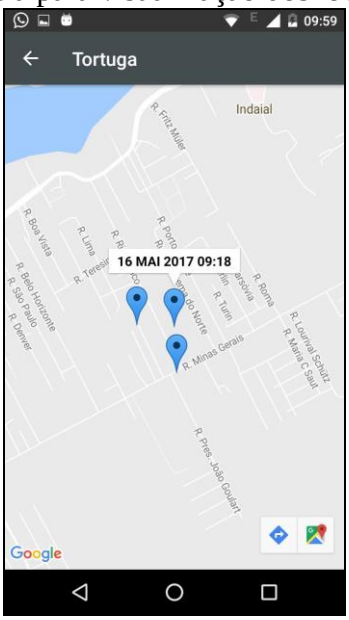
Fonte: elaborado pelo autor.

Na galeria de imagens do acompanhamento é apresentada uma miniatura da imagem e ao seu lado a data em que ela foi adicionada na galeria. As imagens estão ordenadas de forma descendente pela data em que foram adicionadas a galeria. Ao clicar em uma miniatura da galeria ela é apresentada a imagem em modo tela cheia, permitindo ao usuário visualizar os detalhes da imagem. O modo tela cheia permite ao usuário navegar entre as imagens com o movimento de deslizar dos dedos na tela.

3.4.6 Visualização dos locais

Cada acompanhamento cadastrado possui a localização de onde o cágado foi encontrado. Essa localização é salva automaticamente ao se criar um novo acompanhamento. Porém é permitido ao usuário alterar essa localização posteriormente, desde que o dispositivo móvel tenha conexão com a internet. Para visualizar os locais que o cágado foi encontrado é apresentada a opção `Locais` no perfil do cágado. Ao clicar nessa opção é aberto o Google Maps com marcadores nos locais em que o cágado foi encontrado. Conforme mostra a Figura 22, ao selecionar o marcador é apresentada a data e hora em que foi cadastrado o acompanhamento.

Figura 22 – Tela para visualização dos locais de captura



Fonte: elaborado pelo autor.

No método `onCreate` da classe `MapsActivity` é obtido o `SupportMapFragment`. Essa classe é responsável por apresentar a tela do Google Maps na aplicação. Para a utilização do Google Maps foi utilizada a Google Places API disponibilizada pelo Google na versão 10.2.1. No Quadro 10 é possível observar o código do método `onCreate` da classe `MapsActivity`.

Quadro 10 – Código do método `onCreate` da classe `MapsActivity`

```
1     @Override
2     protected void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         getSupportActionBar().setDisplayHomeAsUpEnabled(true);
5
6         tortoiseId = getIntent().getLongExtra("tortoiseId", -1);
7         if (tortoiseId < 0) {
8             throw new RuntimeException("Missing tortoiseId");
9         }
10
11        setContentView(R.layout.activity_maps);
12
13        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
14                .findFragmentById(R.id.map);
15        mapFragment.getMapAsync(this);
16    }
```

Fonte: elaborado pelo autor.

Na linha 13 é obtido o `SupportMapFragment` da *activity* que deve renderizar o Google Maps. Na linha 14 é executado o método `getMapAsync` que retorna o `GoogleMap` quando ele estiver pronto. Para executar esse método é necessário que a classe `MapsActivity` implemente a *interface* `OnMapReadyCallback`. Essa *interface* possui o método `onMapReady` que é executado quando o `GoogleMap` estiver pronto. No Quadro 11 é possível ver o código do método `onMapReady` implementado pela classe `MapsActivity`.

Quadro 11 – Código do método `onMapReady` da classe `MapsActivity`

```
1  @Override
2  public void onMapReady(GoogleMap googleMap) {
3      AnnotationRepository annotationRepository = AnnotationRepository.getInstance(this);
4
5      List<Annotation> annotations = annotationRepository.findAll(tortoiseId);
6      if (annotations.isEmpty()) {
7          return;
8      }
9
10     LatLng position = null;
11     for (Annotation annotation : annotations) {
12         position = new LatLng(annotation.getLatitude(), annotation.getLongitude());
13
14         SimpleDateFormat df = new SimpleDateFormat("dd MMM yyyy HH:mm");
15
16         MarkerOptions options = new MarkerOptions()
17             .position(position)
18             .title(df.format(annotation.getCreatedAt()).toUpperCase())
19            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE));
20
21         googleMap.addMarker(options);
22     }
23     googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(position, 15f));
24 }
```

Fonte: elaborado pelo autor.

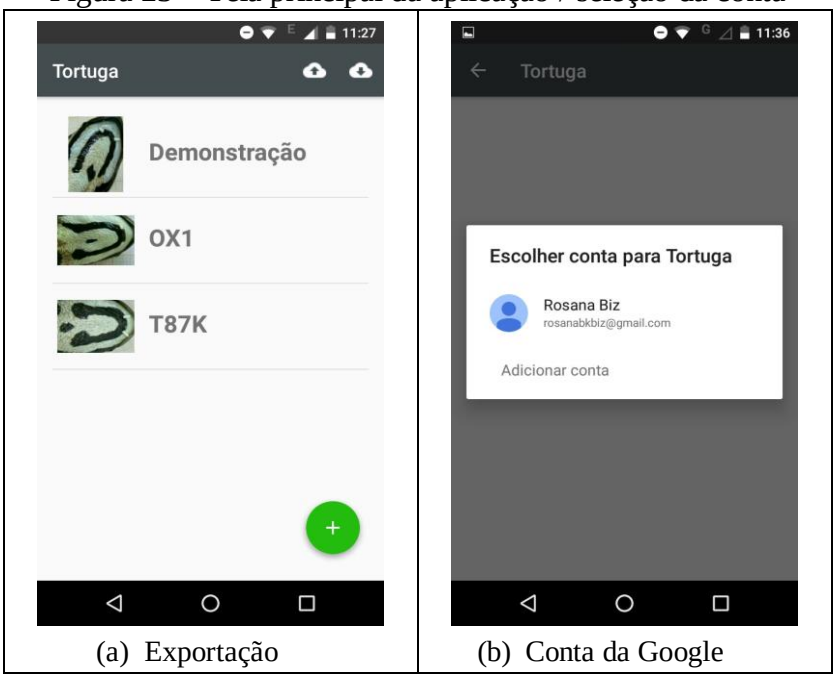
Na linha 5 são obtidos todos os acompanhamentos do cágado, pois é no acompanhamento que é salvo o local que o cágado já foi encontrado. Para cada acompanhamento é obtido a sua localização e então na linha 12 é criado um `LatLng`. Esse objeto é utilizado na linha 17 para informar a posição de um `MarkerOptions`. Essa classe armazena as configurações de um marcador do `GoogleMap`. Na linha 18 é informado o título do marcador e na linha 19 é informado o ícone do marcador. Após informar as configurações

do marcador ele é adicionado ao `GoogleMap` (linha 21). Após todos marcadores serem adicionados no mapa, na linha 23 o foco do `GoogleMap` e movido para a posição do último marcador criado.

3.4.7 Importação e exportação dos dados

O aplicativo permite ao usuário exportar os dados da aplicação para uma plataforma na nuvem e, caso for necessário, posteriormente realizar a importação desses dados. A plataforma utilizada para armazenar os dados da aplicação é o Google Drive. Essa plataforma foi escolhida pela praticidade do usuário, tendo em vista que ao utilizar um dispositivo Android o usuário já possui uma conta nessa plataforma. Para realizar a importação e exportação dos dados para o Google Drive foi utilizada a sua API disponibilizada pelo Google na versão 10.2.1. Na tela inicial da aplicação no canto superior direito existem dois botões para realizar a exportação e importação dos dados da aplicação, conforme apresentado na Figura 23a.

Figura 23 – Tela principal da aplicação / seleção da conta



(a) Exportação (b) Conta da Google

Fonte: elaborado pelo autor.

Ao clicar no botão para realizar a exportação dos dados o método `onOptionsItemSelected` abre a tela `ExportActivity`. A classe `ExportActivity` inicia o processo de exportação criando o `GoogleApiFragment`. Essa classe é responsável pelo gerenciamento da conexão com o API do Google Drive. Na criação da classe `GoogleApiFragment` é chamado o método `buildGoogleApiClient`. Esse método é responsável por criar a conexão com a conta do Google Drive. Nesse momento é apresentada

a tela para seleção da conta do Google Drive, que permite ao usuário selecionar a conta que deve receber os arquivos da exportação. A tela para seleção de conta pode ser vista na Figura 23b.

Quando a conta for selecionada é disparado um evento informando que a conexão foi estabelecida. A partir disso, a classe `ExportActivity` escuta esse evento através do método `onGoogleApiClientConnected` (Quadro 12). Esse método abre a tela que permite ao usuário selecionar o diretório onde serão salvos os arquivos da exportação.

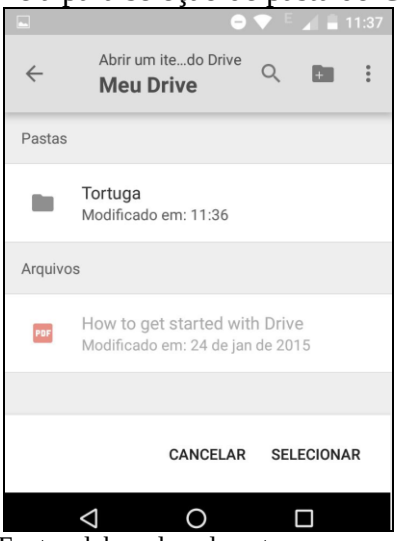
Quadro 12 – Código do método `onGoogleApiClienteConnected` da classe `ExportActivity`

```
1  @Subscribe
2  public void onGoogleApiClientConnected(GoogleApiConnection connection) {
3      if (isDirectoryRequested || !connection.contains(UNIQUE_GOOGLE_API_ID)) {
4          return;
5      }
6
7      googleApiClient = connection.get(UNIQUE_GOOGLE_API_ID);
8      final IntentSender intentSender = Drive.DriveApi
9              .newOpenFileActivityBuilder()
10             .setMimeType(new String[]{"application/vnd.google-apps.folder"})
11             .build(googleApiClient);
12
13     try {
14         startIntentSenderForResult(intentSender, REQUEST_DRIVE_DIRECTORY, null, 0, 0, 0);
15         isDirectoryRequested = true;
16     } catch (IntentSender.SendIntentException e) {
17         throw new RuntimeException("Export: Unable to show Google Drive.", e);
18     }
19 }
```

Fonte: elaborado pelo autor.

Nas linhas 8, 9, 10 e 11, é criado um `intentSender` através da API do Google Drive. Na linha 9 é aberta a tela para seleção de arquivos do Google Drive. Na linha 10 é informado que apenas arquivos do tipo pasta devem estar habilitados para seleção. Na linha 14 é iniciado o processo para seleção de arquivo passando o `intentSender` criado na linha 8. A Figura 24 apresenta a tela do Google Drive que é exibida ao usuário após a execução do código do Quadro 12.

Figura 24 – Tela para seleção de pasta do Google Drive



Fonte: elaborado pelo autor.

Ao selecionar a pasta é chamando o método `onActivityResult` da classe `ExportActivity`. Esse método é responsável por obter o `DriveId` da pasta que o usuário selecionou. Após obter o `DriveId` a exportação dos dados é iniciada através da execução da classe `DriveDataExporterRunnable`. Essa exportação é executada em uma *thread* secundaria. O Quadro 13 apresenta o código executado pela classe `DriveDataExporterRunnable`.

Quadro 13 – Código do método `run` da classe `DriveDataExporterRunnable`

```
1     @Override
2     public void run() {
3         DriveFolder userFolder = Drive.DriveApi.getFolder(googleApiClient, driveId);
4
5         DriveFolder backupFolder = createFolder(userFolder, fileTitle);
6
7         exportJson(backupFolder);
8         exportImages(backupFolder);
9     }
```

Fonte: elaborado pelo autor.

Na linha 3 é obtida a pasta que o usuário selecionou, essa pasta é utilizada na linha 5 onde é criada uma nova pasta com o nome da aplicação seguido pela data e hora em que a exportação foi realizada. Essa pasta irá conter um arquivo `json` com os dados que estão salvos no banco de dados e um arquivo `zip` com as imagens obtidas peso usuário. Na linha 7 é chamando o método `exportJson`, demonstrado no Quadro 14.

Quadro 14 – Código do método `exportJson` da classe `DriveDataExporterRunnable`

```
1     private void exportJson(DriveFolder folder) {
2         DriveApi.DriveContentsResult result = Drive.DriveApi
3                 .newDriveContents(googleApiClient).await();
4
5         if (!result.getStatus().isSuccess()) {
6             throw new RuntimeException("Data export has failed.");
7         }
8
9         DriveContents contents = result.getDriveContents();
10        OutputStream outputStream = contents.getOutputStream();
11        BackupDataExporter exporter = new BackupDataExporter(outputStream, ctx, true);
12
13        try {
14            exporter.exportData();
15            eventBus.post(exporter);
16        } catch (Exception e) {
17            String msg = "Data export has failed. " + e.getMessage();
18            RuntimeException error = new RuntimeException(msg, e);
19            eventBus.post(error);
20        }
21
22        createFile(folder, fileTitle + ".json", "application/json", contents);
23
24    }
```

Fonte: elaborado pelo autor.

Entre as linhas 2 e 9 é obtido um objeto do tipo `DriveContents`. Esse objeto fornece um `OutputStream` onde será escrito o conteúdo do arquivo que irá conter os dados do banco de dados. Na linha 14 é chamado o método `exportData` da classe `BackupDataExporter`. Esse método é responsável por obter os dados do banco de dados da aplicação e exportar eles para um arquivo `json`. No final do método `exportJson`, na linha 22 é chamando o método `createFile`. Ele é responsável por criar o arquivo `json` na pasta de exportação do Google Drive, salvando os dados do banco de dados na nuvem. No Quadro 15 é apresentado o código que obtém os dados da base de dados e exporta para o formato `json`.

Quadro 15 – Trecho de código do método `exportData` da classe `BackupDataExporter` responsável por exportar os dados da base

```
1     @Override
2     public void exportData(OutputStream outputStream) throws Exception {
3         if (json) {
4             GsonBuilder builder = new GsonBuilder();
5             Gson gson = builder.create();
6
7             TortoiseRepository repository = TortoiseRepository.getInstance(ctx);
8             List<Tortoise> tortoises = repository.findAll(true);
9
10            JsonObject root = new JsonObject();
11            JsonElement json = gson.toJsonTree(tortoises);
12            root.add("tortoises", json);
13
14            outputStream.write(root.toString().getBytes(CHARSET));
15            outputStream.flush();
16            outputStream.close();
17        } else {
18        //Código
19        }
20    }
```

Fonte: elaborado pelo autor.

Nas linhas 4 e 5 é obtida uma instância da classe `Gson`. Essa classe é da biblioteca Gson disponibilizada pela Google e visa facilitar a conversão de objetos Java em uma

representação `json`. Na linha 7 é obtida a instância do `TortoiseRepository`, ela é responsável por obter os cágados do banco de dados. Na linha seguinte é chamando o método `findAll` da classe `TortoiseRepository`, na chamada desse método é passado o parâmetro `true`, isso indica que as classes do tipo `Tortoise` retornadas pelo método já devem ter a sua lista interna de `Annotation` inicializada. Nas linhas 10, 11 e 12 é realizada a conversão dos objetos Java para sua representação em `json`. Feito isso, o código da linha 14 escreve os *bytes* desse objeto `json` no `OutputStream` recebido por parâmetro. Além da exportação dos dados da base, a classe `BackupDataExporter` também é responsável por realizar a exportação das imagens da aplicação. O Quadro 16 apresenta o código responsável por realizar essa atividade.

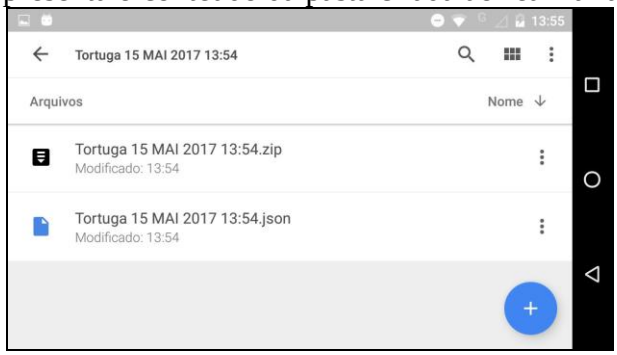
Quadro 16 – Trecho de código do método `exportData` da classe `BackupDataExporter` responsável por exportar as imagens da aplicação

```
1     @Override
2     public void exportData(OutputStream outputStream) throws Exception {
3         if (json) {
4         //Código
5         } else {
6             ImageHandler imageHandler = ImageHandler.getInstance(ctx);
7             List<String> images = imageHandler.getImagePaths("profile");
8
9             AnnotationRepository repository = AnnotationRepository.getInstance(ctx);
10            List<Annotation> annotations = repository.findAll();
11            for (Annotation annotation : annotations) {
12                String folder = String.valueOf(annotation.getId());
13                List<String> paths = imageHandler.getImagePaths(folder);
14                images.addAll(paths);
15            }
16
17            ZipManager.zip(ctx, images, outputStream);
18        }
19    }
```

Fonte: elaborado pelo autor.

Na linha 7 é obtida uma lista com o caminho de todas as imagens do perfil dos cágados. Da linha 9 até 15 são obtidos os caminhos das imagens da galeria de cada acompanhamento, após se obter o caminho de todas imagens da aplicação, então na linha 17 é chamado o método `zip` da classe `ZipManager`. Esse método recebe o caminho das imagens e compacta elas para um arquivo `zip` e escreve os *bytes* desse arquivo no `OutputStream` recebido por parâmetro. O resultado da exportação dos dados pode ser visto na Figura 25. Onde é possível ver a criação de uma pasta no Google Drive, com o nome da aplicação seguido da data e hora em que a exportação foi realizada. Dentro desta pasta existe um arquivo com a extensão `zip` com as imagens da aplicação e um arquivo com a extensão `json`, que possui os dados da base de dados.

Figura 25 – Tela que apresenta o conteúdo da pasta criada ao realizar a exportação dos dados



Fonte: elaborado pelo autor.

Ao clicar no botão para realizar a importação dos dados o método `onOptionsItemSelected` abre a tela `ImportActivity`. A classe `ImportActivity` inicia o processo de exportação criando o `GoogleApiFragment`. Essa classe é responsável pelo gerenciamento da conexão com o API do Google Drive. Na criação da classe `GoogleApiFragment` é chamado o método `buildGoogleApiClient`. Esse método é responsável por criar a conexão com a conta do Google Drive. Nesse momento é apresentada a tela para seleção da conta do Google Drive. Após o usuário selecionar a sua conta do Google Drive é realizada a conexão e então aberta a tela para selecionar a pasta que contém os arquivos que ele deseja importar. O processo para criação da conexão com o Google Drive e seleção de pasta é o mesmo que acontece com a `ExportActivity`, descrito no início da seção.

Ao selecionar a pasta é chamado o método `onActivityResult` da classe `ImportActivity`. Esse método é responsável por obter o `DriveId` da pasta que o usuário selecionou, posteriormente é iniciada e importação dos dados. A importação ocorre através da execução da classe `DriveDataImporterRunnable`, esse processamento é executado em uma *thread* secundaria, evitando o travamento da aplicação. O Quadro 17 apresenta o código executado pela classe `DriveDataImporterRunnable`.

Quadro 17 – Código do método `run` da classe `DriveDataImporterRunnable`

```
1     @Override
2     public void run() {
3         DriveFolder folder = Drive.DriveApi.getFolder(googleApiClient, driveId);
4
5         PendingResult<DriveApi.MetadataBufferResult> pendingResult =
6         folder.listChildren(googleApiClient);
7         DriveApi.MetadataBufferResult result = pendingResult.await();
8         MetadataBuffer metadataBuffer = result.getMetadataBuffer();
9         if (isValidFolder(metadataBuffer)) {
10            try {
11                boolean isJson = isJson(metadataBuffer.get(0));
12                Metadata jsonMetadata = metadataBuffer.get(isJson ? 0 : 1);
13                Metadata zipMetadata = metadataBuffer.get(isJson ? 1 : 0);
14
15                importJson(jsonMetadata);
16                importZip(zipMetadata);
17            } catch (Exception e) {
18                if (e.getMessage().contains("Import")) {
19                    this.eventBus.post(e);
20                } else {
21                    this.eventBus.post(new RuntimeException("Import: Ocorreu um erro
22 durante a importação dos dados. Tente novamente.", e));
23                }
24            }
25        } else {
26            this.eventBus.post(new RuntimeException("Import: A pasta escolhida é
27 inválida para importação dos dados"));
28        }
29    }
```

Fonte: elaborado pelo autor.

Na linha 6 são obtidos todos arquivos da pasta selecionada pelo usuário. Na linha 9 é chamando o método `isValidFolder`, que verifica o conteúdo da pasta selecionada, para a pasta ser considerada válida devem existir dois arquivos, um arquivo `json` e outro `zip`. Nas linhas 15 e 16 são executados os métodos responsáveis pela importação dos dados dos arquivos.

O Quadro 18 apresenta o código do método `importJson`. Através do `Metadata` recebido por parâmetro é obtido o conteúdo do arquivo `json` (linha 2). Na linha 4 então é criada uma instância da classe `BackupDataImporter` que recebe o conteúdo desse arquivo no seu construtor. Na linha 6 então é executado o método `importData` da classe `BackupDataImporter`, nele é realizada a importação dos dados do arquivo `json` para a base de dados do aplicativo.

Quadro 18 – Código do método `importJson` da classe `DriveDataImporterRunnable`

```
1     private void importJson(Metadata jsonMetadata) {
2         InputStream inputStream  = getInputStream(jsonMetadata);
3
4         BackupDataImporter importer = new BackupDataImporter(inputStream, ctx, true);
5         try {
6             importer.importData();
7             eventBus.post(importer);
8         } catch (Exception e) {
9             String msg = "Import: A importação dos dados falhou. " + e.getMessage();
10            RuntimeException error = new RuntimeException(msg, e);
11            eventBus.post(error);
12        }
13    }
```

Fonte: elaborado pelo autor.

No Quadro 19 é possível ver o pedaço de código do método `importData` que é responsável por importar os dados do arquivo `json`. Na linha 4 é obtido o `JsonObject` do conteúdo do `InputStream` recebido por parâmetro. Na linha 6 é obtida a lista de cágados do `JsonObject`, onde cada elemento dessa lista é convertido para um objeto da classe `Tortoise` (linha 9). Posteriormente, esse cágado é salvo na base de dados (linha 12). Na linha 14 são obtidas as anotações do cágado. Nas linhas seguintes é iterado em cada anotação da lista onde a mesma é persistida na base de dados (linha 19).

Quadro 19 – Código do método `importData` da classe `BackupDataImporter`

```
1     @Override
2     protected void importData(InputStream inputStream) throws Exception {
3         if (this.json) {
4             JsonObject root = inputStreamToJson(inputStream);
5             if (root != null) {
6                 JsonArray jsonArray = root.get("tortoises").getAsJsonArray();
7                 for (int i = 0; i < jsonArray.size(); i++) {
8                 String json = jsonArray.get(i).getAsJsonObject();
9                     Tortoise tortoise = gson.fromJson(json, Tortoise.class);
10                    tortoise.setId(-1);
11
12                    tortoiseRepository.save(tortoise);
13
14                    List<Annotation> annotations = tortoise.getAnnotations();
15                    for (Annotation annotation : annotations) {
16                        annotation.setId(-1);
17                        annotation.setTortoiseId(tortoise.getId());
18
19                        annotationRepository.save(annotation);
20                    }
21                }
22            }
23        } else {
24        //Código
25        }
26    }
```

Fonte: elaborado pelo autor.

Além da importação dos dados do arquivo `json`, o método `importData` também realiza a importação das imagens salvas no arquivo `zip`. O Quadro 20 apresenta o trecho de código responsável pela importação desse arquivo.

Quadro 20 – Código do método `importData` da classe `BackupDataImporter`

```
1  @Override
2  protected void importData(InputStream inputStream) throws Exception {
3      if (this.json) {
4      //Código
5      } else {
6          ZipInputStream zin = new ZipInputStream(inputStream);
7          ZipEntry ze;
8          while ((ze = zin.getNextEntry()) != null) {
9              String absolutePath = ze.getName();
10             String path = absolutePath.substring(0, absolutePath.lastIndexOf("/") + 1);
11             String name = absolutePath.substring(absolutePath.lastIndexOf("/") + 1,
12                                                  absolutePath.length());
13
14             File imgFile = ImageHandler.getInstance(ctx).createImageFile(path, name);
15
16             FileOutputStream outputStream = new FileOutputStream(imgFile);
17             IOUtils.copy(zin, outputStream);
18             outputStream.close();
19             zin.closeEntry();
20         }
21         zin.close();
22     }
23 }
```

Fonte: elaborado pelo autor.

Na linha 6 o `InputStream` recebido por parâmetro é encapsulado em um `ZipInputStream`. Nas linhas seguintes é realizada a iteração em cada entrada do arquivo `zip`. Para cada entrada encontrada no arquivo é obtido o caminho absoluto do arquivo (linha 9). Na linha 14 é criado um arquivo no mesmo caminho e com o mesmo nome do arquivo anterior. Na linha 17 são copiados os *bytes* da entrada do arquivo `zip` para o novo arquivo criado na linha 14. Esse procedimento é realizado para cada entrada do arquivo `zip` até que todas as imagens são importadas para a aplicação.

3.4.8 Aplicação web

A aplicação Android, ao possuir conexão com a internet, envia os dados dos cágados cadastrados para uma aplicação web. A aplicação web armazena os dados de todos os cágados reconhecidos pelos dispositivos móveis. Essa aplicação foi desenvolvida visando permitir o compartilhamento das informações de monitoramento entre os biólogos. Nela é possível ver quais áreas já estão sendo estudadas por outros biólogos, assim como, também é possível observar o comportamento da movimentação desses cágados. As próximas seções apresentam o detalhamento da implementação dessa aplicação.

3.4.8.1 Integração dos dados

A integração dos dados é feita pela aplicação Android. Sempre que a aplicação é aberta é verificado no método `onCreate` da classe `MainActivity` se os cágados já foram enviados para a aplicação web e se o dispositivo móvel possui conexão com a internet. Caso os cágados ainda não tenham sido enviados e há conexão com a internet, os dados são enviados para a

aplicação web. O Quadro 21 apresenta o trecho de código do método `onCreate` responsável pela integração dos dados.

Quadro 21 –Código do método `onCreate` da classe `MainActivity`

```
1     @Override
2     protected void onCreate(Bundle savedInstanceState) {
3         super.onCreate(savedInstanceState);
4         setContentView(R.layout.activity_main);
5
6         tortoiseRepository = TortoiseRepository.getInstance(this);
7
8         requestPermissions(new String[]{
9                         Manifest.permission.ACCESS_FINE_LOCATION,
10                        Manifest.permission.ACCESS_COARSE_LOCATION},
11                0);
12
13        if (sendTortoises && InternetUtils.hasConnection(this)) {
14            TortugaIntegration integration =
15 TortugaIntegration.getInstance(MainActivity.this);
16
17            List<Tortoise> tortoises = tortoiseRepository.findAll(true);
18            for (Tortoise tortoise : tortoises) {
19                integration.sendTortoise(tortoise);
20            }
21            sendTortoises = false;
22        }
23
24        //Código
25    }
```

Fonte: elaborado pelo autor.

Na linha 13 é verificado se é necessário realizar a integração dos dados na aplicação web. Na linha 17 é executado o método `findAll` da classe `TortoiseRepository` passando o parâmetro `true`. Esse método retorna todos os cágados e seus acompanhamentos da base de dados. Nas próximas linhas é iterado sobre cada cágado retornado, sendo executado o método `sendTortoise` da classe `TortugaIntegration`. Esse é o método responsável por realizar a integração dos dados para a aplicação web. O Quadro 22 apresenta o código do método `sendTortoise`.

Quadro 22 – Código do método `sendTortoise` da classe `TortugaIntegration`

```
1     public void sendTortoise(Tortoise tortoise) {
2         HashMap<String, Object> json = new HashMap<>();
3         json.put("_id", tortoise.getUuidFromSeries());
4         json.put("name", tortoise.getName());
5         json.put("gender", tortoise.getGender());
6
7         //Imagem perfil
8         File image = imageHandler.getImageFile("profile", tortoise.getId());
9         if (image != null) {
10            String encodedImage = toBase64Str(image);
11            json.put("image", "" + encodedImage);
12        }
13
14        if (!tortoise.getAnnotations().isEmpty()) {
15            List<Annotation> annotations = tortoise.getAnnotations();
16
17            //Endereço
18            Annotation last = annotations.get(annotations.size() - 1);
19            json.put("address", last.getAddress());
20
21            //Posições no mapa
22            List<HashMap<String, Object>> markers = new ArrayList<>();
23            for (Annotation annotation : annotations) {
24                HashMap<String, Object> marker = new HashMap<>();
25                marker.put("title", annotation.getCreatedAtStr());
26                marker.put("pos", annotation.getLatLng());
27
28                markers.add(marker);
29            }
30            json.put("markers", markers);
31        }
32        sendTortoise(json);
33    }
```

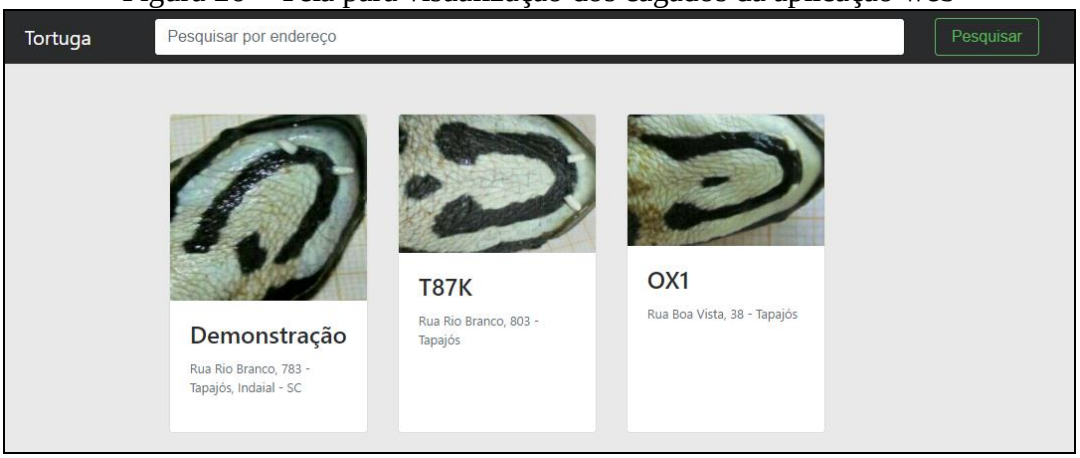
Fonte: elaborado pelo autor.

Na linha 2 é criado um `HashMap` que vai armazenar os campos que devem ser enviados para a aplicação web. Na linha 3 é criado o campo `_id`. Esse campo recebe um UUID baseado nos *bytes* dos descritores do cágado que foram gerados pelo método desenvolvido por Bertoldi (2016). Posteriormente são criados os campos `name` (linha 4) e `gender` (linha 5), ambos informados pelo usuário no cadastro do cágado. Na linha 8 é obtido o `File` que possui os *bytes* da imagem de perfil do cágado. Esses *bytes* são convertidos para uma `String` base64 pelo método `toBase64Str` (linha 10) sendo adicionada no campo `image`. Na linha 19 é adicionado o campo `address` que possui o endereço do cágado. Esse endereço é obtido a partir do último acompanhamento (linha 18) que foi cadastrado para o cágado. Em seguida é realizada a iteração para cada acompanhamento do cágado a fim de criar um marcador que deve ser apresentado no Google Maps. Esse marcador vai possuir o campo `title` com a data e hora em que foi realizado o acompanhamento (linha 25) e o campo `pos` com a localização do cágado (linha 26). Na linha 30 são adicionados os marcadores no `HashMap` que é enviado para a aplicação web na linha 32.

3.4.8.2 Telas da aplicação web

A aplicação web possui duas telas, a principal que apresenta todos os cágados reconhecidos pelas aplicações Android (Figura 26) e a tela que apresenta o perfil do cágado com todos os locais em que o cágado foi encontrado (Figura 27).

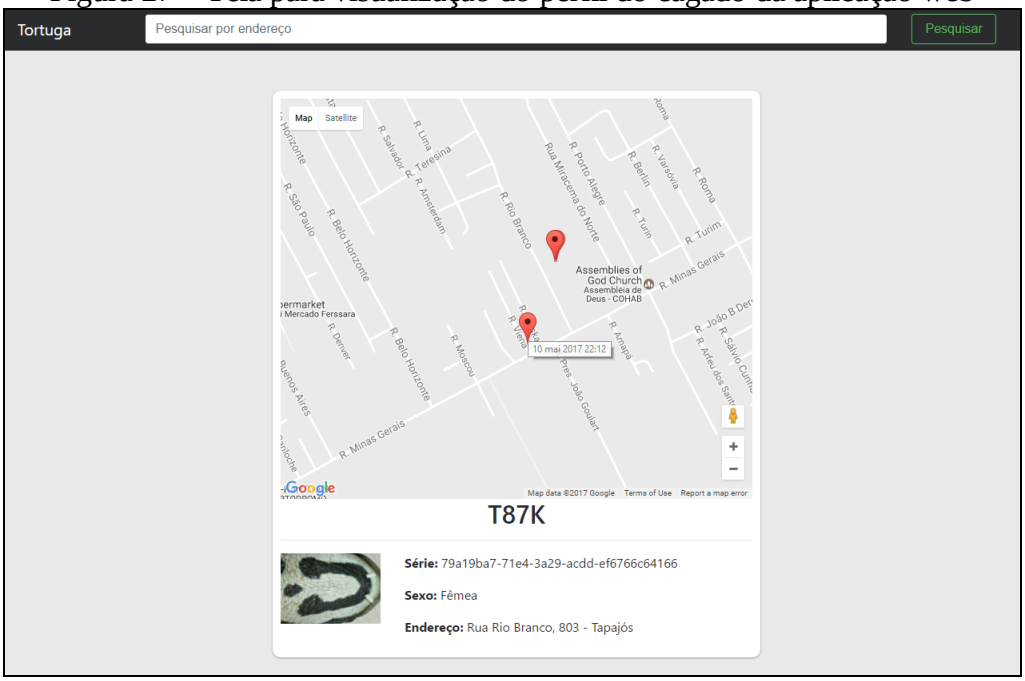
Figura 26 – Tela para visualização dos cágados da aplicação web



Fonte: elaborado pelo autor.

Na tela principal os cágados são apresentados em cartões, onde é possível ver a imagem do perfil, o seu identificador informado pelo usuário e o endereço do seu último acompanhamento. É possível filtrar os cágados apresentados pelo seu endereço. Para isso, basta informar uma parte do endereço no campo encontrado na parte superior da tela e clicar no botão pesquisar. É possível visualizar os detalhes do cágado clicando no cartão. A tela de detalhes do cágado pode ser vista na Figura 27.

Figura 27 – Tela para visualização do perfil do cágado da aplicação web



Fonte: elaborado pelo autor.

Nessa tela são apresentados marcadores no Google Maps que indicam os locais em que o cágado já foi encontrado. Passando o mouse sobre esses marcadores é possível visualizar a data e hora em que o cágado foi encontrado. A partir dela também é possível ver o identificador informado pelo usuário logo abaixo da visualização do mapa. Abaixo do identificador é apresentada a imagem de perfil do cágado, o seu UUID gerado pelos descritores dessa imagem, o sexo do cágado e o endereço do último acompanhamento.

3.5 ANÁLISE DOS RESULTADOS

Nesta seção são apresentados os testes realizados com o protótipo desenvolvido. A seção 3.5.1 apresenta o teste de usabilidade, onde é detalhada a metodologia utilizada, como foi a execução dos testes e a análise dos resultados dos testes efetuados. Na seção 3.5.2 é demonstrado o teste realizado na funcionalidade de *backup* dos dados, e a compatibilidade da aplicação com diferentes hardwares e versões do Android.

3.5.1 Teste de usabilidade

O teste de usabilidade foi feito com onze usuários, com perfis variados, a fim de avaliar a aceitação do protótipo para diferentes públicos.

Para realizar o teste foi solicitado que cada usuário instalasse o `apk` da aplicação em seu próprio smartphone, a fim de verificar a execução da aplicação em diferentes dispositivos e versões do Android. Junto ao `apk` também foi disponibilizado o questionário de perfil, uma lista de tarefas e um questionário de usabilidade, que estão disponíveis no Apêndice A.

Inicialmente, os participantes foram orientados sobre o objetivo do aplicativo e dos testes de usabilidade. Em seguida, cada voluntário preencheu o questionário de perfil de usuário. Feito isso, foi solicitado que o usuário realizasse a instalação do `apk` em seu dispositivo e realizasse a lista de tarefas. A lista de tarefas possui quatorze atividades, onde treze deveriam ser executadas no aplicativo Android e uma na aplicação web. Ao final de cada atividade, o usuário foi orientado a informar se conseguiu realizar a atividade com êxito e se necessário realizar uma observação sobre a atividade.

Para finalizar o teste, cada voluntário respondeu ao questionário de usabilidade do protótipo. O questionário foi composto de quinze perguntas fechadas e uma pergunta aberta. Onde foi possível obter as impressões do usuário sobre a usabilidade do aplicativo, a efetividade das funcionalidades desenvolvidas e as vantagens que o protótipo traz ao monitoramento e acompanhamento de quelônios. Ao final do questionário foi aberto ao

usuário para deixar suas críticas e sugestões sobre o protótipo desenvolvido. Os resultados desse experimento são apresentados nas próximas seções.

3.5.1.1 Análise e interpretação dos dados do perfil de usuário

A primeira análise foi feita em cima dos dados coletados através do questionário de perfil de usuário. O Quadro 23 apresenta o perfil dos voluntários que participaram do teste de usabilidade.

Quadro 23 – Perfil dos usuários que realizaram o teste de usabilidade

Sexo	82% masculino 18% feminino
Idade	73% entre 18 e 25 anos 18% entre 25 e 35 anos 9% mais de 35 anos
Profissão	46% programador 18% estudante 9% analista de custos 9% analista de sistemas 9% biólogo 9% desenhista projetista
Uso de celular	100% todos os dias
Atua em alguma área da biologia	27% sim 73% não
Já realizou o monitoramento e acompanhamento de alguma espécie	9% sim 91% não

Fonte: elaborado pelo autor.

A partir dos dados apresentados é possível perceber que o protótipo foi avaliado por usuários de perfis variados. Tornando possível a validação das funcionalidades e usabilidade do protótipo com diferentes públicos de usuários. É possível verificar que mesmo com a variação de perfil dos candidatos, todos utilizam o celular diariamente. Somente 27% dos voluntários atuam em alguma área da biologia e, apenas 9% já realizou o monitoramento e acompanhamento de alguma espécie.

3.5.1.2 Análise dos resultados da lista de tarefas

Após a análise do perfil dos voluntários, foi realizada a avaliação dos dados obtidos a partir da lista de tarefas executada pelos usuários. Todos voluntários conseguiram realizar o cadastro de um cágado no aplicativo, porém alguns usuários sinalizaram que a opção para realizar o cadastro do cágado na tela inicial da aplicação não está muito clara. Após o cadastro, foi solicitado que o usuário acessasse o perfil do cágado cadastrado e realizassem alguma alteração nas informações cadastradas, nessas atividades não foram relatadas dificuldades pelos usuários.

Após o cadastro e edição do perfil do cágado, foi executada a atividade para adicionar um acompanhamento ao cágado cadastrado. Nas duas primeiras listas de tarefas, era solicitado ao usuário que ele informasse a imagem das listras do cágado para que a aplicação reconhecesse o cágado e então permitisse ao usuário adicionar um novo acompanhamento. Porém, os usuários relataram dificuldades em realizar o cadastro de acompanhamento dessa maneira. Para tornar o cadastro de acompanhamento mais simples, foi alterada a aplicação para permitir o cadastro de acompanhamento pelo perfil do cágado, sem a necessidade das etapas de seleção de imagem e identificação do cágado. Ao realizar as alterações não foram mais apresentadas dificuldades ao adicionar um novo acompanhamento ao cágado.

Os acompanhamentos cadastrados na aplicação buscam auxiliar os biólogos na atividade de monitoramento e acompanhamento dos cágados, a Tabela 1 apresenta os resultados das atividades que estão relacionadas a esse objetivo.

Tabela 1– Respostas das atividades de acompanhamento

Tarefas/Respostas	**Sim**	**Sim, mas com dificuldade**	**Não**
Cadastro de acompanhamento	82%	18%	-
Edição do acompanhamento	91%	-	9%
Selecionar a localização do acompanhamento no Google Maps	91%	-	9%
Vincular imagens ao acompanhamento	82%	9%	9%
Visualizar os locais dos acompanhamentos no Google Maps	91%	-	9%

Fonte: elaborado pelo autor.

A partir da Tabela 1, percebe-se que todos os usuários conseguiram cadastrar o acompanhamento, porém, como explicado anteriormente, antes da alteração da aplicação alguns usuários acharam confusa a forma para cadastrar o acompanhamento. Nas atividades de edição do acompanhamento e seleção de localização 9% dos usuários relataram que a aplicação travou e não foi possível realizar as atividades.

Quanto a vinculação de imagens ao acompanhamento 82% dos usuários realizaram a tarefa sem dificuldade. No entanto 9% dos usuários indicaram que conseguiram realizar a tarefa, mas com dificuldade, quando o usuário selecionou duas imagens a aplicação apresentou apenas uma delas. Foi identificado que isso ocorreu por um erro da aplicação, onde ao abrir a galeria de imagens era possível selecionar mais de uma imagem, o problema foi ajustado e outros usuários não relataram outros erros. Além disso, 9% dos usuários sinalizaram que não conseguiram identificar a opção para vinculação de imagens ao acompanhamento.

Outra tarefa importante para o monitoramento e acompanhamento de cágados é visualizar os locais em que foram realizados os acompanhamentos do cágado. Essa atividade foi realizada sem apresentar problemas por 91% dos usuários. Apenas 9% não conseguiram

realizar essa atividade, identificou-se que o problema foi causado pela instabilidade da internet móvel dos usuários. Como essa funcionalidade utiliza o Google Maps para apresentar os locais no mapa é indispensável à conexão com a internet.

Por fim, na última atividade era solicitado ao usuário para desativar a conexão com a internet do dispositivo, realizar o cadastrado de um novo cágado e então fechar a aplicação. Feito isso, ele deveria ativar a conexão com a internet e abrir a aplicação Android novamente. Essa atividade teve o objetivo de validar a sincronização dos dados da aplicação Android com a Aplicação web. O envio dos dados não impacta a usabilidade do usuário na aplicação Android, pois ele é executado em uma *thread* em segundo plano. A atividade foi realizada com sucesso por 82% dos usuários, onde o cágado cadastrado no aplicativo era apresentado na aplicação web. Foi feita uma observação indicando que seria interessante permitir ao biólogo cadastrar as suas informações para que elas também fossem enviadas para o site. Permitindo assim que os biólogos não identifiquem apenas os cágados e locais em que estão sendo realizados monitoramentos, mas também os próprios biólogos, facilitando assim a interação entre eles. Foi identificado que 18% dos usuários não tiveram o cágado cadastrado apresentado na aplicação web. Esse problema ocorreu apenas com usuários que apresentaram instabilidade da conexão de internet móvel do dispositivo.

3.5.1.3 Análise de usabilidade e funcionalidades

O questionário de avaliação do protótipo aplicado com os voluntários possui oito perguntas relacionadas à usabilidade do aplicativo e sete perguntas que avaliam as funcionalidades entregues pelo protótipo. A Tabela 2 apresenta as respostas das perguntas que avaliam a usabilidade do aplicativo.

Tabela 2 – Respostas das perguntas referentes a usabilidade do protótipo

Perguntas / Critérios de avaliação	Concordo totalmente	Concordo parcialmente	Não concordo nem discordo	Discordo parcialmente	Discordo totalmente
1. O design da interface do aplicativo é atraente	9%	64%	9%	18%	-
2. É fácil navegar pelo aplicativo	64%	27%	-	9%	-
3. Os símbolos e ícones são claros e intuitivos	64%	27%	9%	-	-
4. A interface é semelhante dos demais aplicativos	46%	36%	18%	-	-
5. É fácil realizar os cadastros de cágados e acompanhamentos	64%	27%	9%	-	-
6. As informações armazenadas no aplicativo são facilmente encontradas	82%	9%	9%	-	-
7. Às vezes não sei o que fazer no aplicativo	9%	18%	18%	-	55%
8. Você precisaria do apoio de uma pessoa para usar este aplicativo	27%	9%	9%	-	55%

Fonte: elaborado pelo autor.

Pelos dados apresentados no Tabela 2, é possível perceber que a maior parte dos usuários achou o design da interface do aplicativo atraente e não tiveram problemas para navegar pelo aplicativo. Porém alguns usuários destacaram que seria interessante ter um tutorial de como realizar o primeiro cadastro do cágado.

Praticamente nenhum dos usuários teve dúvidas quanto aos símbolos e ícones apresentados na aplicação. A maioria também concordou que a interface do aplicativo e semelhante aos demais aplicativos. O cadastrado de cágados e acompanhamentos foi considerado fácil pela maioria dos usuários, alguns usuários apresentaram algumas dificuldades, mas em pouco tempo conseguiram efetuar os cadastros. A maioria dos usuários achou fácil encontrar as informações cadastradas no aplicativo, e nenhum usuário teve dificuldades quanto a esse assunto.

Alguns usuários relataram ter dificuldade em realizar alguma atividade no aplicativo, porém mais da metade afirmaram que não tiveram nenhuma dificuldade durante a utilização do aplicativo. Mais da metade dos usuários também afirmaram que não precisariam de outra pessoa para auxiliar no uso do aplicativo, alguns não concordaram nem discordaram, porém, alguns usuários afirmaram que precisariam de ajuda para utilizar a aplicação.

Na sequência são analisadas as respostas das perguntas que avaliaram as funcionalidades entregues pelo protótipo e a sua eficácia em relação ao monitoramento e acompanhamento de cágados. A Tabela 3 apresenta as respostas dessas perguntas.

Tabela 3 – Respostas referentes as funcionalidades do protótipo

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Não concordo nem discordo**	**Discordo parcialmente**	**Discordo totalmente**
1. Você acredita que o aplicativo auxilia no monitoramento e acompanhamento de cágados	64%	36%	-	-	-
2. A comparação entre os cágados cadastrados no aplicativo auxilia na identificação do individuo	73%	27%	-	-	-
3. Você achou importante a funcionalidade que permite realizar o backup dos dados do aplicativo para o Google Drive	73%	9%	9%	9%	-
4. Você acha que visualizar os locais de captura do cágado no Google Maps ajuda a entender a movimentação do mesmo	91%	9%	-	-	-
5. Você utilizaria esse aplicativo para auxiliar no monitoramento e acompanhamento de cágados	73%	27%	-	-	-
6. Você recomendaria esse aplicativo para outra pessoa	64%	27%	9%	-	-
7. Você acredita que a aplicação web ajuda no compartilhamento de informações entre os biólogos	64%	18%	9%	-	9%

Fonte: elaborado pelo autor.

Mais da metade dos usuários acreditam que o aplicativo auxilia no monitoramento e acompanhamento de cágados, os outros usuários concordam parcialmente com essa afirmação. Os usuários também concordam que a comparação realizada entre os cágados cadastrados auxiliou na identificação do indivíduo. A maioria dos usuários achou importante a funcionalidade que permite realizar o *backup* dos dados do aplicativo para o Google Drive. Alguns usuários não concordam nem discordam, e alguns discordaram parcialmente.

A maioria dos usuários achou que visualizar os locais de captura do cágado no Google Maps ajuda a entender a movimentação do cágado, alguns usuários concordaram parcialmente com a afirmativa. A maior parte dos usuários utilizaria o aplicativo para auxiliar no monitoramento e acompanhamento de cágados, outros usuários concordaram parcialmente com a afirmativa. Mais da metade dos usuários recomendariam o aplicativo para outra pessoa, 27% recomendariam parcialmente e 9% não souberam opinar. Mais da metade dos usuários acreditam que a aplicação web ajuda no compartilhamento de informações entre os biólogos, 18% concordaram parcialmente com a afirmativa, 9% não souberam opinar e 9% discordaram totalmente.

3.5.2 Teste de backup dos dados e compatibilidade

Para validar a funcionalidade de backup dos dados da aplicação foi realizada a exportação dos dados para o Google Drive e posteriormente a importação a partir de outro dispositivo. Também foi possível realizar o *download* dos arquivos no computador e validar o conteúdo dos mesmos. Não foram encontrados problemas nos testes executados. Porém foi observado que devido à instabilidade da conexão de internet móvel é recomendado que a exportação dos dados seja realizada quando o dispositivo estiver conectado a uma rede Wi-Fi.

O teste de compatibilidade foi realizado utilizando onze *smartphones* Android, possibilitando verificar a execução da aplicação em diferentes hardwares e versões do sistema operacional. O Quadro 24 apresenta as características dos smartphones utilizados nos testes.

Quadro 24 – Smartphones utilizados nos testes

Quantidade	**Dispositivo**	**Versão do Android**	**Tamanho da tela**	**Resolução**
1	Motorola Moto G	5.1.1	4.5	(1280x720)
3	Motorola Moto G 2nd gen	6.0	5	(1280x720)
1	Motorola Moto G 3rd gen	6.0	5	(1280x720)
1	Motorola Moto G4	7.0	5.5	(1920x1080)
1	Motorola Moto X 2nd gen	6.0	5.2	(1920x1080)
1	Samsung Galaxy S7	7.0	5.1	(2560x1440)
1	Asus Zenfone 2	6.0	5.5	(1920x1080)
1	Asus Zenfone 3 Max	6.0.1	5.2	(1280x720)
1	Samsung Galaxy J5	6.0.1	5	(1280x720)

Fonte: elaborado pelo autor.

O aplicativo foi executado sem problemas em todos os *smartphones* e todas funcionalidades foram executadas com êxito. As telas da aplicação se ajustaram de acordo com o tamanho e resolução de cada dispositivo. Apenas os títulos de algumas telas foram cortados, devido ao tamanho do dispositivo. Porém, isso não afetou a experiência do usuário. O dispositivo com a versão mais antiga do Android também não apresentou nenhuma diferença para os mais novos, mostrando que a aplicação possui uma boa compatibilidade com a maioria das versões de dispositivos Android.

4 CONCLUSÕES

O presente trabalho propôs o desenvolvimento de uma aplicação que auxiliasse biólogos no monitoramento de cágados da espécie Phrynops Williamsi. A aplicação deveria facilitar ao usuário encontrar as informações dos acompanhamentos realizados com os cágados já catalogados. Também deveriam ser apresentados no Google Maps os locais onde foram realizadas as capturas do cágado, permitindo ao usuário analisar de forma mais clara a movimentação do indivíduo.

Para o desenvolvimento da aplicação móvel foi utilizada a linguagem de programação Java no ambiente de desenvolvimento Android Studio. Para realizar o processamento da imagem do cágado, foi utilizada a biblioteca OpenCV4Android. No desenvolvimento da aplicação web foi utilizado a linguagem de programação JavaScript para o *back-end*, no *front-end* foi utilizado HTML, CSS e JavaScript. Foi utilizada a IDE Atom para o desenvolvimento da aplicação web.

Os resultados atingidos foram satisfatórios, a aplicação Android atingiu o objetivo de auxiliar ao usuário encontrar as informações dos cágados já catalogados, permitindo a identificação do cágado através de uma imagem das listras abaixo da cabeça do indivíduo. A possibilidade de manter o histórico dos acompanhamentos do cágado, permitindo ao usuário adicionar imagens através da câmera ou da própria galeria do dispositivo e a visualização dos locais em que o cágado já foi capturado, facilita a realização do monitoramento e acompanhamento dessa espécie.

A aplicação web permite que biólogos identifiquem quais áreas já estão sendo monitoradas e quais indivíduos já foram catalogados. A possibilidade de o usuário encontrar as informações dos cágados catalogados pelo aplicativo em um único lugar facilita o compartilhamento das informações de monitoramento entre os biólogos.

A funcionalidade de exportação e importação de backup dos dados fornece ao usuário a garantia de que os dados dos acompanhamentos estão seguros, permitindo a restauração dos dados caso ocorra algum problema com o seu dispositivo móvel. A funcionalidade de exportação também facilita ao biólogo utilizar os dados da aplicação em outros estudos, tendo em vista que os arquivos ficam disponíveis em sua conta do Google Drive.

Como limitação, destacam-se os problemas encontrados na realização da exportação dos dados para o Google Drive e na sincronização dos dados entre as aplicações quando o dispositivo móvel não possuía uma conexão de internet móvel estável. Outra limitação foi a baixa quantidade de biólogos que realizaram os testes de usabilidade da solução desenvolvida,

tendo em vista que biólogos que já realizaram as atividades de monitoramento/ acompanhamento com outras ferramentas teriam mais embasamento para validar a solução desenvolvida.

Ao final do desenvolvimento do projeto, acredita-se que as ferramentas desenvolvidas possam contribuir nas atividades de monitoramento e acompanhamento de cágados da espécie Phrynops Williamsi, visto os resultados apresentados pelo trabalho.

4.1 EXTENSÕES

Para extensões deste trabalho propõem-se:

a) melhorar o algoritmo de identificação, tornando-o mais eficiente em imagens de diferentes ângulos das listras;
b) incluir uma funcionalidade que verifique a qualidade da imagem das listras do cágado, avisando ao usuário quando é necessário selecionar outra imagem;
c) permitir a exportação dos dados em formato Excel para permitir aos biólogos realizarem analises estatísticas dos dados;
d) incluir um passo-a-passo para instruir como realizar o acompanhamento de um cágado;
e) incluir as informações do biólogo na aplicação web, visando facilitar o contato entre os pesquisadores da área;
f) permitir que a aplicação web também realize a identificação dos cágados através da imagem das listras;
g) criar mecanismo para tirar a foto das listras do cágado automaticamente quando a câmera do dispositivo detectar as listras do cágado;
h) permitir adicionar várias imagens ao perfil do cágado, facilitando a identificação do usuário.

REFERÊNCIAS

BALESTRA, Rafael A. M. et al. Roteiro para Inventários e Monitoramentos de Quelônios Continentais. **Biodiversidade Brasileira**, Brasília/DF, [s.l.], v.16, n. 1, p. 114-152, Ago. 2015. Disponível em: <http://www.icmbio.gov.br/revistaeletronica/index.php/BioBR/article/view/471/459>. Acesso em: 17 set. 2016.

BERTOLDI, Guilherme. O. **Tortuga**: Um protótipo para identificação de cágados da espécie Phrynops Williamsi. 2016. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: < http://dsc.inf.furb.br/arquivos/tccs/monografias/2016_1_guilherme-oecksler-bertoldi_monografia.pdf>. Acesso em: 07 set. 2013.

BOERO, Ferdinando. The Study of Species in the Era of Biodiversity: A Tale of Stupidity. **Diversity Open Access Biodiversity Journal**, Suiça, v. 2, n. 1, p. 115-126, Jan. 2010.

BORTOLON, Matheus. **Plantarum**: Uma aplicação Android para consultas de plantas. 2014. 83 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://dsc.inf.furb.br/arquivos/tccs/monografias/2014_1_matheus-bortolon_monografia.pdf>. Acesso em: 14 ago. 2016.

BURGHARDT, Tilo. **Visual Animal Biometrics**: Automatic detection and individual identification by coat pattern. 2008. 177 f. Dissertação (Doutorado em Filosofia) – Departamento de Ciência da Computação, Universidade de Bristol, Inglaterra. Disponível em: <http://www.bmva.org/thesis-archive/2008/2008-burghardt.pdf>. Acesso em: 17 set. 2016.

COSTA, Raul et al. **Monitoramento in situ da biodiversidade**: Uma proposta para a composição de um Sistema Brasileiro de Monitoramento da Biodiversidade. 2. ed. Brasília/DF: ICMBio, 2013.

CULLEN JR., Larry; VALLADARES-PADUA, Cláudio; RUDRAN, Rudy. **Métodos de estudos em biologia da conservação e manejo da vida silvestre**. 2. ed. Curitiba: Universidade Federal do Paraná, 2006.

EDWARDS, J. D. Novel Technology for the Remote Monitoring of Animals. **Companion Animal Society Newsletter**, Nova Zelândia, v. 23, n. 2, p. 56-59, Jun. 2012.

JVISUAL. **LikeThat Garden**. Estados Unidos, 2015. Disponível em: <https://www.likethatapps.com/LikeThatGarden/index.html>. Acesso em: 10 set. 2016.

LEAFSNAP. **Leafsnap**: An electronic field guide. Reino Unido, 2011. Disponível em: <http://leafsnap.com>. Acesso em: 02 set. 2016.

LISBOA FILHO, Jugurta; IOCHPE, Cirano. **Introdução a sistemas de informação geográficas com ênfase em banco de dados**. Viçosa, 1996.

LUSTOSA, Ana Paula G. et al. **Manejo conservacionista e monitoramento populacional de quelônios amazônicos**. 2. ed. Brasília/DF: Ibama, 2016.

PIC4TURTLE. **Pic4Turtle**. Brasil, 2016. Disponível em: <https://www.pic4turtle.com>. Acesso em: 02 set. 2016.

REISSER, Júlia et al. Photographic identification of sea turtles: method description and validation, with an estimation of tag loss. **Endangered Species Research**, Alemanha, v. 5, n. 2, p. 73-82, Set. 2008.

SILVA, Marcel S. **Sistemas de Informações Geográficas**: elementos para o desenvolvimento de bibliotecas digitais geográficas distribuídas. 2006. 167 f. Dissertação (Mestrado em Ciência da Informação) - Faculdade de Filosofia e Ciências, Universidade Estadual Paulista, Campus de Marília. Disponível em: <https://www.marilia.unesp.br/Home/Pos-Graduacao/CienciadaInformacao/Dissertacoes/santos_ms_me_mar.pdf>. Acesso em: 11 jul. 2017.

VIÉ, Jean Christophe et al. **The IUCN Red List**: A key conservation tool. Gland, Suiça: IUCN, 2008.

YOCCOZ, Nigel G.; NICHOLS James D.; BOULINIER, Thierry. Monitoring of biological diversity in space and time. **TRENDS in Ecology & Evolution**, Estados Unidos, v. 16, n. 8, p. 446-453, Ago. 2001.

ZEILER, Michael. **Modeling our World**: the ESRI guide to geodatabase design. 1. ed. California: Environmental Systems Research Institute, Inc., 1999.

APÊNDICE A – Questionário do perfil de usuário, lista de tarefas e avalição de usabilidade

Neste apêndice constam o questionário de perfil de usuário, a lista de tarefas executadas pelos voluntários e o questionário de usabilidade aplicado ao final do experimento.

Quadro 25 – Questionário de perfil de usuário

Olá!

Você está sendo convidado a participar de uma avaliação do aplicativo para identificação de cágados Tortuga. Este aplicativo tem como principal objetivo realizar a identificação de cágados da espécie Phrynops Whilliamsi através de um dispositivo Android.

O aplicativo busca facilitar o monitoramento e acompanhamento de cágados permitindo ao usuário manter um histórico com as informações de capturas dos cágados. Cada cágado cadastrado possui um perfil, onde é possível visualizar no Google Maps os locais onde o cágado foi reconhecido e o seu histórico de capturas.

Gostaríamos de sua contribuição para avaliar o aplicativo que foi desenvolvido. Para tanto, pedimos que você responda ao questionário de perfil de usuário e, em seguida, realize as tarefas listadas informando se foi possível ou não executar cada uma delas. Ao final, é apresentado um questionário para você realizar uma avaliação geral do aplicativo.

PERFIL DE USUÁRIO

Sexo: ( ) Feminino ( ) Masculino

Idade:

( ) Tenho menos de 18 anos ( ) Tenho entre 25 e 35 anos

( ) Tenho entre 18 e 25 anos ( ) Tenho mais de 35 anos

Profissão:_____

Você utiliza celular com qual frequência?

( ) Todos os dias ( ) A cada 15 dias

( ) 2 a 3 vezes por semana ( ) 1 vez ao mês

( ) 1 vez por semana

Enumere as funções que você mais utiliza no celular (em ordem de prioridade):

( ) Alarme ( ) Áudio ( ) Calendário

( ) Vídeos ( ) Foto ( ) Agenda

( ) Whatsapp ( ) Facebook ( ) Outros aplicativos

Especifique quais: _____

Você atua em alguma área da biologia?

( ) Sim ( ) Não

Você já realizou o monitoramento e acompanhamento de alguma espécie?

( ) Sim ( ) Não

Em qual aparelho você irá utilizar o aplicativo para a realização das tarefas?

( ) Celular ( ) Tablet

Modelo:_____ **Versão do Android:**_____ **Tamanho da tela:**_____

Quadro 26 – Lista de tarefas

LISTA DE TAREFAS

A seguir, é apresentada uma lista com 14 atividades que têm por objetivo avaliar o aplicativo desenvolvido. Por gentileza, realize o que é solicitado em cada tarefa, indicando se a mesma foi concluída ou não.

1. Cadastrar um novo cágado.

   Ao abrir o aplicativo, navegue pela tela inicial e acesse uma das opções para obter uma imagem das listras do cágado e posteriormente, tente realizar o cadastro do cágado.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

2. Acessar o perfil do cágado cadastrado.

   Após o fim do cadastro do cágado, navegue pela tela inicial e tente acessar/encontrar o cágado que foi cadastrado na tarefa anterior.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

3. Acessar tela para edição do cágado.

   Navegue pela tela que apresenta o perfil do cágado que você cadastrou e tente ir para tela de edição do cágado.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

4. Alterar o cadastro do cágado.

   Navegue pela tela de edição do cágado e tente alterar as informações cadastradas.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

5. Cadastrar acompanhamento para o cágado.

   No perfil do cágado cadastrado acesse a tela de acompanhamentos e tente adicionar um novo acompanhamento para o cágado.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

6. Acessar tela para edição do acompanhamento.

   Navegue pela tela que apresenta os dados do acompanhamento e tente ir para a tela de edição do acompanhamento.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

7. Alterar o cadastro do acompanhamento.

Navegue pela tela de edição do acompanhamento e tente alterar as informações cadastradas.

A tarefa foi executada? Sim, não? Por quê?

_____

_____

8. Alterar a localização do acompanhamento.

   Ainda na tela para edição do acompanhamento tente alterar o local onde o acompanhamento foi cadastrado. Para realizar essa atividade é necessário que o celular tenha conexão com a internet.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

9. Visualizar imagens do acompanhamento.

   Navegue pela tela que apresenta o acompanhamento cadastrado e tente ir para a galeria de imagens do acompanhamento.

   A tarefa foi executada? Sim, não? Por quê?

   _____

   _____

10. Vincular nova imagem ao acompanhamento.

    Na tela que apresenta a galeria de imagens do acompanhamento tente adicionar uma nova imagem.

    A tarefa foi executada? Sim, não? Por quê?

    _____

    _____

11. Visualizar locais de acompanhamento no Google Maps.

    Acesse o perfil do cágado em que foi cadastrado o acompanhamento na atividade 4 e tente visualizar no Google Maps o local onde o acompanhamento foi cadastrado. Para realizar essa atividade é necessário que o celular tenha conexão com a internet.

    A tarefa foi executada? Sim, não? Por quê?

    _____

    _____

12. Exportar os dados do aplicativo para o Google Drive.

    Navegue pela tela inicial do aplicativo e tente exportar os dados do aplicativo para o Google Drive. Para realizar essa atividade é necessário que o celular tenha conexão com a internet.

    A tarefa foi executada? Sim, não? Por quê?

    _____

    _____

13. Importar os dados do Google Drive.

    Novamente na tela inicial, tente importar os dados que foram exportados na tarefa anterior para o aplicativo. Para realizar essa atividade é necessário que o celular tenha conexão com a internet.

    A tarefa foi executada? Sim, não? Por quê?

    _____

14. Visualizar o cágado na aplicação web.

    Desative a conexão com a internet do dispositivo móvel. Agora realize o cadastro de um novo cágado. Ative a conexão com a internet e feche o aplicativo (certifique-se de que ele não esteja executando em segundo plano), feito isso, abra o aplicativo novamente. Acesse o site https://tcctortuga.herokuapp.com e verifique se o cágado que você cadastrou no aplicativo se encontra no site.

    A tarefa foi executada? Sim, não? Por quê?

    _____

    _____

Quadro 27 – Questionário de usabilidade

QUESTIONÁRIO DE AVALIAÇÃO DO APLICATIVO

Após a utilização do aplicativo, você está convidado a responder um questionário de avaliação do mesmo. As respostas deverão ser feitas na tabela abaixo observando às impressões obtidas com a utilização do aplicativo. Você deve responder preenchendo uma das alternativas. Após o questionário com perguntas objetivas, é apresentado um espaço para comentários gerais sobre a ferramenta e sugestões de melhorias.

Perguntas/Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Não concordo nem discordo**	**Discordo parcialmente**	**Discordo totalmente**
O design da interface do aplicativo é atraente					
É fácil navegar pelo aplicativo					
Os símbolos e ícones são claros e intuitivos					
A interface é semelhante dos demais aplicativos					
É fácil realizar os cadastros de cágados e acompanhamentos					
Você acredita que o aplicativo auxilia no monitoramento e acompanhamento de cágados					
As informações armazenadas no aplicativo são facilmente encontradas					
A comparação entre os cágados cadastrados no aplicativo auxilia na identificação do individuo					
Você achou importante a funcionalidade que permite realizar o backup dos dados do aplicativo para o Google Drive					
Você acha que visualizar os locais de captura do cágado no Google Maps ajuda a entender a movimentação do mesmo					
Às vezes não sei o que fazer no aplicativo					
Você precisaria do apoio de uma pessoa para utilizar o aplicativo					
Você utilizaria esse aplicativo para auxiliar no monitoramento e acompanhamento de cágados					
Você recomendaria esse aplicativo para outra pessoa					
Você acredita que a aplicação web ajuda no compartilhamento de informações entre os biólogos					

Qual é a sua opinião sobre o aplicativo quanto ao seu uso e funcionalidades?

Fique à vontade para fazer críticas e sugestões.

_____

_____

_____

_____