UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

CORPUS MAPPING: UM APLICATIVO PARA O MAPEAMENTO CORPORAL DE LESÕES CUTÂNEAS

JANAÍNA CARRARO MENDONÇA LIMA

BLUMENAU
2015

2015/2-11

JANAÍNA CARRARO MENDONÇA LIMA

CORPUS MAPPING: UM APLICATIVO PARA O MAPEAMENTO CORPORAL DE LESÕES CUTÂNEAS

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
2015

2015/2-11

CORPUS MAPPING: UM APLICATIVO PARA O MAPEAMENTO CORPORAL DE LESÕES CUTÂNEAS

Por

JANAÍNA CARRARO MENDONÇA LIMA

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. Dalton Solano dos Reis, Mestre – FURB
Membro:	Prof. Matheus Carvalho Viana, Doutor – FURB

Blumenau, 08 de dezembro de 2015

Dedico este trabalho aos meus pais, que sempre me incentivaram nos estudos e ao meu esposo, que esteve ao meu lado e me apoiou ao longo destes anos de faculdade.

AGRADECIMENTOS

Agradeço a Deus, por ter me concedido a oportunidade de chegar até aqui.

À minha família, em especial aos meus pais, Claudiomiro Mendonça e Eliane Carraro Mendonça, por sempre terem me incentivado a estudar e a lutar por meus objetivos, por apoiarem as minhas decisões e pelo esforço em sempre fazer o melhor por mim e minha irmã.

Ao meu esposo, Ezequiel Pires Lima, que esteve ao meu lado ao longo desta jornada e sempre me apoiou e incentivou, sendo amigo e companheiro em todos os momentos.

Ao meu orientador, Aurélio Hoppe, pela motivação e incentivo, por ter acompanhado cada detalhe e estar sempre disposto a me ajudar, tornando possível a conclusão deste trabalho.

Aos professores do curso de Ciência da Computação, pelo aprendizado que me proporcionaram e que com certeza contribuiu para a realização deste trabalho.

Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes.

Marthin Luther King

RESUMO

Este trabalho descreve o desenvolvimento de um aplicativo Android que realiza o mapeamento corporal de lesões cutâneas, técnica utilizada na detecção precoce do melanoma. O aplicativo faz o mapeamento corporal por meio da captura de imagens com a câmera do dispositivo e da vinculação das mesmas na região corporal onde as pintas foram fotografadas. Os dados são armazenados no próprio dispositivo, com a possibilidade de exportá-los para o Google Drive. Os resultados obtidos a partir dos experimentos e testes de usabilidade demonstram que o aplicativo desenvolvido é capaz de realizar o mapeamento corporal, incluindo imagens de qualquer parte do corpo, e de manter o histórico das pintas ao longo do tempo.

Palavras-chave: Mapeamento corporal. Lesões cutâneas. Android.

ABSTRACT

This document describes the development of an Android application that performs the body mapping of skin lesions, technique used for early detection of melanoma. The application does the body mapping by capturing images with device camera and links them with the body region where the moles were photographed. Data is stored on device, with the possibility of exporting them to Google Drive. The results obtained through experiments and usability tests show that the developed application is able to maintain the body mapping, including images from any body part, and keep track of moles over time.

Key-words: Body mapping. Skin lesions. Android.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ABCD – Assimetria, Borda, Cor e Diâmetro

API – *Application Programming Interface*

SDK – *Software Development Kit*

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

O câncer é hoje no Brasil um problema de saúde pública. Estima-se que em 2015 surgirão 576 mil novos casos de câncer no país, dentre eles 104 mil serão de câncer de pele, sendo que desses, aproximadamente 6% dos casos serão de melanoma (INSTITUTO NACIONAL DO CÂNCER, 2014). Segundo Souza et al. (2009), apesar da menor ocorrência, o melanoma é o mais agressivo entre todos os tipos de câncer de pele, sendo responsável por 75% de todas as mortes causadas por câncer cutâneo, por causa da sua alta capacidade de metástase, isto é, de se espalhar por outros órgãos do corpo.

Goulart et al. (2011) afirma que os melhores resultados para pacientes com melanoma são alcançados quando há detecção precoce. Souza et al. (2009), explica que o diagnóstico precoce é importante para todos os casos, mas principalmente para o melanoma, pois na fase inicial pode ser curável, porém, iniciada a metástase, torna-se praticamente fatal.

Dentre as técnicas de diagnóstico de melanoma, está o mapeamento corporal (*total body photography*), que, segundo Mayer et al. (2014), pode complementar a avaliação por dermatoscopia para a detecção precoce, e auxiliar no autoexame da pele pelos pacientes. O mapeamento consiste em manter um histórico de imagens do corpo do paciente em posições pré-estabelecidas. A partir das imagens é possível acompanhar as mudanças nas pintas existentes bem como o surgimento de novas lesões, que serve de alerta para detectar um possível câncer de pele.

Tyagi, Miller e Cockburn (2012) reforçam que o monitoramento das lesões cutâneas é necessário a fim de reduzir a morbidade e a mortalidade desta doença cuja incidência está aumentando, sendo que poucos avanços foram alcançados no tratamento da mesma.

Diante do cenário exposto acima, este trabalho apresenta o desenvolvimento de um aplicativo para a plataforma Android capaz de realizar o mapeamento corporal. Possibilitando assim o acompanhamento das modificações e surgimento de novas lesões cutâneas, favorecendo o diagnóstico precoce da doença e evitando exames invasivos desnecessários.

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um aplicativo para a plataforma Android que possibilite a realização do mapeamento corporal de lesões cutâneas.

Os objetivos específicos do trabalho são:

a) capturar imagens por meio da câmera do dispositivo móvel, possibilitando a vinculação com a região corporal onde a pinta está localizada;

b) armazenar os dados e as imagens ao longo do tempo para permitir a comparação da evolução das pintas, colocando-os na nuvem;
c) disponibilizar um mecanismo que seja capaz de padronizar a distância ao qual as imagens são capturadas;
d) prover uma interface gráfica que facilite a navegabilidade e visualização das lesões cutâneas, permitindo de forma prática o acompanhamento das pintas existentes e o aparecimento e/ou crescimento de pintas mais escuras e de acúmulo anormal pelo corpo do paciente.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. O primeiro capítulo é composto pela justificativa do trabalho, os objetivos definidos para o mesmo e a apresentação de sua estrutura.

O segundo capítulo apresenta a fundamentação teórica, abordando conceitos sobre o câncer de pele do tipo melanoma e do mapeamento corporal, esclarecendo pontos que devem ser considerados no desenvolvimento do trabalho.

No terceiro capítulo é apresentado o desenvolvimento do aplicativo, onde são listados os requisitos principais e a especificação por meio de diagramas. Na sequência, é detalhada a implementação do aplicativo, descrevendo as técnicas e ferramentas utilizadas, e são apresentados e discutidos os resultados obtidos.

Por fim, o quarto capítulo apresenta as conclusões e as sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em três seções. A seção 2.1 conceitua o câncer de pele e aborda o melanoma. A seção 2.2 expõe o que é o mapeamento corporal e suas vantagens. Na seção 2.3 são apresentados alguns trabalhos correlatos que descrevem abordagens para o diagnóstico de lesões cutâneas através da manutenção do mapeamento corporal.

2.1 CÂNCER DE PELE MELANOMA

De acordo com Pereira (2012a), o câncer de pele é o crescimento anormal e descontrolado das células da pele. Segundo o autor, existem diversos tipos de câncer de pele, sendo possível dividi-los em dois grupos: melanoma e não melanoma.

Segundo o Instituto Nacional do Câncer (2015b), o câncer de pele do tipo não melanoma é o mais frequente no Brasil, correspondendo a 25% de todos os casos de tumores malignos registrados no país. Apesar da maior incidência, é o que apresenta menor taxa de mortalidade. Segundo Pereira (2012a), o câncer de pele não melanoma engloba diversos tipos de câncer, dentre eles, os mais comuns são o carcinoma basocelular e o carcinoma espinocelular.

O melanoma é uma neoplasia maligna que tem comportamento agressivo, origem nos melanócitos (células produtoras de melanina, substância que determina a cor da pele) e predominância em adultos de pele clara. De acordo com o Instituto Nacional do Câncer (2015a), embora o câncer de pele seja um dos mais frequentes no Brasil, o melanoma representa apenas 4% das neoplasias malignas da pele. Contudo, é o mais grave devido a sua alta capacidade de metástase, podendo espalhar-se por outros órgãos do corpo. Devido à detecção precoce do tumor, nos últimos anos houve melhora na sobrevida dos pacientes diagnosticados com melanoma.

Os principais fatores de risco para o melanoma são: pele clara, presença de múltiplos nevos melanocíticos (pintas), histórico familiar de melanoma e a exposição à radiação ultravioleta (PEREIRA, 2012c). Segundo Savory (2015), exames de rotina para rastreamento de câncer de pele em tais indivíduos de alto risco ajudam a detectar melanomas em sua fase inicial, aumentando as chances de sobrevida.

Segundo Castro (2008), durante muito tempo se defendeu a retirada indiscriminada dos nevos, que resultava em uma série de cicatrizes e quase sempre se confirmava o diagnóstico de lesão benigna. A utilização da regra ABCD (assimetria, borda, cor e diâmetro) diminuiu um pouco as excisões, mas mesmo assim muitos procedimentos eram feitos sem necessidade.

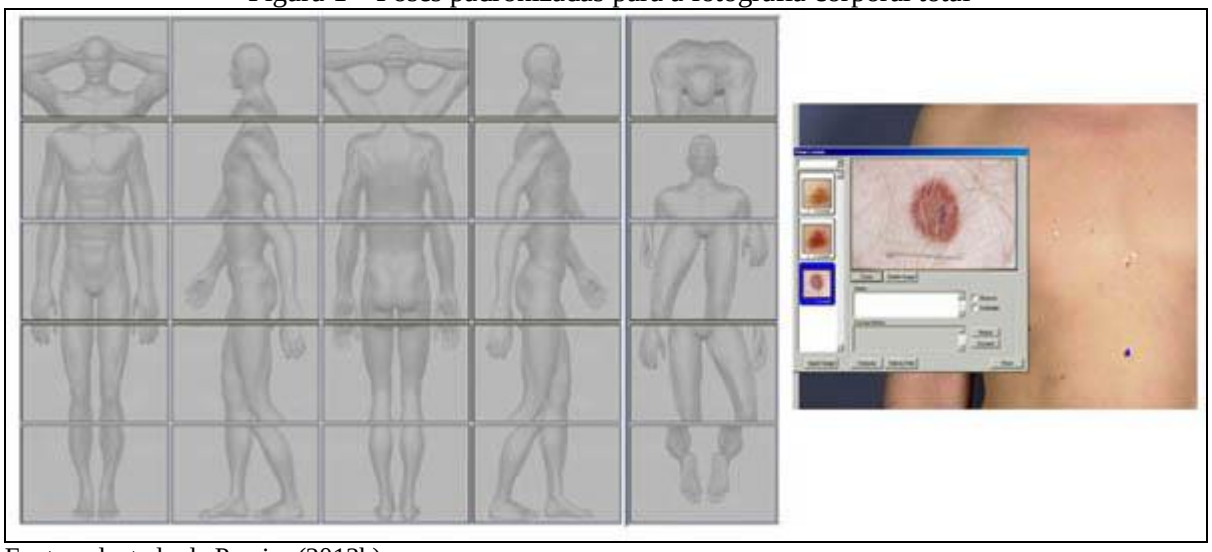
Para Castro (2009), o diagnóstico do melanoma se tornou bem mais sensível e específico com o surgimento da dermatoscopia, método que consiste na busca de lesões cutâneas suspeitas com o auxílio de um dermatoscópio. Contudo, a objetividade desse exame clínico tem sido potencializada com o mapeamento corporal dos nevos, descrito na próxima seção.

2.2 MAPEAMENTO CORPORAL

Conforme exposto anteriormente, os nevos melanocíticos (pintas ou sinais) são considerados potenciais precursores de melanomas. Na maioria das vezes, é muito difícil diferenciar lesões benignas e malignas apenas com o exame clínico. Para ajudar nessa diferenciação, muitos médicos utilizam a dermatoscopia, um método de diagnóstico não invasivo que permite a visualização de estruturas não visíveis ao olho nu (PEREIRA, 2012b). Todavia, em pacientes com muitas pintas se torna muito difícil acompanhar a evolução das lesões apenas com a memória do médico. Para auxiliar nesta situação, existe o mapeamento corporal.

O mapeamento corporal consiste em fotografias digitais de alta resolução de todo o corpo em poses padronizadas (Figura 1), associado à dermatoscopia digital de nevos melanocíticos suspeitos (PEREIRA, 2012b). As imagens são arquivadas em um software que permite a comparação em intervalos de tempo pré-determinados.

Figura 1 – Poses padronizadas para a fotografia corporal total



Fonte: adaptado de Pereira (2012b).

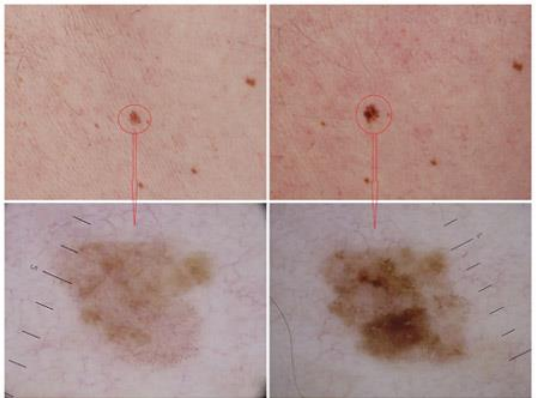
Segundo Pereira (2012b), o objetivo principal das fotografias corporais é criar um mapa de navegação, permitindo que o médico acompanhe as pintas existentes, o surgimento de novas pintas e o acúmulo anormal pelo corpo do paciente. Além das fotos corporais, o autor expõe que algumas pintas são seguidas pela dermatoscopia digital, sendo monitoradas

em determinados intervalos de tempo. Esse procedimento é chamado por Salerni et al. (2012) de "*two-step method*". O autor afirma que o uso desse método em populações com alto risco de desenvolver melanomas é eficiente para detectar a doença em fases iniciais, com uma baixa taxa de excisões. Porém, o acompanhamento em longo prazo é necessário para detecção de melanomas de crescimento lento.

Após a realização do mapeamento corporal e a dermatoscopia, os nevos são classificados em baixo, médio e alto risco. Os de alto risco geralmente são removidos e enviados para biópsia. Os nevos de risco moderado são seguidos de três em três meses, pois uma pequena porcentagem dos nevos benignos pode mudar em três meses, mas a maioria dos melanomas muda nesse período de tempo. Os nevos classificados como baixo risco são seguidos em doze meses, assim como as fotos corporais são repetidas a cada doze meses (PEREIRA, 2012b).

Para Malvehy e Puig (2002), o acompanhamento digital que o mapeamento corporal possibilita, é útil para melhorar a precisão do diagnóstico de lesões com características atípicas, pois ajuda a diferenciar as lesões benignas de melanomas, pois melanomas se alteram significativamente ao longo de um período de tempo relativamente curto (Figura 2).

Figura 2 – Crescimento de nevo preexistente



Melanoma que sofreu mudanças significativas no intervalo de 1 ano

Fonte: adaptado de Pereira (2012).

Para Savory (2015), uma das limitações do mapeamento corporal é o seu alto custo, que pode ser superior a 500 dólares por paciente e, muitas vezes, não é coberto pelos planos de saúde. Além de ser um exame demorado e que geralmente traz benefício apenas a longo prazo.

Apesar dessas limitações, o mapeamento corporal traz muitos benefícios, que segundo Savory (2015) são: a possibilidade do diagnóstico de melanomas no início da doença, a diminuição do número de biópsias e cirurgias desnecessárias, a possibilidade de identificar

mudanças súbitas em nevos pré-existentes, o controle do surgimento de nevos pelo próprio paciente e o aumento da precisão no diagnóstico.

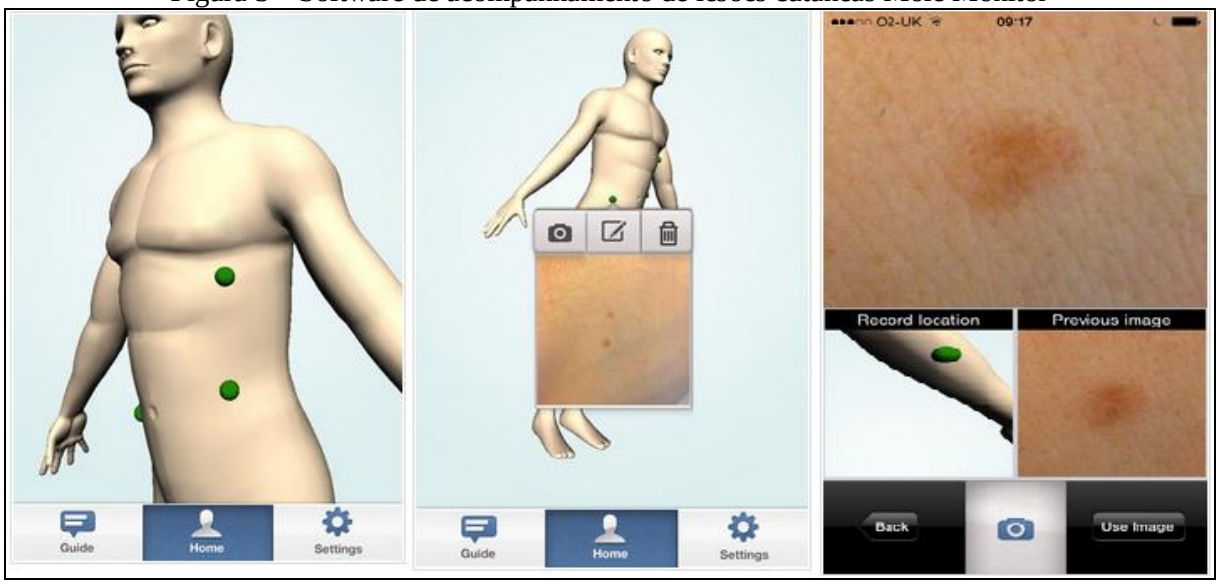
2.3 TRABALHOS CORRELATOS

Esta seção tem como objetivo apresentar alguns trabalhos relacionados ao mapeamento corporal de lesões cutâneas. Dentre os trabalhos selecionados estão: o aplicativo Mole Monitor (PRO-CAL, 2014), que tem como objetivo auxiliar o acompanhamento de nevos melanocíticos; o Handyscope, produto desenvolvido pela Foto Finder (2010a) para dermatoscopia digital e gerenciamento de imagens de lesões cutâneas; e o DermaGraphix, que é o módulo de mapeamento corporal do Mirror, um software completo para imagens médicas, desenvolvido pela Canfield Scientific (2015). Além destes, há o trabalho de Barbosa (2014), que desenvolveu um protótipo para dispositivos móveis com o objetivo de auxiliar no mapeamento cutâneo.

2.3.1 Mole Monitor

O Mole Monitor (PRO-CAL, 2014) é um aplicativo para iOS desenvolvido com o objetivo de auxiliar no acompanhamento de pintas e lesões da pele. O aplicativo apresenta um modelo de corpo humano em 3D, no qual o paciente pode associar uma imagem à determinada região do corpo, ou selecionar a região do corpo que deseja registrar uma nova imagem. A Figura 3 apresenta o fluxo realizado na aplicação, com a visualização de imagens que estão associadas à determinada região do corpo.

Figura 3 – Software de acompanhamento de lesões cutâneas Mole Monitor



Fonte: Pro-Cal (2014).

O Mole Monitor (PRO-CAL, 2014) usa os sensores do dispositivo, em especial o acelerômetro, para controlar a distância em que a foto é capturada. Ele tenta estabelecer um padrão para as imagens e a escala do tamanho das pintas. O controle da distância é feito da seguinte maneira: ao escolher capturar uma nova imagem, o usuário deve posicionar o dispositivo bem próximo à pele, quando ouvir um bipe, o usuário deve mover o telefone de forma de se distancie da pele, quando ouvir um segundo bipe, significa que já pode tirar a foto. Se o usuário mover o telefone muito rápido ou muito devagar, irá ouvir dois bipes e isso significa que ele deve repetir o procedimento. Esta funcionalidade pode ser desativada pelo usuário.

O Mole Monitor não apresenta opção para cadastrar mais de um paciente, o objetivo é que o próprio paciente faça a captura das imagens e acompanhe a evolução das lesões. O histórico gerado servirá como um auxílio no momento da consulta médica, em que podem ser utilizados relatórios gerados pelo aplicativo, que incluem as imagens das lesões e o histórico médico do paciente.

O aplicativo apresenta ainda ferramentas para comparação das imagens, além de um mecanismo que notifica o usuário quando está na hora de atualizar os registros. É possível avaliar as lesões através da regra ABCDE, porém a classificação não é automática, o usuário escolhe avaliar a lesão, e o aplicativo guia o usuário por meio de perguntas que devem ser respondidas pelo próprio usuário.

2.3.2 Handyscope

O handyscope (FOTO FINDER, 2010a) é um produto para dermatoscopia digital desenvolvido pela FotoFinder Systems que permite capturar imagens brilhantes polarizadas e não polarizadas com até 20x de ampliação dos nevos. Fisicamente, o handyscope se constitui em um estojo que comporta um conjunto de lentes especiais que é alinhado com a lente da câmera do *smartphone*, e um conjunto de LEDS quem proveem iluminação por luz polarizada. A Figura 4 apresenta como é o acessório.

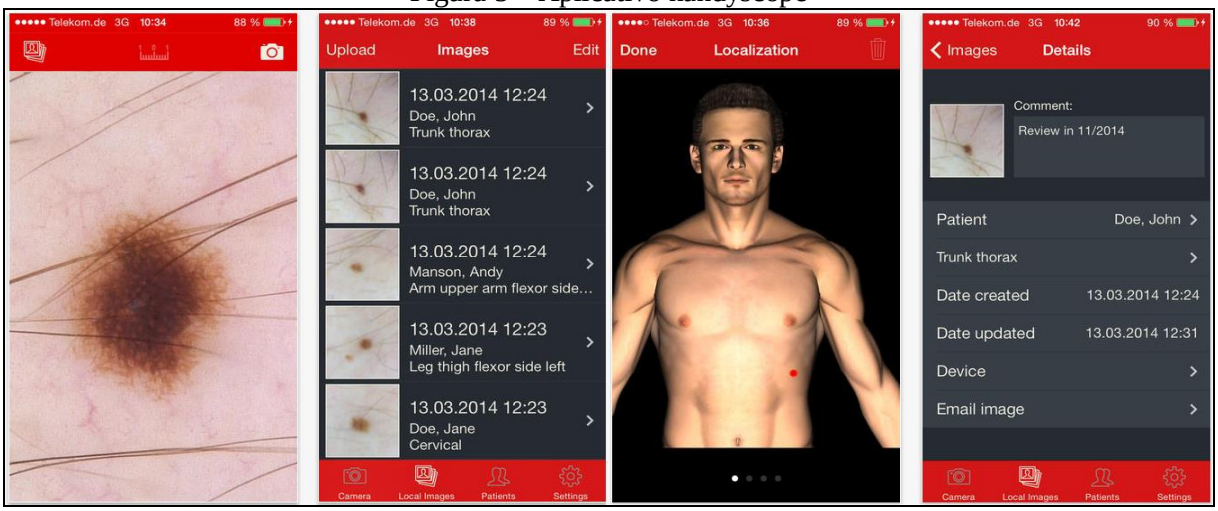
Figura 4 – Acessório físico do handyscope encaixado em um iPhone

Fonte: Foto Finder (2010b).

O aplicativo, disponível apenas para iPhone e iPod touch, organiza as imagens capturadas em ordem cronológica e permite ao profissional marcar as imagens com dados do

paciente, localização e comentários. No momento da captura da imagem, o usuário tem a opção de utilizar uma escala na tela, que permite medir o tamanho das pintas com mais facilidade. É possível associar as imagens à parte do corpo correspondente, por meio de um modelo de corpo humano em 3D. O handyscope também permite o envio de e-mails com as imagens, facilitando o compartilhamento entre profissionais, estudantes e pacientes. A Figura 5 apresenta algumas telas do aplicativo handyscope, onde é possível ver uma das imagens capturadas, a lista de todas as imagens e seus detalhes, além do modelo de ser humano virtual.

Figura 5 – Aplicativo handyscope



Fonte: Foto Finder (2010a).

O profissional tem a opção de fazer *upload* das imagens para o FotoFinder Hub (FOTO FINDER, 2010c), que é uma plataforma na nuvem para gerenciamento e análise das imagens. Dependendo do plano escolhido, é possível comparar as imagens lado à lado, gerar relatórios, além de disponibilizar a opção de obter uma segunda opinião de profissionais especialistas em dermatoscopia. Até 24 horas após a solicitação de segunda opinião o médico recebe o resultado da classificação realizada por esses especialistas.

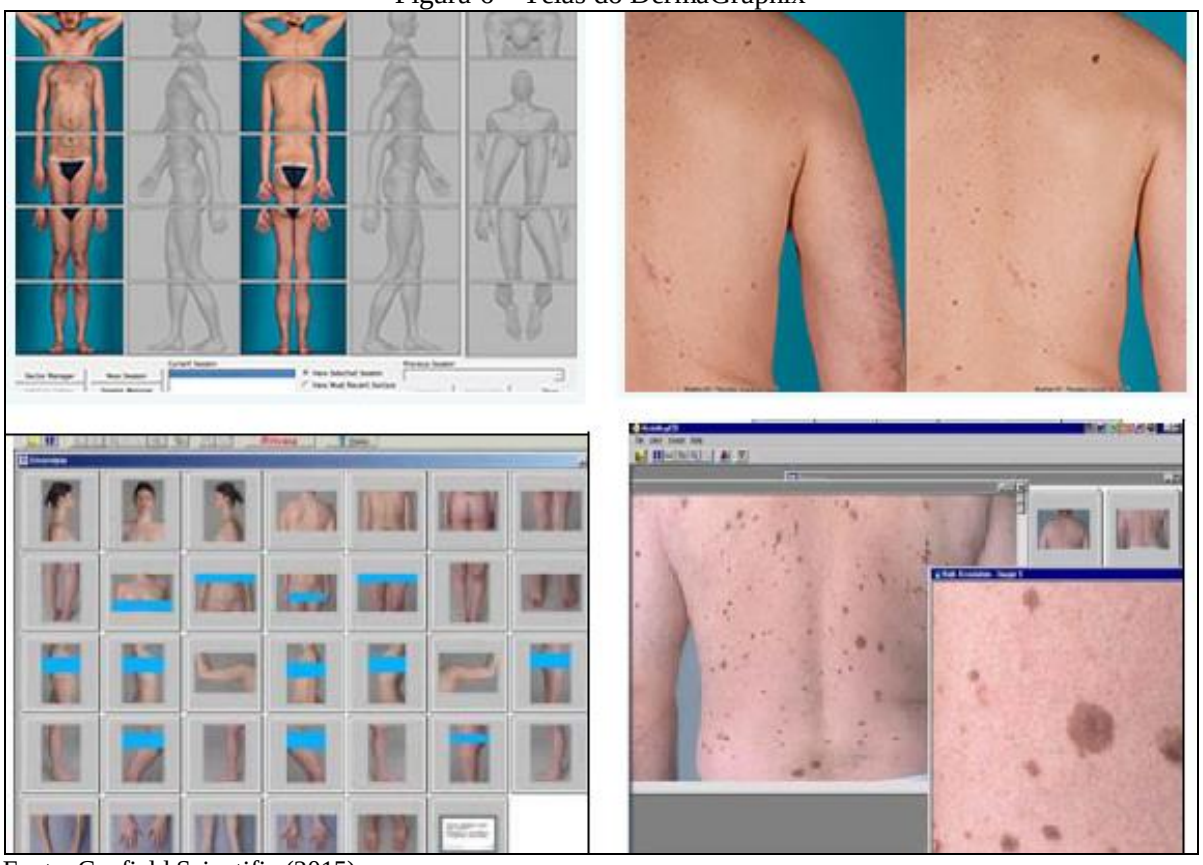
2.3.3 DermaGraphix

O Mirror (CANFIELD SCIENTIFIC, 2015), é um software completo para imagens relacionadas à medicina, conhecido e reconhecido mundialmente pela comunidade médica. É composto por vários módulos, dentre os quais o DermaGraphix, que é o módulo de mapeamento corporal, construído com o objetivo de oferecer uma ferramenta para acompanhar pacientes com risco de desenvolver melanoma. Esse módulo se integra ao Mirror PhotoFile, que é o módulo que faz o gerenciamento das imagens.

O DermaGraphix apresenta as posições padronizadas para o mapeamento corporal em imagens do corpo humano e após o mapeamento, substitui pelas imagens do próprio paciente.

É possível comparar lado a lado as imagens capturadas em diferentes períodos para assim identificar novas lesões ou alterações nas lesões pré-existentes. A Figura 6 mostra algumas telas do software.

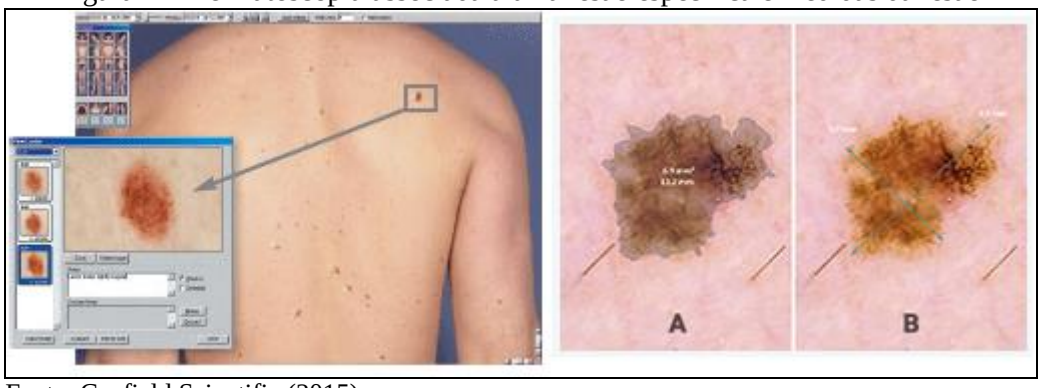
Figura 6 – Telas do DermaGraphix



Fonte: Canfield Scientific (2015).

Caso identifique lesões suspeitas, o médico pode realizar a dermatoscopia destas lesões e associar a imagem da dermatoscopia a uma lesão específica na foto do mapeamento corporal. Na Figura 7 é possível ver um exemplo desse recurso e como o software é capaz de detectar e medir as lesões.

Figura 7 – Dermatoscopia associada a uma lesão específica e medidas da lesão



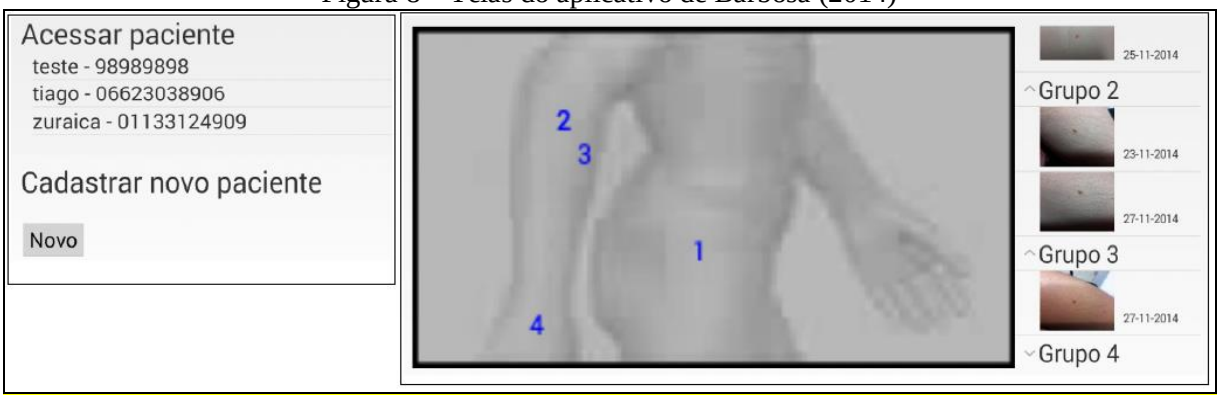
Fonte: Canfield Scientific (2015).

A partir de um conjunto de imagens coletadas em diferentes períodos, surge o histórico do paciente, com o qual é possível comparar as mudanças na pele do mesmo bem como as lesões existentes. Com essa comparação, é possível mensurar a evolução de cada lesão da qual há suspeitas de algum malefício à saúde. A ferramenta possui também a opção de criar um CD com o mapeamento corporal para entregar ao paciente, assim o próprio paciente pode se auto examinar em intervalos de tempo menores até fazer o mapeamento completo novamente.

2.3.4 Mapeamento corporal de lesões cutâneas desenvolvido por Barbosa

O trabalho de Barbosa (2014) propôs o desenvolvimento de um protótipo para dispositivos móveis, especificamente para a plataforma Android, que permite a realização do mapeamento corporal de lesões cutâneas. O mapeamento ocorre a partir da vinculação de imagens a regiões específicas do corpo. O usuário seleciona a região do corpo e o aplicativo acessa a câmera do dispositivo. Após a captura da imagem, a mesma é vinculada com a região do corpo previamente selecionada. A Figura 8 apresenta a tela inicial do aplicativo e a tela de grupos de imagens por região do corpo.

Figura 8 – Telas do aplicativo de Barbosa (2014)



Fonte: Barbosa (2014).

O protótipo alcançou seu objetivo, permitindo que o mapeamento corporal seja realizado e mantido um histórico para acompanhar a evolução das pintas pelo corpo do paciente. Contudo, foram identificadas possíveis melhorias para o aplicativo, como: permitir ao usuário visualizar duas imagens simultaneamente para fins de comparação, possibilidade de manter backup das imagens na nuvem e melhorar a usabilidade do aplicativo.

2.3.5 Comparativo entre os trabalhos correlatos

A partir das informações obtidas com os trabalhos descritos, foi montado o Quadro 1 com as principais características de cada trabalho descrito anteriormente.

Quadro 1 – Características dos trabalhos relacionados

Características / trabalhos relacionados	Pro-Cal (2014)	Foto Finder (2010a)	Canfield Scientific (2015)	Barbosa (2014)
Portável para dispositivos	Sim	Sim	Não	Sim
Armazenamento	Dispositivo	Dispositivo e Nuvem	Servidor próprio	Dispositivo
Controle da distância	Sim	Não	-	Não
Usuário alvo	Paciente	Médico	Médico	Médico e Paciente
Dermatoscopia	Não	Sim	Sim	Não
Posições padronizadas para o mapeamento corporal	Não	Não	Sim	Sim

A partir do Quadro 1, conclui-se que dos softwares analisados, o DermaGraphix (CANFIELD SCIENTIFIC, 2015) é o mais completo quando se trata do mapeamento corporal, porém é o único que não possui suporte para dispositivos móveis. O Mole Monitor (PRO-CAL, 2014) e o protótipo de Barbosa (2014) armazenam os dados apenas no próprio dispositivo, sendo que o primeiro é para uso dos próprios pacientes e o segundo permite o cadastro de vários pacientes, tornando possível o uso por parte de médicos também. O Handyscope (FOTO FINDER, 2010a) é para uso dos médicos no acompanhamento das lesões de diferentes pacientes, com acessório que se acopla ao dispositivo permitindo que seja realizada a dermatoscopia sem o uso de um dermatoscópio tradicional, além de ser possível armazenar os dados na nuvem.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas do desenvolvimento do aplicativo. Na seção 3.1 são enumerados os principais requisitos do projeto desenvolvido. A seção 3.2 apresenta a especificação do aplicativo com diagramas de casos de uso e de pacotes. A seção 3.3 detalha a implementação do aplicativo, destacando os principais pontos. Por fim, a seção 3.4 apresenta os experimentos realizados e resultados obtidos.

3.1 REQUISITOS

O aplicativo desenvolvido deve:

a) permitir ao usuário tirar fotos do corpo a partir da câmera de um dispositivo Android (Requisito Funcional – RF);
b) permitir o cadastro de pacientes (RF);
c) exibir partes do corpo para que o usuário possa vincular a imagem capturada a uma determinada região corporal (RF);
d) permitir o usuário indicar o risco das pintas e inserir anotações em cada uma delas (RF);
e) permitir a consulta das imagens capturadas por meio da navegação pelas partes do corpo (RF);
f) permitir ao usuário visualizar duas imagens simultaneamente (RF);
g) permitir que o usuário faça backup dos dados para uma plataforma na nuvem (RF);
h) permitir a importação dos dados de backup exportados para a nuvem (RF);
i) disponibilizar ao usuário um gabarito que sirva de referência para estimar o tamanho da pinta (RF);
j) ser desenvolvido para a plataforma Android (Requisito Não Funcional - RNF);
k) utilizar o banco de dados SQLite para persistir os dados (RNF).

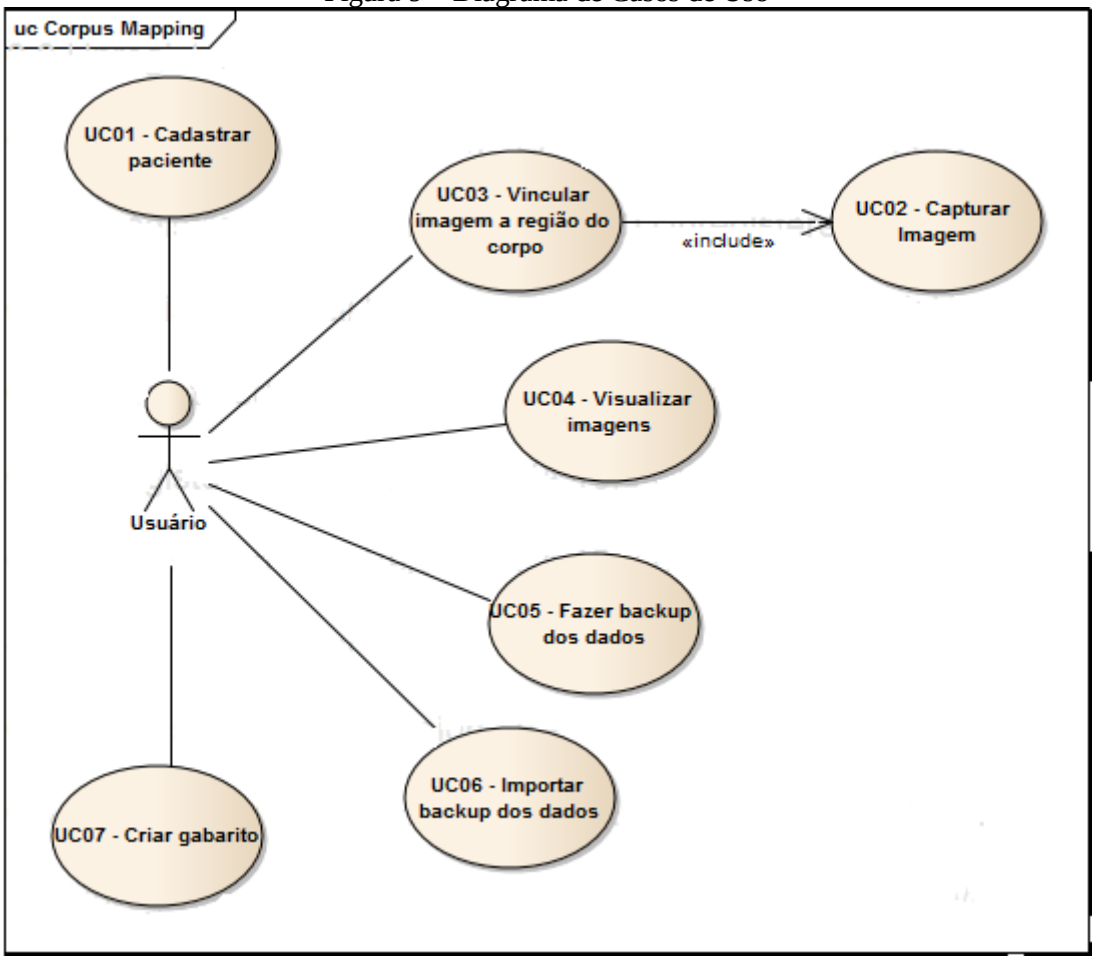
3.2 ESPECIFICAÇÃO

A especificação do aplicativo foi representada em diagramas da *Unified Modeling Language* (UML), utilizando a ferramenta Enterprise Architect. Foram elaborados os diagramas de casos de uso e de pacotes, sendo descritos nas próximas seções.

3.2.1 Casos de Uso

Nesta seção são descritos os casos de uso do protótipo, ilustrados na Figura 9. Identificou-se apenas um ator, denominado `Usuário`, o qual utiliza todas as funcionalidades do protótipo.

Figura 9 – Diagrama de Casos de Uso



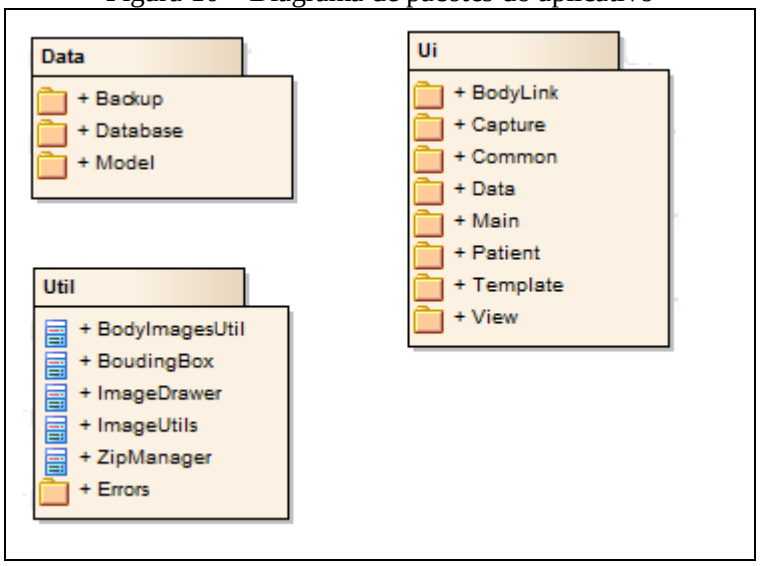
No caso de uso `UC01 - Cadastrar Paciente`, o usuário pode realizar o cadastro de novos pacientes. No caso de uso `UC02 - Capturar imagem`, o usuário realiza a captura de imagens por meio da câmera do dispositivo. No caso de uso `UC03 - Vincular imagem a região do corpo`, o usuário realiza a vinculação da imagem a uma determinada região do corpo. No caso de uso `UC04 - Visualizar imagens`, o usuário visualiza as imagens capturadas através da navegação pelo diagrama corporal. No caso de uso `UC05 - Fazer backup dos dados`, o usuário cria um backup dos dados, exportando-os para o Google Drive. O caso de uso `UC06 - Importar backup dos dados`, o usuário importa um backup dos dados realizado anteriormente. No caso de uso `UC07 - Criar Gabarito` o usuário cria, com as orientações do aplicativo, o gabarito a ser utilizado na captura das imagens.

O detalhamento dos casos de uso do aplicativo pode ser consultado no Apêndice A. Nele estão descritas as pré-condições, os cenários e os fluxos alternativos de cada caso de uso.

3.2.2 Diagrama de Pacotes

Para melhor organização do código, optou-se por separar as classes do aplicativo em diferentes pacotes, de acordo com sua especificidade. A Figura 10 exibe o diagrama de pacotes que compõem o aplicativo.

Figura 10 – Diagrama de pacotes do aplicativo

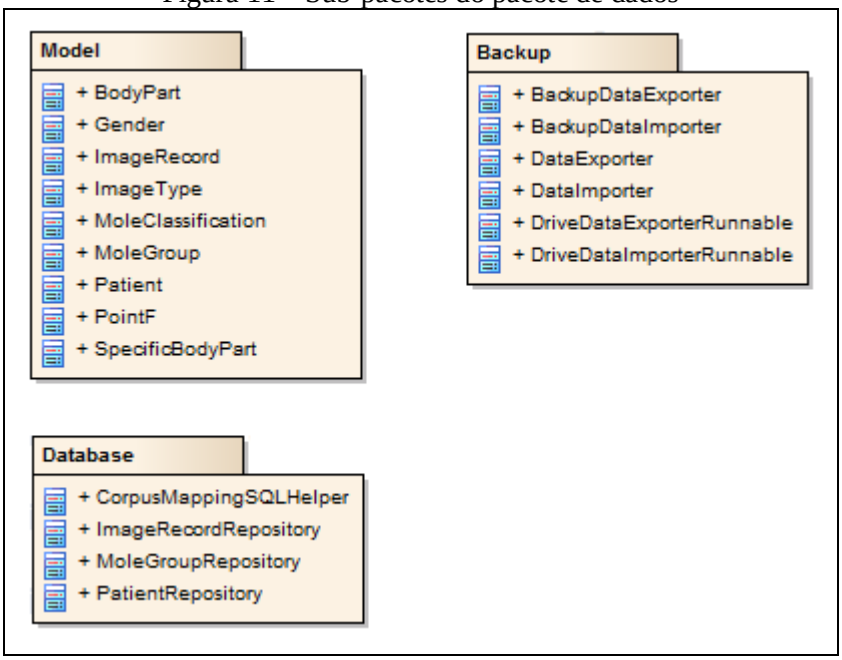


Como é possível ver no diagrama, os pacotes possuem dentro deles sub-pacotes que dividem as classes conforme sua especialidade. As próximas seções destinam-se ao detalhamento de cada pacote, destacando a responsabilidade de cada classe dentro do aplicativo. A seção 3.2.2.1 descreve como o pacote de dados está estruturado e a função de cada classe dentro do mesmo. Na seção 3.2.2.2 é apresentado o pacote de interface do usuário, seus sub-pacotes e responsabilidades das classes nele contidas. Por fim, a seção 3.2.2.3 descreve as classes do pacote de utilitários.

3.2.2.1 Pacote de dados

O pacote `Data` é responsável pelas classes de acesso a base de dados, classes de modelos de dados e classes de exportação e importação de backup. Esse pacote foi dividido em três sub-pacotes, conforme o diagrama da Figura 11.

Figura 11 – Sub-pacotes do pacote de dados



O pacote `Model` contém as classes que representam as estruturas de dados utilizados no aplicativo. A classe `Patient` representa um paciente e contém os atributos necessários para manter o cadastro do paciente. A classe `ImageRecord` representa os dados da imagem de uma pinta e contém atributos como data e hora da captura da imagem, caminho do arquivo da imagem salvo no cartão de memória e grupo ao qual essa imagem pertence. A classe `MoleGroup` representa o grupo de imagens de uma pinta, engloba atributos como `id` do paciente, parte do corpo a que a pinta está associada, posição `xy` da pinta e classificação de risco. A classe `PointF` representa um ponto com as coordenadas `xy` e é utilizada para armazenar a posição de uma pinta.

As demais classes do pacote `Model` são enumerações utilizadas por outras classes. A classe `BodyPart` contém itens que representam as partes do corpo humano. Os itens da classe `SpecificBodyPart` representam as partes do corpo humano e suas variações, por exemplo, para a cabeça, existem quatro variações: de frente, de costas, face direita e face esquerda. A classe `Gender` representa o sexo do paciente. A classe `MoleClassification` contém as opções disponíveis para a classificação de risco de uma pinta. Por fim, a classe `ImageType` representa se uma imagem é do tipo local ou global. Na Figura 31 do Apêndice B é apresentado o diagrama de classes do pacote `Model`.

O pacote `Database` é responsável pelas classes de acesso a base de dados. A classe `CorpusMappingSQLHelper` estende a classe `SQLiteOpenHelper` do Android e é responsável pela criação e versionamento da base de dados. As classes `PatientRepository`, `ImageRecordRepository` e `MoleGroupRepository` são responsáveis por obter, inserir e

alterar dados das tabelas que armazenam os dados de pacientes, imagens e grupos. A Figura 32 do Apêndice B apresenta o diagrama de classes do pacote `Database`.

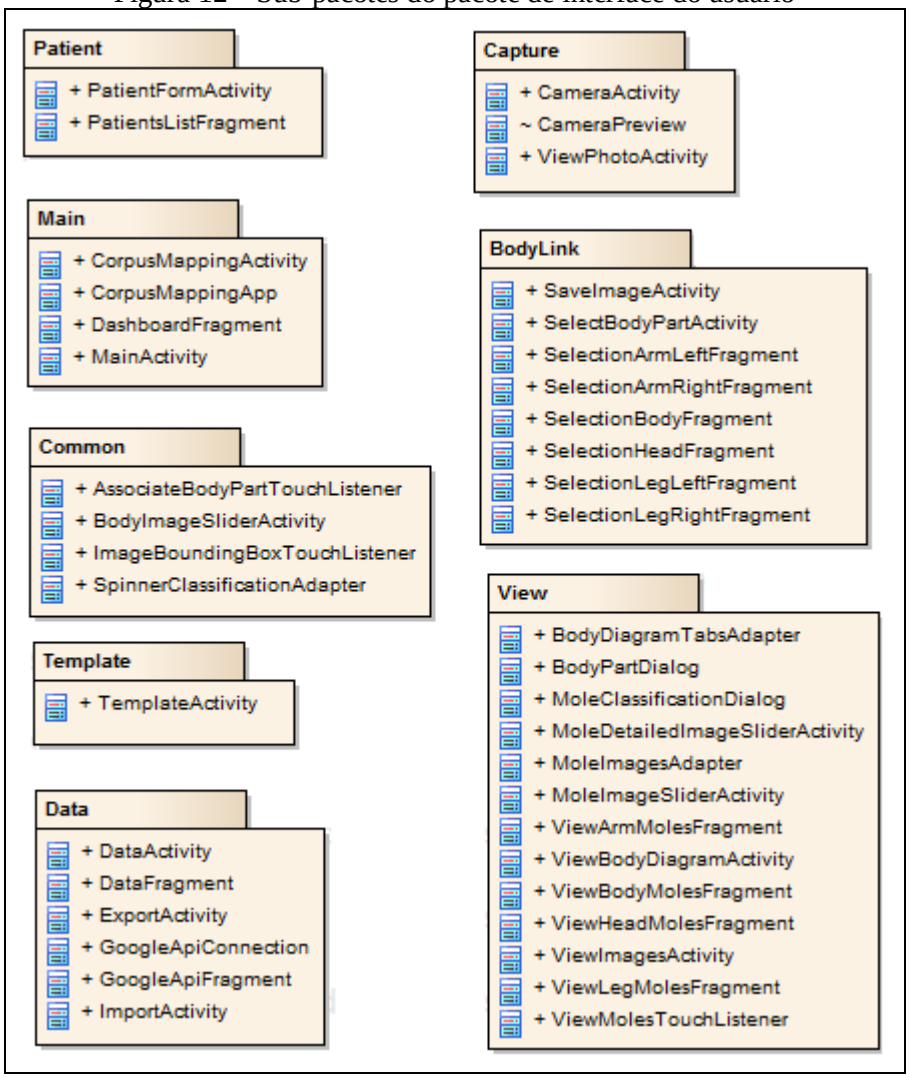
O pacote `Backup` contém as classes responsáveis pela exportação e importação dos dados de backup do Google Drive. A classe `BackupDataExporter` implementa a interface `DataExporter` e é responsável por ler os dados da base de dados, transformá-los para o formato JSON, ler as imagens do cartão de memória, compactá-las em um arquivo zip e escrever os dados em um `OutputStream` recebido da classe `DriveDataExporterRunnable`. A classe `DriveDataExporterRunnable` implementa a interface `Runnable` do Java, pois é executada em outra *thread*. A execução ocorre após o usuário selecionar em qual diretório do Google Drive os dados devem ser salvos. Essa classe chama a classe `BackupDataExporter` informando o `OutputStream` onde os dados devem ser escritos e após esse procedimento realiza o *upload* para o Drive.

A classe `BackupDataImporter` implementa a interface `DataImporter`, sua responsabilidade é ler o `InputStream` que contém os dados em JSON e o arquivo zip com as imagens e realizar a importação dessas informações para a base de dados do aplicativo. Sua execução é acionada pela classe `DriveDataImporterRunnable`, que implementa a interface `Runnable` do Java e é executada após o usuário selecionar em qual diretório do Google Drive estão os dados a serem importados. Na Figura 33 do Apêndice B é possível ver o diagrama de classes do pacote `Backup`.

3.2.2.2 Pacote de interface do usuário

O pacote de interface do usuário (`UI`) abrange as classes do tipo *activities* e *fragments*, que basicamente formam as telas apresentadas no aplicativo. Optou-se por organizar as classes em sub-pacotes conforme suas funcionalidades. A Figura 12 exibe os sub-pacotes criados. Nos próximos parágrafos serão detalhadas as classes internas de cada sub-pacote, elucidando suas responsabilidades dentro do aplicativo.

Figura 12 – Sub-pacotes do pacote de interface do usuário



O pacote `Patient` abrange o cadastro e a seleção de pacientes. A classe `PatientFormActivity` é responsável por apresentar uma tela com os campos necessários ao cadastro do paciente. A classe `PatientListFragment` apresenta os pacientes cadastrados em um componente `ListView` e é utilizada pela classe `MainActivity` do pacote `Main`. A Figura 34 do Apêndice B apresenta o diagrama de classes do pacote `Patient`.

O pacote `Capture` engloba a implementação da funcionalidade de captura da imagem. A classe `CameraActivity` é responsável por abrir a câmera do dispositivo e, em conjunto com a classe `CameraPreview`, define onde será realizada a renderização do *preview* da câmera. Após a captura da imagem, a classe `CameraActivity` chama a abertura da `ViewPhotoActivity`. A classe `ViewPhotoActivity` realiza um processamento na imagem capturada para posteriormente apresentá-la ao usuário. A imagem é recortada para manter apenas a região de interesse e então é apresentada para o usuário, permitindo que ele confirme

o uso da mesma ou descarte-a. O diagrama de classes do pacote `Capture` é apresentado na Figura 35 do Apêndice B.

As classes do pacote `BodyLink`, são responsáveis pela vinculação da imagem capturada com uma região corporal. A classe `SaveImageActivity` é invocada pela `ViewPhotoActivity` do pacote `Capture` quando o usuário confirma a utilização da imagem. Sua responsabilidade é salvar a imagem capturada, porém, antes de salvá-la é necessário saber a localização da pinta. Por isso, é apresentada uma tela com o desenho do corpo humano para que o usuário selecione a parte do corpo onde a pinta está localizada.

Após a seleção da parte do corpo, é preciso saber o local específico da pinta. Desta forma, é chamada a abertura da tela `SelectBodyPartActivity`, que dependendo da parte do corpo selecionada, adiciona um *fragment* diferente na tela. Se for selecionada a região da cabeça, é adicionado o `SelectionHeadFragment`, que apresenta a cabeça em diferentes posições. Se a região selecionada for o tronco, é utilizado o `SelectionBodyFragment`, que apresenta o tronco de frente e de costas. Caso a região selecionada for um dos braços, é apresentado o *fragment* correspondente com o lado selecionado, que pode ser o `SelectionArmRightFragment` para o braço direito, ou o `SelectionArmLeftFragment` para o braço esquerdo. O mesmo para a região das pernas, será apresentado o `SelecionLegRightFragment` para a perna direita, e para a perna esquerda o `SelecionLegLeftFragment`. Na Figura 36 do Apêndice B é apresentado o diagrama de classes do pacote `BodyLink`.

O pacote `View` engloba as classes que permitem a visualização das imagens e consulta do histórico das pintas. A classe `ViewBodyDiagramActivity` utiliza a classe `BodyDiagramTabsAdapter` para apresentar as partes do corpo na forma de abas, cada aba contém um *fragment* que apresenta uma parte do corpo. Na aba da cabeça é utilizado o `ViewHeadMolesFragment`. Na aba do tronco é utilizado o `ViewBodyMolesFragment`. A próxima aba é a dos braços, que utiliza o `ViewArmMolesFragment`. E, por fim, a aba das pernas utiliza o `ViewLegMolesFragment`.

Cada *fragment* desenha nas partes do corpo pontos representando os grupos de pintas vinculadas em cada região. Para visualizar as imagens de um grupo, o usuário deve tocar sobre o ponto no desenho. Neste momento, é aberta a `MoleImageSliderActivity`, responsável por apresentar o histórico de imagens de uma pinta. São apresentadas duas imagens simultaneamente, permitindo a comparação entre imagens capturadas em datas distintas. Para visualizar uma imagem de forma ampliada, o usuário deve selecionar a imagem

desejada, então é aberta a `MoleDetailedImageSliderActivity`, que apresenta a imagem ampliada. Além disso, também permite navegar por outras imagens, capturar uma nova imagem para o grupo e inserir anotações. A Figura 37 do Apêndice B exibe o diagrama de classes do pacote `View`.

O pacote `Data` abrange as funcionalidades de criação e importação de backup dos dados. A classe `DataActivity` juntamente com a classe `DataFragment`, é responsável por apresentar as opções disponíveis: criar backup, importar backup e limpar e restaurar os dados. Quando a opção escolhida é a criar backup, é chamado a abertura da `ExportActivity`, que inicia o processo de exportação dos dados para o Google Drive. Para as opções de importar backup e limpar e restaurar os dados é chamado a abertura da `ImportActivity`, que inicia o processo de importação dos dados do Google Drive para o aplicativo. As classes `GoogleApiFragment` e `GoogleApiConnection` são utilizadas pelas classes `ExportActivity` e `ImportActivity` para gerenciar a conexão com os serviços do Google. Na Figura 38 do Apêndice B é apresentado o diagrama de classes do pacote `Data`.

O pacote `Template` contém uma única classe, a `TemplateActivity`, responsável por apresentar ao usuário uma tela com orientações de como criar o gabarito a ser utilizado na captura das imagens. A Figura 39 do Apêndice B é apresenta o diagrama de classes do pacote `Template`.

O pacote `Main` compreende a classe inicial do aplicativo e as classes de acesso às funcionalidades do mesmo. A classe `MainActivity` é a primeira tela aberta no aplicativo, ela utiliza o `PatientListFragment` do pacote `Patient` para listar os pacientes e solicita que o usuário selecione um paciente para continuar. Após selecionar um paciente, o `id` do mesmo é definido na classe `CorpusMappingApp`. Em seguida, é chamado a abertura da `CorpusMappingActivity`, que apresenta as opções de acesso as funcionalidades do aplicativo através de um menu lateral e utiliza o `DashboardFragment` para apresentar as principais opções do aplicativo em um painel. Na Figura 40 do Apêndice B é possível consultar o diagrama de classes do pacote `Main`.

O Pacote `Common` contém classes comuns para diversas funcionalidades. A classe `ImageBoundingBoxTouchListener` é responsável por verificar se um toque do usuário foi realizado dentro de alguma *bounding box*. A classe `AssociateBodyPartTouchListener` estende a classe `ImageBoundingBoxTouchListener` e é utilizada tanto na vinculação da imagem com uma região corporal quanto na visualização das imagens na consulta do mapeamento corporal. A classe `BodyImageSliderActivity` apresenta os desenhos das partes

do corpo de forma ampliada, sendo utilizada na vinculação e na consulta do mapeamento. Por fim, a classe `SpinnerClassificationAdapter` disponibiliza as opções de classificação de risco de uma pinta, sendo utilizada na vinculação, no momento de salvar uma imagem e na visualização, quando é permitido alterar os dados da pinta. A A seguir, a **Erro! Autoreferência de indicador não válida.** apresenta o diagrama do pacote `Common`. A classe `ImageBoundingBoxTouchListener` é responsável por verificar se um toque do usuário foi realizado dentro de alguma *bounding box*. A classe `AssociateBodyPartTouchListener` estende a classe `ImageBoundingBoxTouchListener` e é utilizada tanto na vinculação da imagem com uma região corporal quanto na visualização das imagens na consulta do mapeamento corporal.

Figura 41 do Apêndice B exibe o diagrama de classes do pacote `Common`.

3.2.2.3 Pacote de utilitários

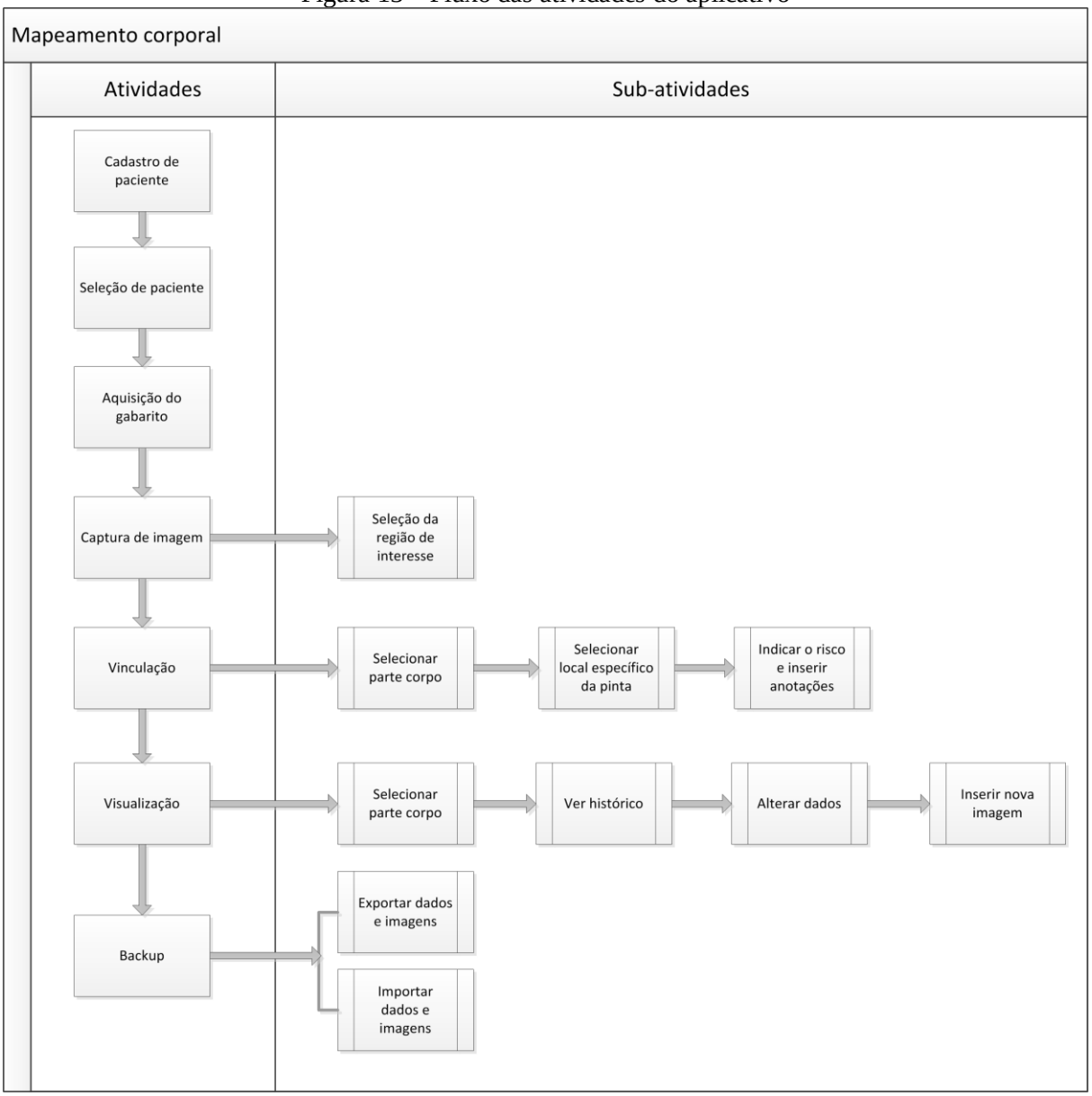
O pacote `Util` agrupa classes auxiliares com funções que são utilizadas pelas outras classes do aplicativo. A classe `ImageUtils` contém métodos para manipulação de imagens, como recorte e obtenção do caminho onde uma imagem está salva no cartão de memória. A classe `ImageDrawer` disponibiliza funções para desenhar pontos que representam as pintas nos desenhos das partes do corpo.

A classe `ZipManager` contém um método para compactar uma lista de arquivos em um arquivo zip. A classe `BoudingBox` guarda os valores mínimos e máximos de uma *bounding box*. Por fim, a classe `BodyImagesUtil` disponibiliza métodos para obter as *bounding boxes* de cada região corporal. A Figura 42 do Apêndice B apresenta o diagrama de classes do pacote `Util`.

3.2.3 Fluxo de atividades

Para melhor entendimento do funcionamento do aplicativo, optou-se pela construção de um fluxo de atividades, conforme demonstrado na Figura 13.

Figura 13 – Fluxo das atividades do aplicativo



A primeira atividade é a de cadastro dos pacientes. Posteriormente, o usuário deve selecionar um dos pacientes cadastrados para ter acesso às demais funcionalidades. Antes de capturar a primeira imagem, é necessário que o usuário crie o gabarito que deverá estar presente nas imagens, o aplicativo fornece as instruções para realizar esta ação.

Após a criação do gabarito, o usuário pode capturar imagens. A captura envolve a seleção da região de interesse por parte do aplicativo. Em seguida, é necessário vincular a imagem capturada com uma região corporal. Primeiramente, deve-se selecionar a parte do corpo, depois o local específico da pinta e então o aplicativo fornece a opção de inserir anotações e classificar o risco da pinta.

Na sequência, tem-se a atividade de visualização das imagens, no qual o usuário seleciona a parte do corpo que deseja consultar e em seguida pode consultar o histórico das

pintas, inserir anotações e capturar uma nova imagem que já ficará associada com o local da pinta sendo visualizada.

Por fim, há a atividade de backup dos dados, que é subdividida em duas etapas: criação do backup (exportação dos dados), e a importação de um backup criado anteriormente, permitindo a restauração de uma possível perda de dados.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O aplicativo foi desenvolvido utilizando a linguagem de programação Java, no ambiente de desenvolvimento Android Studio na versão 1.1.0. O Android Studio é a ferramenta de desenvolvimento oficial de aplicativos Android e é disponibilizada pelo próprio Google.

Optou-se por utilizar a versão 21 do *Software Development Kit* (SDK) do Android. A versão alvo é a 21 e a mínima é a 16, desta forma, o aplicativo é compatível com dispositivos que possuem a versão 4.1 do Android ou superior.

Para a implementação da funcionalidade de backup dos dados, foi utilizada a *Application Programming Interface* (API) base dos serviços do Google (*play services*) e a API do Google Drive na versão 6.5.

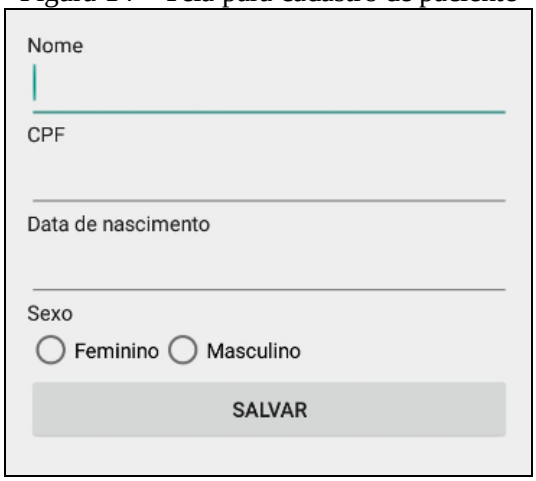
3.3.2 Etapas da implementação

O funcionamento do aplicativo foi dividido em sete etapas. A primeira etapa é o cadastro de pacientes, descrito na seção 3.3.2.1. Na sequência, segue a etapa de seleção do paciente, descrita na seção 3.3.2.2. Logo após, a etapa descrita na seção 3.3.2.3 é a de aquisição do gabarito a ser utilizado na captura das imagens. A quarta etapa é a de captura da imagem, descrita na seção 3.3.2.4. A próxima etapa é a de vinculação da imagem capturada, descrita na seção 3.3.2.5. A sexta etapa, descrita na seção 3.3.2.6 é a de visualização das imagens. Por fim, a sétima e última etapa, é a de backup dos dados do aplicativo, descrita na seção 3.3.2.7.

3.3.2.1 Cadastro de pacientes

O cadastro de pacientes abrange as informações de: nome, CPF, data de nascimento e sexo. A Figura 14 apresenta a tela para cadastro de novo paciente.

Figura 14 – Tela para cadastro de paciente

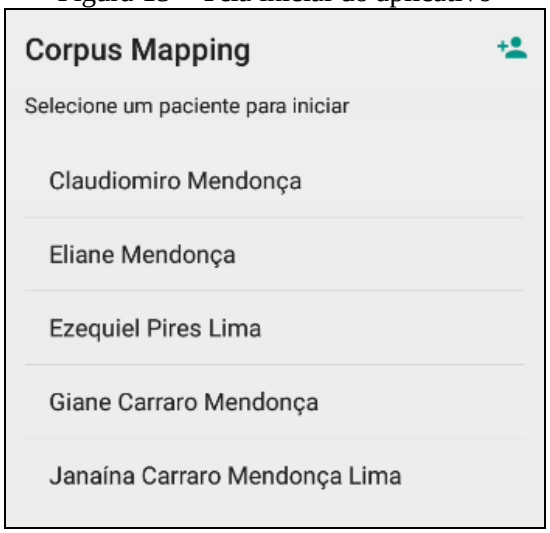


A *activity* que forma a tela de cadastro é a `PatientFormActivity`. Os pacientes são salvos em uma tabela do SQLite por meio da classe `PatientRepository`.

3.3.2.2 Seleção de paciente

Os pacientes cadastrados são apresentados em uma lista na tela inicial do aplicativo, onde é exibida uma mensagem orientando o usuário a selecionar um paciente para ter acesso às demais funcionalidades. O usuário também pode optar por inserir um novo paciente, que após o cadastro, é exibido na lista de pacientes. A Figura 15 apresenta a tela inicial do aplicativo.

Figura 15 – Tela inicial do aplicativo



A classe `PatientListFragment` é responsável por apresentar na tela um componente `ListView` com os pacientes. Eles são carregados no método `onStart`, conforme é apresentado no Quadro 2.

Quadro 2 – Código do método `onStart` da classe `PatientListFragment`

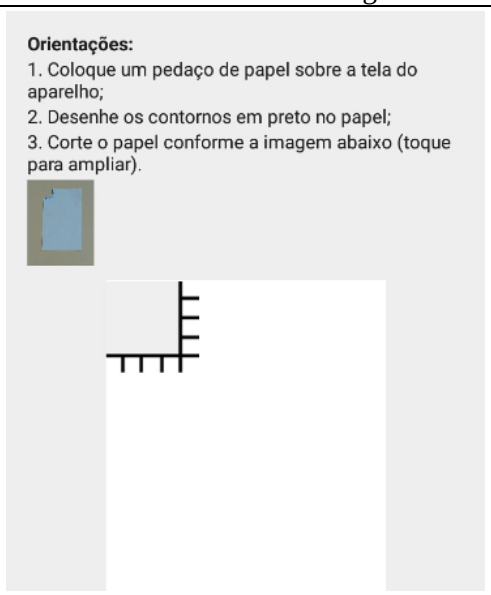
```
1     @Override
2     public void onStart() {
3         super.onStart();
4         List<Patient> patients = repository.getAll();
5         ArrayAdapter<Patient> adapter = new ArrayAdapter<Patient>(
6                 getActivity(),
7                 android.R.layout.simple_list_item_1,
8                 patients);
9         listPatients.setAdapter(adapter);
10    }
```

Ao selecionar um paciente, o método `onItemClick` da classe `PatientListFragment`, define o `id do paciente` selecionado na classe `CorpusMappingApp`, assim qualquer classe do aplicativo que precisar do `id do paciente` selecionado, poderá obtê-lo a partir da instância única da classe `CorpusMappingApp`.

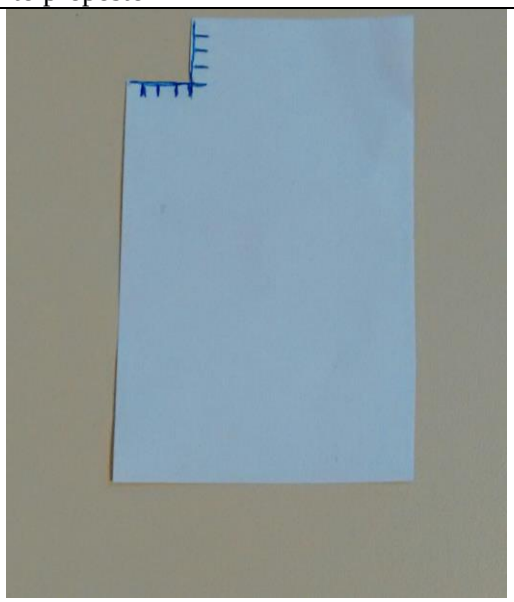
3.3.2.3 Aquisição do gabarito

Foi desenvolvida uma tela que orienta o usuário como criar o gabarito a ser utilizado na captura das imagens. A tela, exibida na Figura 16, é apresentada por meio da `TemplateActivity` e pode ser acessada através do menu lateral do aplicativo.

Figura 16 – Gabarito proposto



(a) Tela com orientações para criação do gabarito



(b) Exemplo de gabarito criado

Para criar o gabarito o usuário precisa colocar um pedaço de papel sobre a tela do celular. Em seguida deve desenhar os contornos em preto no papel e por fim, cortar o papel para que fique conforme a Figura 16.

3.3.2.4 Capturar imagem

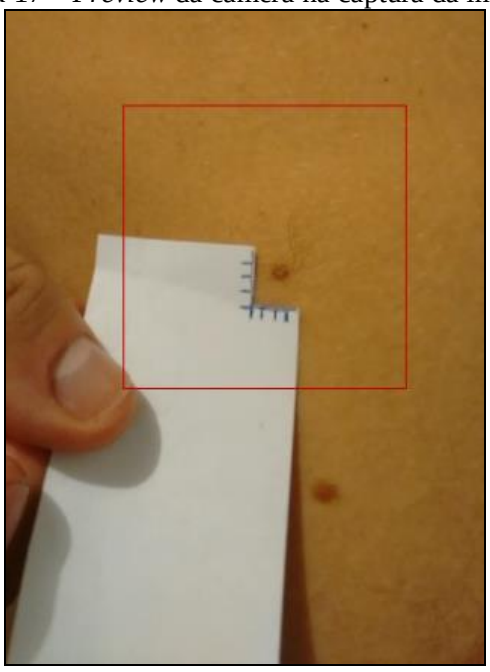
O processo de captura da imagem possui algumas etapas, sendo elas: a utilização da câmera do dispositivo, desenho da região de interesse no *preview* da câmera, a captura propriamente dita e o recorte da imagem para manter apenas a região de interesse. As próximas seções destinam-se ao detalhamento da implementação das etapas mencionadas.

3.3.2.4.1 Captura da imagem

Quando o usuário escolhe a opção de capturar uma nova imagem é aberta a câmera do dispositivo. O aplicativo customizou o *preview* da câmera, para apresentar o desenho da região de interesse onde devem estar a pinta e o gabarito.

As classes `CameraActivity` e `CameraPreview` são as responsáveis por abrir a câmera e desenhar o *preview*. O layout da `CameraActivity` é composto por um *container* que contém um `SurfaceView`, onde a câmera desenha o *preview*, e um quadrado vermelho onde deve estar a pinta e o gabarito. Na Figura 17 é possível ver o quadrado representando visualmente a região de interesse.

Figura 17 – *Preview* da câmera na captura da imagem



O Quadro 3 demonstra o código do método `onResume` da `CameraActivity`. Este método faz parte do ciclo de vida da *Activity* e é chamado sempre que a *Activity* é apresentada na tela.

Quadro 3 – Método `onResume` da `CameraActivity`

```
1  protected void onResume() {
2      super.onResume();
3      if (camera == null) {
4          camera = Camera.open();
5          camera.startPreview();
6          camera.setErrorCallback(new ErrorCallback() {
7             // CÓDIGO...
8          });
9      }
10     if (camera != null) {
11         if (Build.VERSION.SDK_INT >= 14)
12             setCameraDisplayOrientation(context,
13                     CameraInfo.CAMERA_FACING_BACK, camera);
14         cameraPreview.setCamera(camera);
15     }
16 }
```

O método `onResume` (Quadro 3) abre a câmera e inicia o *preview* (linhas 5 e 6). Posteriormente, na linha 14, a câmera é passada para o objeto `cameraPreview`, responsável por configurá-la. A classe `CameraPreview` adiciona um *callback* no `SurfaceHolder` da `SurfaceView` onde é desenhado o *preview* ao vivo. O Quadro 4 apresenta a implementação do método `surfaceCreated` da *callback*. Na linha 5, é chamado o método `setPreviewDisplay` da câmera, informando para a câmera onde deverá ser renderizada a visualização.

Quadro 4 – Método `surfaceCreated`

```
1  public void surfaceCreated(SurfaceHolder holder) {
2      // O surface foi criado, avisa a câmera onde desenhar o preview.
3      try {
4          if (camera != null) {
5              camera.setPreviewDisplay(holder);
6          }
7      } catch (IOException exception) {
8          Log.e(TAG, "IOException on setPreviewDisplay()", exception);
9      }
10 }
```

A imagem é capturada quando o usuário toca na tela, para tal, é adicionado um *listener* de *click* na `surfaceView`. No momento em que ocorre o toque na tela, o método `onClick` do *listener* é chamado, que por sua vez chama o método `takeFocusedPicture` (Quadro 5).

Quadro 5 – Código da captura da imagem

```
1  public void takeFocusedPicture() {
2      camera.autoFocus(autoFocusCallback);
3  }
4
5  Camera.AutoFocusCallback autoFocusCallback = new
6  Camera.AutoFocusCallback() {
7      @Override
8      public void onAutoFocus(boolean success, Camera camera) {
9          camera.takePicture(null, null, jpegCallback);
10     }
11 };
12
13 PictureCallback jpegCallback = new PictureCallback() {
14     public void onPictureTaken(byte[] data, Camera camera) {
15         //salva a imagem temporariamente
16         FileOutputStream outStream = null;
17         try {
18             outStream = new FileOutputStream(path);
19             outStream.write(data);
20         } catch (IOException e) {
21             e.printStackTrace();
22         } finally {
23             if (outStream != null) {
24                 try {
25                     outStream.close();
26                 } catch (IOException e) {}
27             }
28         }
29
30         Intent intent = new Intent(context, ViewPhotoActivity.class);
31         intent.putExtra(IMAGE_PATH, path);
32         startActivityForResult(intent, REQUEST_CODE_IMAGE);
33         surfaceView.setClickable(true);
34     }
35 };
```

O método `takeFocusedPicture` chama o método `autoFocus` da `camera` passando uma *callback* para capturar a imagem no momento em que a câmera estiver com o foco ajustado (linha 9). Na chamada do método `takePicture` é passado por parâmetro uma *callback* que será executada quando a imagem já foi processada e estiver disponível no formato JPEG. Neste momento, a imagem é salva temporariamente no cartão de memória do dispositivo (linhas 18 e 19). Na linha 32, é invocada abertura da `ViewPhotoActivity`, responsável por recortar a imagem para manter apenas a região de interesse e apresentar a imagem ao usuário para que ele decida se quer utilizá-la ou descartá-la.

3.3.2.4.2 Seleção da região de interesse

A imagem capturada deve ser recortada para manter apenas a região de interesse, que compreende toda a área do quadrado vermelho da Figura 17. O método `onCreate` da classe

`ViewPhotoActivity` lê a imagem que foi salva pela `jpegCallback` da classe `CameraActivity` e chama a rotina responsável por recortar a imagem. O Quadro 6 apresenta o código do método `onCreate`.

Quadro 6 – Método `onCreate` da classe `ViewPhotoActivity`

```
1  protected void onCreate(Bundle savedInstanceState) {
2      super.onCreate(savedInstanceState);
3      setContentView(R.layout.activity_view_photo);
4
5      path = getIntent().getStringExtra(IMAGE_PATH);
6      uri = Uri.fromFile(new File(path));
7
8      Bitmap originalImage = BitmapFactory.decodeFile(path, null);
9      originalImage = adjustRotation(originalImage);
10     croppedImage = ImageUtils.cropImage(this, originalImage);
11
12     ImageView imageView = (ImageView) findViewById(R.id.imageView);
13     imageView.setImageBitmap(croppedImage);
14     //CÓDIGO
15 }
```

Na linha 5, é obtido o caminho onde foi salva a imagem. A leitura do arquivo da imagem em um `Bitmap` é feita na linha 8. Na linha 10 é feita a chamada da rotinha que realiza o corte da imagem, método `cropImage` da classe `ImageUtils`, representado no Quadro 7.

Quadro 7 – Método `cropImage` da classe `ImageUtils`

```
1  public static Bitmap cropImage(Context context, Bitmap image) {
2      int centerY = image.getHeight() / 2;
3      int centerX = image.getWidth() / 2;
4      DisplayMetrics displayMetrics =
5  context.getResources().getDisplayMetrics();
6      float dpi = displayMetrics.densityDpi;
7      // quantidade de pixels por dp
8      float pixelForDp = dpi / 160f;
9
10     float roiSizeOnScreen = EXTERNAL_SQUARE_SIZE * pixelForDp;
11     int roiSizeOnImage = (int) (roiSizeOnScreen * image.getHeight()) /
12 displayMetrics.heightPixels;
13     int width = (int) (roiSizeOnImage + (roiSizeOnImage * 0.2));
14     int height = (int) (roiSizeOnImage + (roiSizeOnImage * 0.1));
15
16     int x = centerX - (width / 2);
17     int y = centerY - (height / 2);
18     Bitmap croppedImage = Bitmap.createBitmap(image, x, y, width,
19 height);
20     return croppedImage;
21 }
```

É necessário cortar a imagem de uma forma que fique apenas a região de interesse. Para isso, são obtidas as coordenadas xy do centro da imagem (linhas 2 e 3). O tamanho da região de interesse está em dp (*Density Independent Pixels*), um dp equivale a 1 pixel em uma tela de 160 dpi. Desta forma, na linha 10 é obtido o valor da densidade dpi da tela do

dispositivo, para na sequência, dividir esse valor por 160 e assim obter a quantidade de pixels por dp. Para descobrir o tamanho em pixels da região de interesse na tela, basta multiplicar o tamanho em dp pela quantidade de pixels por dp que foi calculada, conforme a linha 12. Porém, para saber qual o tamanho da imagem cortada, é necessário obter o tamanho da região de interesse na imagem, esse cálculo é feito nas linhas 13 e 14 por meio da regra de três.

Na linha 18 é chamado método `createBitmap` informando a imagem original, os valores x e y de onde deve-se iniciar o corte e a largura e altura esperados. Após o corte da imagem, a mesma é apresentada ao usuário, que tem a possibilidade de confirmar o seu uso ou descarta-la. Ao escolher descartar, o aplicativo volta para a câmera, ao escolher salvar segue para a etapa de vinculação.

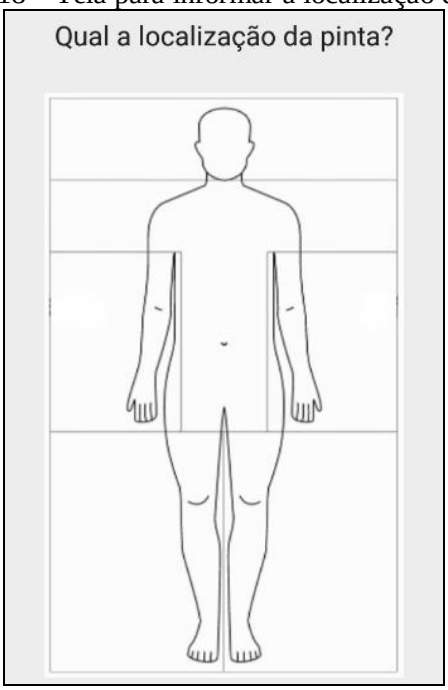
3.3.2.5 Vincular imagem a região corporal

Para a vinculação da imagem a uma região do corpo, é necessário primeiramente que o usuário selecione a parte do corpo que deseja vincular, depois ele deve selecionar o local específico da pinta. As seções seguintes descrevem os detalhes destas etapas.

3.3.2.5.1 Selecionar parte do corpo

A classe responsável por apresentar a tela de seleção da parte do corpo ao usuário é a `SaveImageActivity`. A Figura 18 demonstra a tela que é exibida ao usuário.

Figura 18 – Tela para informar a localização da pinta



No método `onCreate` da *activity* é adicionado um *listener* de toque no componente `ImageView` que apresenta a imagem do corpo. O *listener* usado é o `BodyTouchListener`, que

estende a classe `ImageBoundingBoxTouchListener`, responsável por verificar se o toque ocorreu dentro de alguma *bounding box*. O `BodyTouchListener` passa via parâmetro uma *bounding box* de cada região corporal para a `ImageBoundingBoxTouchListener`, que verifica se o toque ocorreu dentro de uma *bounding box* e neste caso chama o método `onClickInnerBoundingBox`, implementado na classe `BodyTouchListener` e demonstrado no Quadro 8.

Quadro 8 – Método `onClickInnerBoundingBox`

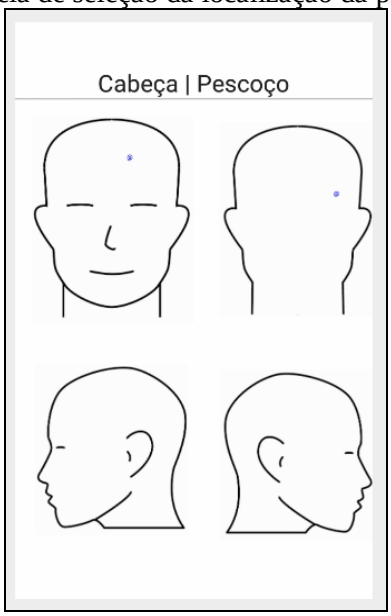
```
1  public void onClickInnerBoundingBox(ImageView view, PointF touchPoint,
2  int boundingBoxId) {
3      BodyPart bodyPart = null;
4
5      if (boundingBoxId == headBBox.id) {
6          bodyPart = BodyPart.HEAD;
7      } else if (boundingBoxId == shoulderBBox.id || boundingBoxId ==
8  bodyBBox.id) {
9          bodyPart = BodyPart.BODY;
10     } else if (boundingBoxId == leftArmBBox.id) {
11         bodyPart = BodyPart.LEFT_ARM;
12     } else if (boundingBoxId == rightArmBBox.id) {
13         bodyPart = BodyPart.RIGHT_ARM;
14     } else if (boundingBoxId == leftLegBBox.id) {
15         bodyPart = BodyPart.LEFT_LEG;
16     } else if (boundingBoxId == rightLegBBox.id) {
17         bodyPart = BodyPart.RIGHT_LEG;
18     }
19
20     if (bodyPart != null) {
21         Intent i = new Intent(activity, SelectBodyPartActivity.class);
22         i.putExtra(SelectBodyPartActivity.PARAM_BODY_PART,
23 bodyPart.name());
24         activity.startActivityForResult(i,
25 SelectBodyPartActivity.REQUEST_CODE);
26     }
27 }
```

No método `onClickInnerBoundingBox`, é verificado qual *bounding box* foi selecionada pelo usuário (linhas 5 à 18). Na linha 24 é chamada a abertura da `SelectBodyPartActivity`, em que o usuário irá selecionar o local específico da pinta.

3.3.2.5.2 Selecionar local específico da pinta

Após o usuário selecionar a parte do corpo com a qual deseja vincular a imagem, é necessário selecionar o local específico da pinta. Para isso, a `SelectBodyPartActivity` apresenta as variações da parte do corpo selecionada pelo usuário na etapa anterior. Por exemplo, se o usuário selecionou a cabeça, é apresentada uma imagem com a cabeça de frente, outra de costas, uma com a face direita e outra com a face esquerda conforme pode ser visto na Figura 19.

Figura 19 – Tela de seleção da localização da pinta na cabeça



Como é possível perceber na Figura 19, o aplicativo lê os grupos de pintas existentes e os desenha na imagem da região corporal. Para cada parte do corpo da Figura 18 existe uma classe `Fragment`, para a cabeça, por exemplo, existe a classe `SelectionHeadFragment`, para o braço direito, a `SelectionArmRightFragment`. Essas classes são responsáveis por apresentar na tela as variações da região corporal e desenhar as pintas já vinculadas em cada região. Para desenhar as pintas, é utilizado o método `drawPointsOfBodyPart` da classe utilitária `ImageDrawer`, demonstrado no Quadro 9.

Quadro 9 – Método `drawPointsOfBodyPart` da classe `ImageDrawer`

```
1  public static void drawPointsOfBodyPart(ImageView view,
2  SpecificBodyPart bodyPart, int resourceId) {
3      Bitmap bitmap = BitmapFactory.decodeResource(view.getResources(),
4  resourceId);
5      CorpusMappingApp app = CorpusMappingApp.getInstance();
6      long patientId = app.getSelectedPatientId();
7      ImageRecordRepository repository = getImageRecordRepository(view);
8      List<ImageRecord> imageRecords =
9  repository.getByBodyPartId(patientId, bodyPart);
10
11     List<PointF> listPoints = new ArrayList<>();
12     for (ImageRecord i : imageRecords) {
13         PointF position = i.getMoleGroup().getPosition();
14         if (!listPoints.contains(position)) {
15             listPoints.add(position);
16         }
17     }
18     if (listPoints.size() > 0) {
19         PointF[] points = listPoints.toArray(new
20 PointF[listPoints.size()]);
21         bitmap = ImageDrawer.drawPoint(bitmap, points);
22     }
23     view.setImageBitmap(bitmap);
24 }
```

O método recebe como parâmetro o componente `ImageView` onde será apresentada a imagem da região corporal, qual é a região corporal que ele vai trabalhar e o id do recurso que representa a imagem dentro do aplicativo. Nas linhas 8 e 9, são obtidos os registros de imagens da região corporal informada via parâmetro. Em seguida (linhas 12 a 17), são buscados os pontos (x, y) de cada pinta. Na linha 21, é chamado o método `drawPoint`, que irá desenhar cada um dos pontos na imagem da região corporal. O Quadro 10 apresenta a implementação do método `drawPoint`.

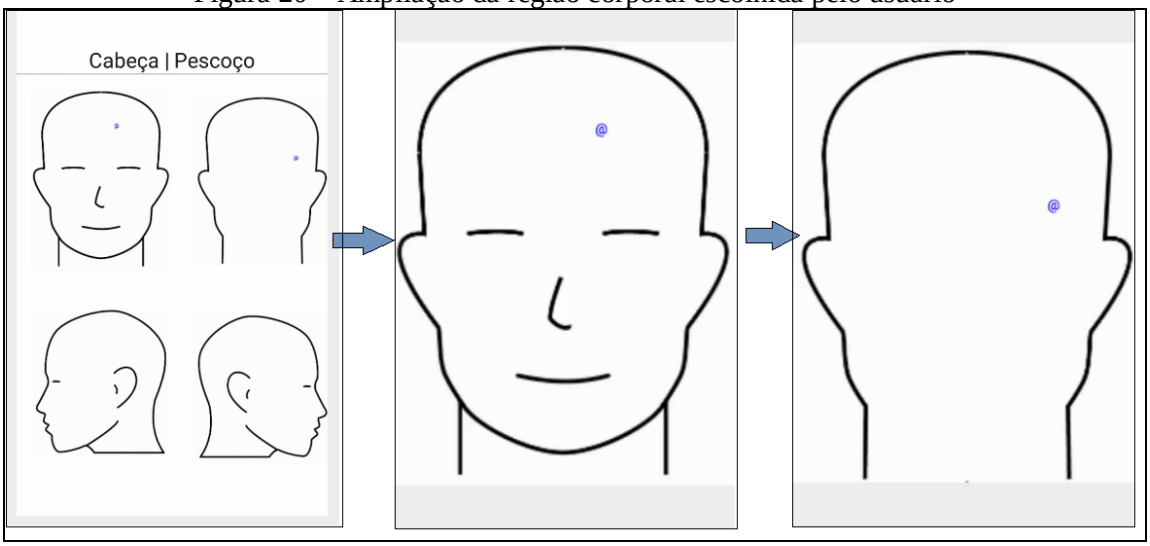
Quadro 10 – Método `drawPoint` da classe `ImageDrawer`

```
 1  public static Bitmap drawPoint(Bitmap bitmap, PointF... point) {
 2      Bitmap mutableBitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);
 3
 4      Canvas canvas = new Canvas(mutableBitmap);
 5
 6      Paint paintText = new Paint(Paint.ANTI_ALIAS_FLAG);
 7      paintText.setColor(Color.BLUE);
 8      paintText.setTextSize(12);
 9      paintText.setStyle(Paint.Style.FILL);
10      paintText.setFakeBoldText(true);
11
12      for (PointF p : point) {
13          canvas.drawText("@", p.x, p.y, paintText);
14      }
15      return mutableBitmap;
16  }
```

O método `drawPoint` recebe via parâmetro o `bitmap` (ou imagem) onde devem ser desenhados os pontos e um *array* contendo os pontos. Na segunda linha, é criada uma cópia do `bitmap` recebido por parâmetro, é neste `bitmap` onde os pontos serão desenhados. Em sequência, nas linhas 12 a 14, é iterado pelos pontos, desenhando cada um deles. O retorno do método é o `bitmap` com os pontos desenhados. O resultado desse processo pode ser visto na Figura 19.

Quando o usuário seleciona uma das regiões corporais, a mesma é ampliada, através da `BodyImageSliderActivity`. É possível navegar em cada uma das partes ampliadas, através do gesto de toque da direita para a esquerda. O resultado da ampliação pode ser visto na Figura 20.

Figura 20 – Ampliação da região corporal escolhida pelo usuário



A partir do toque do usuário na tela da parte do corpo ampliada, o aplicativo identifica a posição do toque e verifica nas imagens existentes se já existe algum grupo para aquela posição. De acordo MIT Touch Lab, a largura média do dedo indicador é 16-20 mm para adultos, o que equivale a uma gama entre 45-57 pixels. No aplicativo proposto, utilizou-se o tamanho de 50 pixels. Desta forma, a cada toque do usuário, será analisado se já existe uma imagem em uma posição 25 pixels em cada direção a partir do centro do toque, totalizando os 50 pixels esperados.

O método `onClickInnerBoundingBox` da classe `AssociateBodyPartTouchListener` é executado quando o aplicativo identifica que o toque ocorreu dentro da região corporal. Como é possível ver no Quadro 11, na linha 7 é chamado o método `getMoleGroup`, que verifica se já existe um grupo na posição do local escolhido pelo usuário, se existir retorna o grupo, caso contrário retorna `null`.

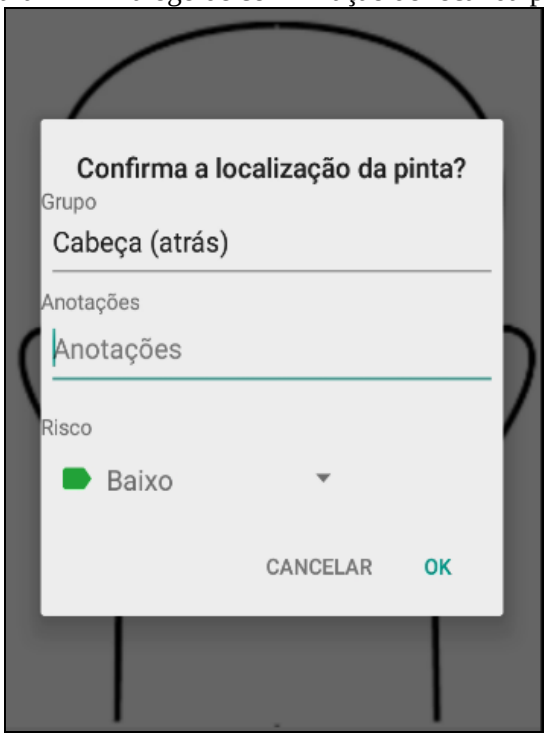
Quadro 11 – Confirmação da localização da pinta

```
1  public void onClickInnerBoundingBox(final ImageView view,
2   final PointF touchPoint, int boundingBoxId) {
3      if (bitmap == null) {
4          bitmap = BitmapFactory.decodeResource(view.getResources(),
5  resourceId);
6      }
7      MoleGroup moleGroup = getMoleGroup(touchPoint);
8
9      if (moleGroup == null) {
10         Bitmap newBitmap = ImageDrawer.drawPoint(bitmap, touchPoint);
11         view.setImageBitmap(newBitmap);
12         view.setAdjustViewBounds(false);
13     }
14
15     AlertDialog.Builder builder = new AlertDialog.Builder(activity);
16     builder.setTitle("Confirma a localização da pinta?");
17
```

```
18      // CÓDIGO ...
19
20      if (moleGroup != null) {
21          edtGroupName.setText(moleGroup.getGroupName());
22          edtAnnotation.setText(moleGroup.getAnnotations());
23          int classification = moleGroup.getClassification().ordinal();
24          spinnerClassification.setSelection(classification);
25      } else {
26          //sugere um nome para o grupo
27          edtGroupName.setText(bodyPart.getBodyPartName());
28
29  spinnerClassification.setSelection(MoleClassification.NONE.ordinal());
30      }
31
32      // CÓDIGO ...
33
34      AlertDialog dialog = builder.create();
35      dialog.show();
36  }
```

Na sequência, é aberto um diálogo (linhas 15 a 36) para o usuário confirmar a localização da pinta. Neste momento, é possível indicar o risco da pinta (baixo, médio ou alto), informar um nome para o grupo e inserir anotações a respeito da pinta. A Figura 21 apresenta como é o diálogo de confirmação.

Figura 21 – Diálogo de confirmação do local da pinta



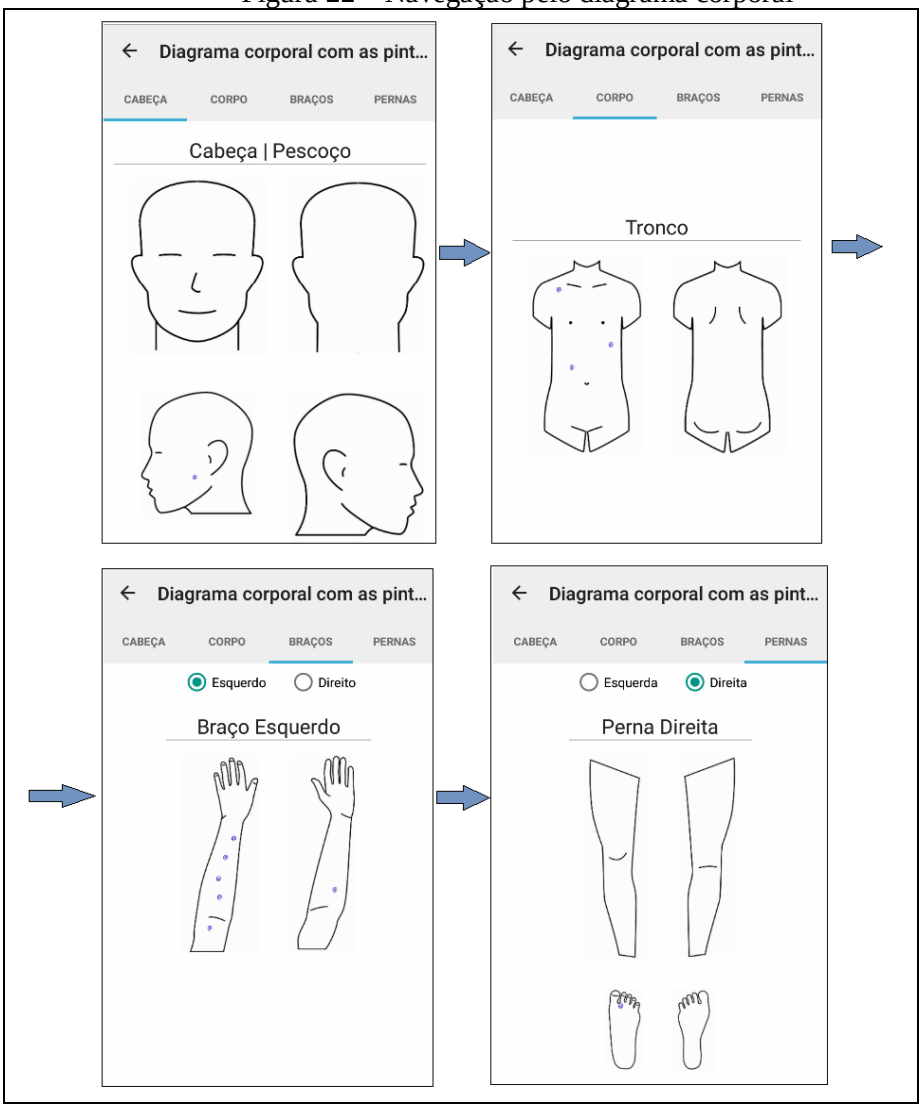
Quando for confirmada a localização, a imagem da pinta será salva na pasta do aplicativo no cartão de memória do dispositivo, em um diretório exclusivo do paciente. Além disso, será inserido um registro na tabela `IMAGE_RECORD`, responsável por armazenar as informações de uma imagem. As informações salvas são: id do paciente, data e hora que a

imagem foi capturada, anotações, caminho do arquivo da imagem no cartão de memória, a parte do corpo e o id do grupo. Se a imagem for de um novo grupo, também será inserido um registro na tabela `MOLE_GROUP`, que armazena as informações a respeito de um grupo de imagens de uma única pinta. Podemos dizer que um `MoleGroup` é um agrupador de `ImageRecord`. No `MoleGroup` fica armazenado o id do paciente, o risco da pinta (baixo, médio ou alto), o nome do grupo e a posição (x, y) da pinta.

3.3.2.6 Visualizar imagens

O aplicativo permite que o usuário consulte as imagens através da navegação pelo diagrama corporal, a *activity* responsável pela navegação é a `ViewBodyDiagramActivity`. Conforme apresentado na Figura 22, o usuário navega pelas partes do corpo através de gestos na tela do dispositivo, as partes do corpo apresentam um desenho no local onde já foi mapeado alguma pinta.

Figura 22 – Navegação pelo diagrama corporal



A classe `ViewBodyDiagramActivity` cria um *fragment* para cada região corporal, `ViewHeadMolesFragment` para a cabeça, `ViewArmMolesFragment` para os braços e mãos, `ViewBodyMolesFragment` para o tronco e, por fim, `ViewLegMolesFragment` para as pernas e pés. Cada *fragment* é responsável por exibir o desenho da parte do corpo correspondente e marcar as pintas já mapeadas para essa parte do corpo, para isso é utilizado o método `drawPointsOfBodyPart` da classe `ImageDrawer`, já demonstrado no Quadro 9.

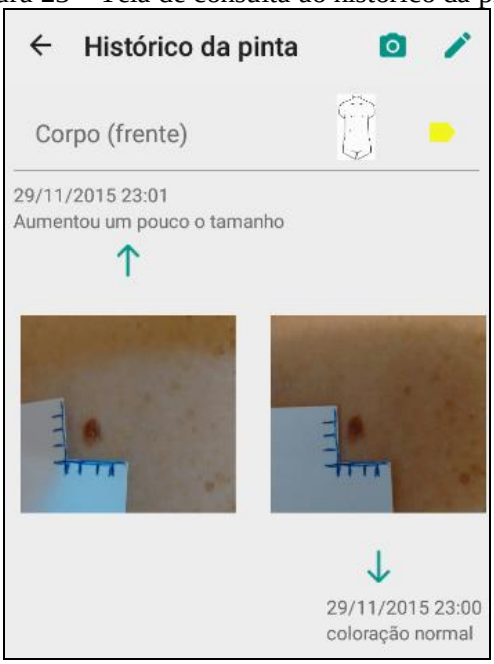
Para consultar o histórico de alguma pinta, o usuário deve clicar sobre a parte do corpo desejada, o aplicativo irá ampliar a parte do corpo selecionada, em seguida, o usuário deve tocar sobre a pinta que deseja consultar. É adicionado na imagem ampliada da parte do corpo o `ViewMolesTouchListener` que é avisado quando ocorre o toque na tela e nesse momento verifica se existe alguma pinta associada ao ponto onde ocorreu o toque. O Quadro 12 apresenta o código que é executado pelo *listener*.

Quadro 12 – Código executado quando ocorre toque sobre a parte do corpo

```
1  CorpusMappingApp app = CorpusMappingApp.getInstance();
2  long patientId = app.getSelectedPatientId();
3  ImageRecordRepository repository =
4                  ImageRecordRepository.getInstance(activity);
5
6  List<ImageRecord> imageRecords =
7              repository.getByBodyPartAndPosition(patientId, bodyPart,
8  touchPoint);
9
10 if (!imageRecords.isEmpty()) {
11     Intent i = new Intent(activity, MoleImageSliderActivity.class);
12     i.putExtra(PARAM_IMAGES,
13             imageRecords.toArray(new
14 mageRecord[imageRecords.size()]));
15     activity.startActivity(i);
16 }
```

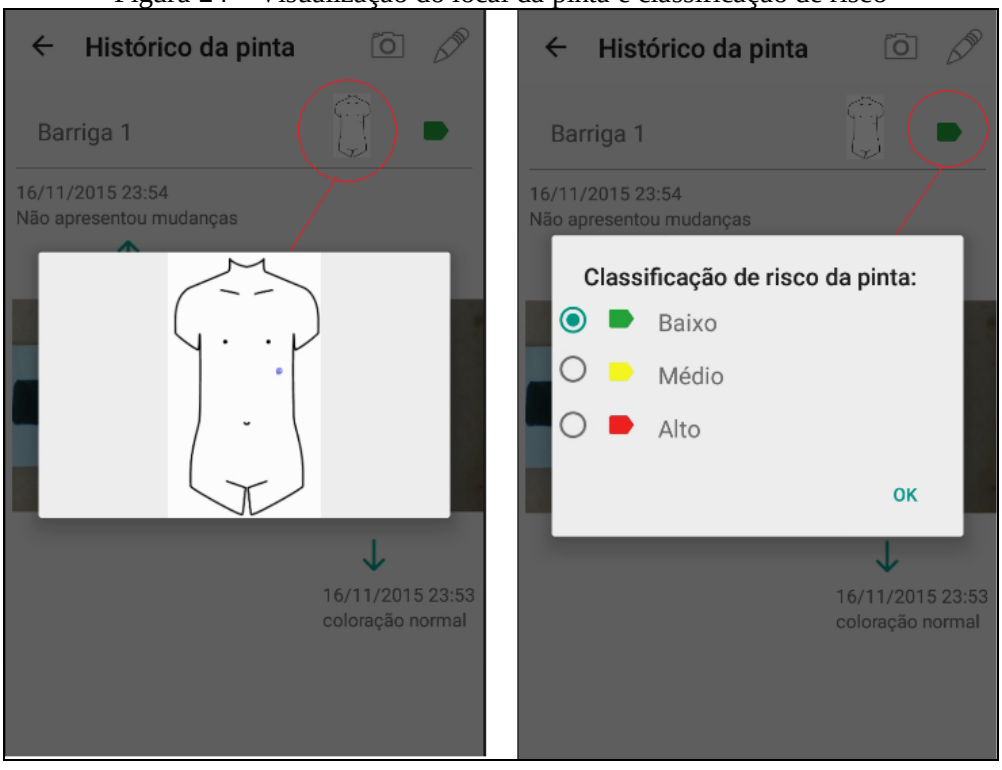
Na linha 7 é chamado o método `getByBodyPartAndPosition` da classe `ImageRecordRepository`, que retorna os registros de imagens associados à parte do corpo e posição selecionada pelo usuário. Se for encontrado algum registro, nas linhas 11 a 15 é solicitada a abertura da `MoleImageSliderActivity`, que é responsável por apresentar as imagens da pinta selecionada. A Figura 23 exibe a tela que é aberta para a consulta do histórico da pinta.

Figura 23 – Tela de consulta ao histórico da pinta



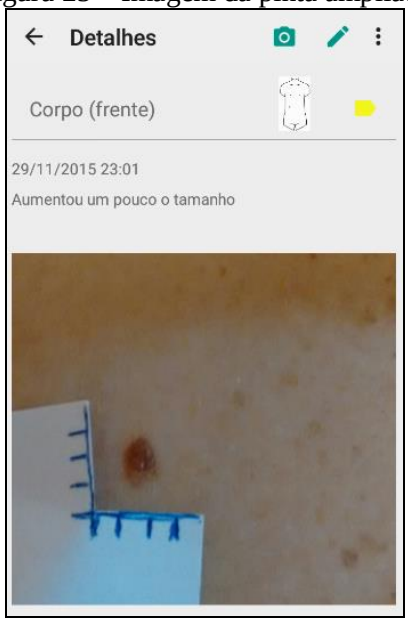
A `MoleImageSliderActivity` cria um componente `ViewPager` para exibir as imagens uma ao lado da outra, através de gestos na tela, é possível navegar pelas próximas imagens. Como é possível ver na Figura 23, é apresentada a data e hora que as imagens foram capturadas, bem como as anotações inseridas. Além disso, na parte superior é apresentado o nome do grupo, sua localização e classificação de risco. Conforme apresentado na Figura 24, é possível visualizar o local da pinta e alterar a classificação de risco da mesma.

Figura 24 – Visualização do local da pinta e classificação de risco



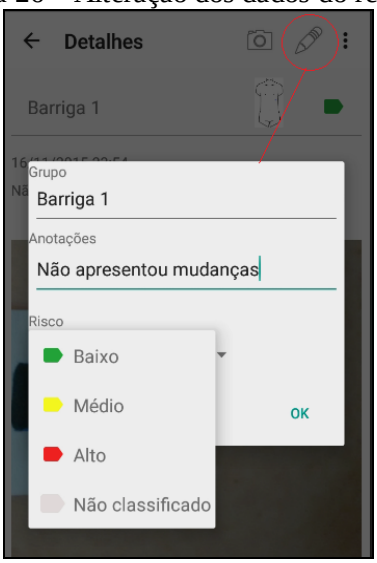
Para visualizar uma imagem de forma ampliada, basta clicar sobre a imagem desejada. Após o click, é chamado a abertura da `MoleDetailedImageSliderActivity`, que apresenta a imagem que foi solicitada. Mesmo apresentando apenas uma imagem por vez, é possível navegar pelas outras imagens. A Figura 25 apresenta a tela gerada pela `MoleDetailedImageSliderActivity` que exibe a imagem de forma ampliada. Nesta tela também é possível consultar a data e hora da captura da imagem, o nome do grupo e sua classificação.

Figura 25 – Imagem da pinta ampliada



Existe a opção de capturar uma nova imagem para este grupo, basta clicar sobre o ícone de câmera no canto superior direito. Também é possível alterar os dados da imagem, como anotações e classificação de risco, para isto é necessário clicar sobre o ícone de lápis no canto superior direito (Figura 26).

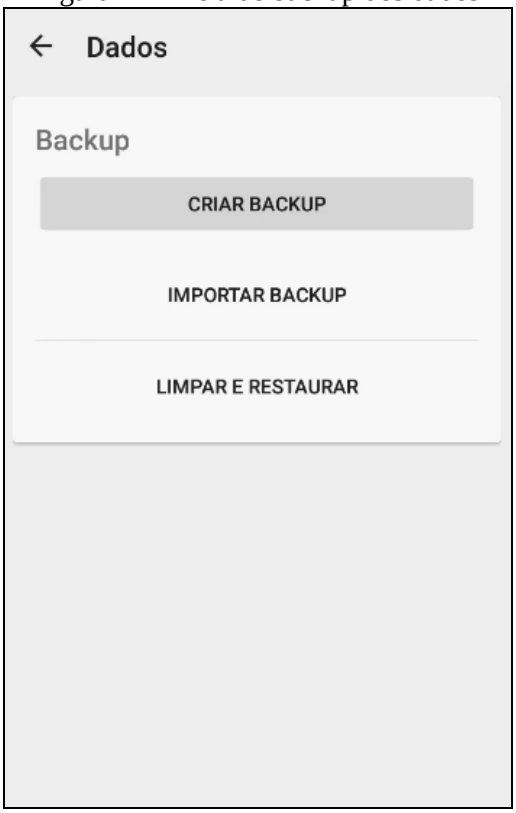
Figura 26 – Alteração dos dados do registro



3.3.2.7 Backup

O aplicativo permite que o usuário faça backup dos dados da aplicação, exportando-os para uma plataforma na nuvem e, posteriormente, realize a importação os dados que foram exportados. A plataforma escolhida para armazenar o backup foi o Google Drive. Para isso, foi utilizada a API disponibilizada pelo Android para acessar serviços do Google. A Figura 27 apresenta a tela que é exibida quando a opção de backup é acessada. A classe responsável por essa tela é a `DataActivity` em conjunto com a `DataFragment`.

Figura 27 – Tela de backup dos dados



Conforme demonstrado na Figura 27, o aplicativo apresenta as opções de criar backup, importar um backup que foi exportado anteriormente e limpar todos os dados do aplicativo e restaurá-lo com um backup. As próximas seções destinam-se ao detalhamento da implementação dessas opções.

3.3.2.7.1 Criar backup

Quando a opção de criar backup é acionada, o método `onClick` da classe `DataFragment` chama a abertura da `ExportActivity`. A classe `ExportActivity` inicia o processo adicionando o `GoogleApiFragment`, responsável por gerenciar a conexão com o Drive. No método `onCreate` da classe `GoogleApiFragment`, é realizada a chamada do método `buildGoogleApiClient`, que obtém uma instância da classe `GoogleApiClient`,

disponibilizada pelo Android e é o ponto de entrada para integração com os serviços do Google. O Quadro 13 apresenta o código do método `buildGoogleApiClient`.

Quadro 13 – Método que obtém uma instância da classe `GoogleApiClient`

```
1  private void buildGoogleApiClient() {
2      GoogleApiClient.Builder builder = new
3  GoogleApiClient.Builder(getActivity(), this, this);
4      builder.addApi(Drive.API).addScope(Drive.SCOPE_FILE);
5      client = builder.build();
6  }
```

Nas linhas 2 e 3, é instanciado um `builder` responsável por configurar o `GoogleApiClient`. Nesse momento, é informado como parâmetro do `builder`, o próprio *fragment* que implementa as interfaces `ConnectionCallbacks` e `OnConnectionFailedListener`. A interface `ConnectionCallbacks` possui os métodos `onConnected` e `onConnectionSuspended`, que são chamados quando o *client* conecta ou desconecta dos serviços. Na execução do método `onConnected`, é disparado um evento avisando que a conexão foi realizada, a classe `ExportActivity` recebe esse evento no método `onGoogleApiClientConnected` (Quadro 14) e solicita a abertura da *activity* do Drive que permite escolher a pasta onde será salvo o backup.

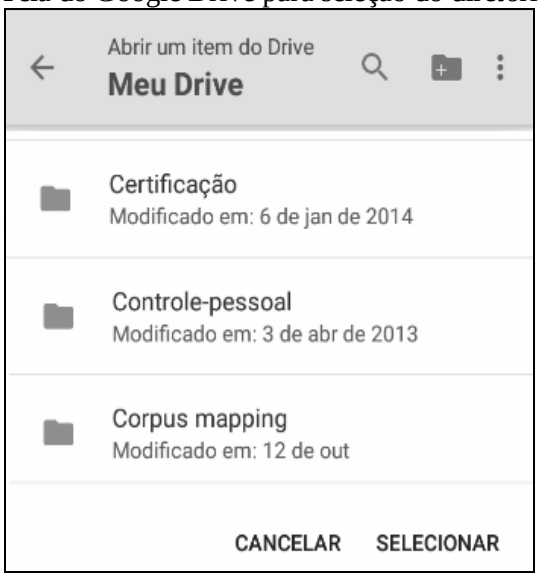
Quadro 14 – Abertura da *activity* para seleção da pasta onde será salvo o backup

```
 1  public void onGoogleApiClientConnected(GoogleApiConnection connection)
 2  {
 3      if (isDirectoryRequested ||
 4  !connection.contains(UNIQUE_GOOGLE_API_ID)) {
 5          return;
 6      }
 7
 8      googleApiClient = connection.get(UNIQUE_GOOGLE_API_ID);
 9      IntentSender intentSender = Drive.DriveApi
10              .newOpenFileActivityBuilder()
11              .setMimeType(new String[]{"application/vnd.google-
12  apps.folder"})
13              .build(googleApiClient);
14
15      try {
16          startIntentSenderForResult(intentSender,
17  REQUEST_DRIVE_DIRECTORY, null, 0, 0, 0);
18          isDirectoryRequested = true;
19      } catch (IntentSender.SendIntentException e) {
20          throw new ExportError("Unable to show Google Drive.", e);
21      }
22  }
```

Nas linhas 9 a 13 do Quadro 14, é criado uma `intentSender` através da API do Drive. A `intentSender` é utilizada na linha 16 para mandar abrir a *activity* do Drive que permite a seleção de um arquivo. Na linha 11 é informado o *MimeType* do tipo *folder*, desta forma, o

usuário só poderá escolher arquivos do tipo pasta. A Figura 28 apresenta a tela do Google Drive que é aberta após a execução do código do Quadro 14.

Figura 28 – Tela do Google Drive para seleção do diretório do backup



Após o usuário selecionar a pasta para o backup, é executado o método `onActivityResult` da classe `ExportActivity`. Nesse momento, é obtido o *id* da pasta selecionada, que vem em um objeto do tipo `DriveId`. Com o id do diretório, é realizado a chamada da execução da classe `DriveDataExporterRunnable`, que ocorre em uma *thread* separada, para não travar a interface do aplicativo. O Quadro 15 apresenta o código do método `run` da classe `DriveDataExporterRunnable`.

Quadro 15 – Método `run` da classe `DriveDataExporterRunnable`

```
1  public void run() {
2  DriveFolder driveFolder = Drive.DriveApi.getFolder(googleApiClient,
3  driveId);
4  /* Cria um folder onde será criado o arquivo json com os dados
5   * e um arquivo zip com as imagens */
6  MetadataChangeSet.Builder builder = new MetadataChangeSet.Builder();
7  MetadataChangeSet changeSet = builder.setTitle(fileTitle).build();
8
9  PendingResult<DriveFolder.DriveFolderResult> pendingResult =
10         driveFolder.createFolder(googleApiClient, changeSet);
11
12 DriveFolder.DriveFolderResult driveFolderResult =
13 pendingResult.await();
14
15 DriveFolder backupDriveFolder = driveFolderResult.getDriveFolder();
16
17 exportJson(backupDriveFolder);
18 exportImages(backupDriveFolder);
19 }
```

No início do método, é obtido o `driveFolder` a partir do *id* do diretório selecionado pelo usuário. Na linha 12 é solicitada a criação de uma pasta dentro do diretório selecionado,

nessa pasta será criado um arquivo JSON com os dados provenientes de tabelas do SQLite e um arquivo zip com as imagens compactadas. Na linha número 19 é chamado o método `exportJson`, demonstrado no Quadro 16.

Quadro 16 – Método `exportJson` da classe `DriveDataExporterRunnable`

```
1  private void exportJson(DriveFolder backupDriveFolder) {
2      DriveApi.DriveContentsResult result =
3  Drive.DriveApi.newDriveContents(googleApiClient).await();
4
5      if (!result.getStatus().isSuccess()) {
6          throw new ExportError("Data export has failed.");
7      }
8
9      DriveContents contents = result.getDriveContents();
10     boolean isJson = true;
11     BackupDataExporter exporter = new
12 BackupDataExporter(contents.getOutputStream(), context, isJson);
13     try {
14         exporter.exportData();
15         eventBus.post(exporter);
16     } catch (Exception e) {
17         e.printStackTrace();
18         ExportError error = new ExportError("Data export has failed. "
19 + e.getMessage(), e);
20         eventBus.post(error);
21     }
22
23     MetadataChangeSet.Builder builder = new
24 MetadataChangeSet.Builder();
25     builder.setTitle(fileTitle + ".json");
26     builder.setMimeType("application/json");
27     MetadataChangeSet changeSet = builder.build();
28     backupDriveFolder.createFile(googleApiClient, changeSet,
29 contents).await();
30 }
```

Nas linhas 2 a 9 é obtido um objeto do tipo `DriveContents`, que fornece um `OutputStream` onde será escrito o conteúdo do arquivo com os dados em JSON. Logo após é chamado o método `exportData` da classe `BackupDataExporter`, que é responsável por exportar os dados para JSON no `OutputStream` fornecido pelo `DriveContents`. Na linha 28 (Quadro 16) é realizado a chamada do método `createFile` do `DriveFolder`. É nesse momento que é realizado *upload* dos dados para o Drive.

No Quadro 17 é possível ver o código implementado para exportar os dados no formato JSON. É criado um `JsonObject` onde são adicionados os pacientes cadastrados, os grupos de pintas e os dados das imagens vinculadas. Após adicionar os dados no `JsonObject`, o mesmo é escrito no `OutputStream` recebido por parâmetro. Um exemplo do JSON gerado pode ser visto no Quadro 37 do Apêndice C.

Quadro 17 – Exportação dos dados para json

```
1  public void exportData(OutputStream outputStream) throws Exception {
2      if (json) {
3
4          GsonBuilder builder = new GsonBuilder();
5          // CÓDIGO...
6
7          Gson gson = builder.create();
8          JsonObject root = new JsonObject();
9
10         addPatients(gson, root);
11         addMoleGroups(gson, root);
12         addImageRecords(gson, root);
13
14         outputStream.write(root.toString().getBytes(CHARSET_NAME));
15         outputStream.flush();
16         outputStream.close();
17     } else {
18         // CÓDIGO...
19     }
20 }
```

Além de exportar os dados para JSON, a classe `BackupDataExporter` também compacta as imagens um arquivo zip. O Quadro 18 apresenta o código responsável por realizar essa atividade.

Quadro 18 – Código para compactar as imagens

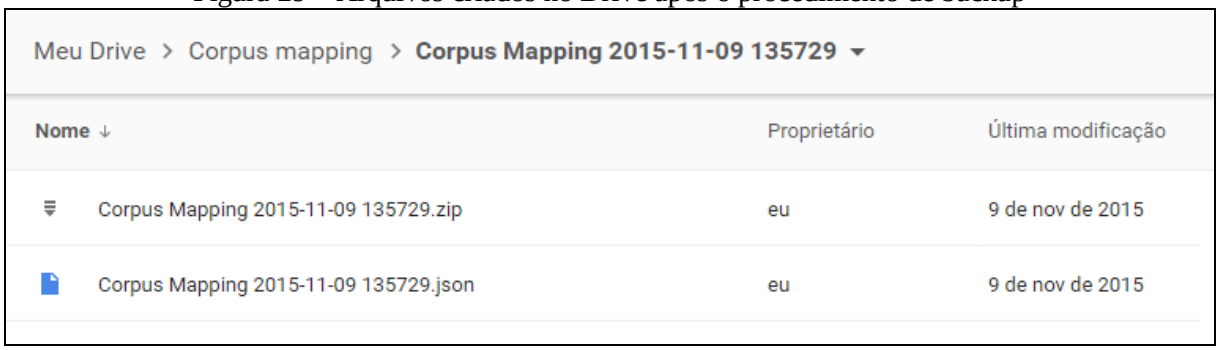
```
1  public void exportData(OutputStream outputStream) throws Exception {
2      if (json) {
3          // CÓDIGO...
4      } else {
5          ImageRecordRepository repository =
6  ImageRecordRepository.getInstance(context);
7          List<ImageRecord> imageRecords = repository.getAll();
8          // obtem as imagens
9          List<String> images = new ArrayList<>();
10
11         for (ImageRecord record : imageRecords) {
12             if (record.getImagePath() != null) {
13                 images.add(record.getImagePath());
14             }
15         }
16         // manda compactar as imagens em um arquivo zip
17         ZipManager.zip(context, images, outputStream);
18     }
19 }
```

Na linha número 7 é obtida uma lista com todos os registros de imagens vinculadas. Nas linhas 11 a 14 é iterado por cada registro para obter o caminho onde a imagem foi salva no cartão de memória. Após esse procedimento, é chamado o método `zip` da classe `ZipManager` para que cada imagem seja adicionada como uma entrada do arquivo zip.

O resultado da execução de todo esse procedimento de exportação dos dados pode ser visto na Figura 29. Onde é possível perceber a criação de uma pasta no Google Drive, com o

nome Corpus Mapping acrescido da data e hora de realização do backup. Dentro dessa pasta, existe um arquivo `.zip` com as imagens compactadas e um arquivo `.json` com os dados do aplicativo.

Figura 29 – Arquivos criados no Drive após o procedimento de backup



3.3.2.7.2 Importar backup

Conforme apresentado na Figura 27, é possível escolher entre importar um backup ou limpar e restaurar a aplicação. A ação de limpar e restaurar irá excluir todos os dados da base de dados e importar o que estiver no arquivo de backup. Nesta opção o usuário pode perder dados que tenham sido inseridos ou alterados após a criação do backup. A ação de importar backup irá realizar um *merge* com os dados atuais do aplicativo, ou seja, dados inseridos ou alterados após a criação do backup não serão perdidos.

Quando uma destas opções é acionada, o método `onClick` da classe `DataFragment` chama a abertura da `ImportActivity`. A classe `ImportActivity` inicia o processo adicionando o `GoogleApiFragment`, que gerencia a conexão com o Drive. Após a conexão com o Drive ter sido estabelecida, é necessário que o usuário selecione a pasta de backup que ele deseja importar. O processo de conexão com o Drive e seleção da pasta é semelhante ao que acontece com a `ExportActivity`, descrito na seção Criar backup.

Após o usuário selecionar a pasta onde está o backup, é executado o método `onActivityResult` da classe `ImportActivity`. Nesse momento, é obtido o `DriveId` da pasta selecionada. Em seguida, é realizado a chamada da execução da classe `DriveDataImporterRunnable`, que ocorre em uma *thread* separada, para não travar a interface do aplicativo. O Quadro 19 apresenta o código do método `run` da classe `DriveDataImporterRunnable`.

Quadro 19 – Método run da classe DriveDataImporterRunnable

```
1  DriveFolder folder = Drive.DriveApi.getFolder(googleApiClient,
2  driveId);
3
4  PendingResult<DriveApi.MetadataBufferResult> pendingResult =
5  folder.listChildren(googleApiClient);
6  DriveApi.MetadataBufferResult result = pendingResult.await();
7  MetadataBuffer metadataBuffer = result.getMetadataBuffer();
8  if (metadataBuffer.getCount() == 2) {
9
10     try {
11         Metadata jsonMetadata;
12         Metadata zipMetadata;
13         String firstMimeType = metadataBuffer.get(0).getMimeType();
14         if (firstMimeType.equals("application/json")) {
15             jsonMetadata = metadataBuffer.get(0);
16             zipMetadata = metadataBuffer.get(1);
17         } else {
18             zipMetadata = metadataBuffer.get(0);
19             jsonMetadata = metadataBuffer.get(1);
20         }
21         importJson(jsonMetadata);
22         importZip(zipMetadata);
23     } catch (Exception e) {
24         Log.e(TAG, e.getMessage(), e);
25         if (e instanceof ImportError) {
26             this.eventBus.post(e);
27         } else {
28             this.eventBus.post(new ImportError("Ocorreu um erro
29 durante a importação dos dados. Tente novamente.", e));
30         }
31     }
32 } else {
33     this.eventBus.post(new ImportError("A pasta escolhida é inválida
34 para importação dos dados"));
35 }
```

Na linha número 5 é solicitado a listagem dos arquivos filhos da pasta escolhida pelo usuário. É esperado que a pasta de backup possua dois arquivos (linha 8), um arquivo JSON e um arquivo ZIP. Nas linhas 14 a 20 é obtido o `Metadata` dos arquivos para, nas linhas 21 e 22 chamar os métodos responsáveis pela importação de cada um.

O Quadro 20 apresenta o código do método `importJson`, que a partir do `Metadata`, manda o Drive abrir o arquivo (linha 5). Após abrir o arquivo, é obtido o conteúdo do mesmo (linha 13). Na linha 17, o conteúdo é passado para a classe `BackupDataImporter` na forma de um `InputStream`. O método `importData` da classe `BackupDataImporter` é quem efetivamente importa os dados para as tabelas do aplicativo.

Quadro 20 – Método `importJson` da classe `DriveDataImporterRunnable`

```
1  private void importJson(Metadata jsonMetadata) {
2      DriveFile driveFile = Drive.DriveApi.getFile(googleApiClient,
3  jsonMetadata.getDriveId());
4      DriveApi.DriveContentsResult result =
5  driveFile.open(googleApiClient, DriveFile.MODE_READ_ONLY,
6  null).await();
7
8      if (!result.getStatus().isSuccess()) {
9          throw new ImportError("A importação falhou. Tente novamente.
10 Status do Google Drive: " + result.getStatus().getStatusMessage());
11     }
12
13     DriveContents contents = result.getDriveContents();
14
15     boolean isJson = true;
16     BackupDataImporter importer = new
17 BackupDataImporter(contents.getInputStream(), context, false, isJson);
18     try {
19         importer.importData();
20         eventBus.post(importer);
21     } catch (Exception e) {
22         e.printStackTrace();
23         ImportError error = new ImportError("A importação dos dados
24 falhou. " + e.getMessage(), e);
25         eventBus.post(error);
26     } finally {
27         IOUtils.closeQuietly(importer);
28     }
29 }
```

No Quadro 21 é possível ver o pedaço de código do método `importData` da classe `BackupDataImporter`, que importa os dados em JSON para as tabelas. Na linha número 4 é chamado o método `inputStreamToJson` que lê o `InputStream` recebido e faz o *parser* para um `JsonObject`. Na linha seguinte, é obtido uma instância do `SQLiteDatabase` com permissão para escrita. Em sequência, é iniciada uma transação com banco de dados, assim, se ocorrer algum erro durante a importação, os dados não ficarão corrompidos.

Quadro 21 – Importação dos dados em JSON para as tabelas da base de dados

```
1  protected void importData(InputStream inputStream) throws Exception {
2      Log.d(TAG, "import data: " + (this.json ? "json" : "zip"));
3      if (this.json) {
4          JsonObject root = inputStreamToJson(inputStream);
5          SQLiteDatabase database = dbHelper.getWritableDatabase();
6          try {
7              database.beginTransaction();
8
9              if (!merge) {
10                 cleanDatabase(database);
11             }
12             if (root != null) {
13                 GsonBuilder builder = new GsonBuilder();
14                // CÓDIGO...
15                 Gson gson = builder.create();
16
17                 importPatients(database, gson, root);
18                 importMoleGroups(database, gson, root);
19                 importImageRecords(database, gson, root);
20             }
21             database.setTransactionSuccessful();
22         } catch (Exception e) {
23             Log.e(TAG, e.getMessage(), e);
24             throw e;
25         } finally {
26             database.endTransaction();
27             database.close();
28         }
29     } else {
30         // CÓDIGO...
31     }
32 }
```

Na linha 9 é verificado se a opção escolhida é de *merge* (Importar backup) ou não (Limpar e restaurar), se não for de *merge*, é chamado o método `cleanDatabase`, para que sejam excluídos todos os dados da base. Nas linhas 12 a 19 é realizada a importação de cada uma das tabelas. Em seguida, na linha 21, é chamado o método que marca a transação como executada com sucesso e na linha 26 a transação é finalizada.

Além da importação dos dados em JSON, o método `importData` também realiza a importação das imagens do arquivo ZIP. O Quadro 22 demonstra o código responsável por realizar esse trabalho.

Quadro 22 – Importação das imagens

```
1  protected void importData(InputStream inputStream) throws Exception {
2      Log.d(TAG, "import data: " + (this.json ? "json" : "zip"));
3      if (this.json) {
4          // CÓDIGO...
5      } else {
6          ZipInputStream zin = new ZipInputStream(inputStream);
7          ZipEntry ze = null;
8          while ((ze = zin.getNextEntry()) != null) {
9              Log.d(TAG, ze.getName());
10             String name = ze.getName();
11             if (name.contains("/")) {
12                 String path = name.split("/")[0];
13                 String fileName = name.split("/")[1];
14
15                 long id = Long.parseLong(path.split("_")[0]);
16                 String patientName = path.split("_")[1];
17
18                 // obtém o diretório de imagens do paciente
19                 File patientImagesDir =
20 ImageUtils.getPatientImagesDir(id, patientName);
21                 if (!patientImagesDir.exists()) {
22                     patientImagesDir.mkdirs();
23                 }
24                 File imgFile = new File(patientImagesDir, fileName);
25
26                 if (!imgFile.exists()) {
27                     Log.d(TAG, "copying image: " + imgFile.getName());
28                     // copia a imagem apenas se não existir
29                     boolean created = imgFile.createNewFile();
30                     if (created) {
31                         FileOutputStream outputStream = new
32 FileOutputStream(imgFile);
33                         IOUtils.copy(zin, outputStream);
34                         outputStream.close();
35                     }
36                 }
37             }
38             zin.closeEntry();
39         }
40         zin.close();
41     }
42 }
```

Na linha 6 o `InputStream` recebido é encapsulado em um `ZipInputStream`. Nas linhas seguintes é iterado por cada entrada do arquivo ZIP, cada entrada representa uma imagem. A partir do nome da entrada possível obter o *id* do paciente (linha 15), na sequência é obtido o diretório onde as imagens do paciente ficam salvas, se a imagem ainda não existir no diretório, ela é copiada (linha 33).

3.4 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados os experimentos realizados com o aplicativo. A seção 3.4.1 apresenta o experimento de usabilidade, detalhando a metodologia utilizada e a análise dos resultados dos testes aplicados. Na seção 3.4.2 são demonstrados os experimentos de backup dos dados e compatibilidade do aplicativo com diferentes *hardwares* e versões do Android.

3.4.1 Experimento de usabilidade

O experimento de usabilidade foi realizado com doze usuários, com perfis variados, para avaliar a aceitação e eficiência das funcionalidades implementadas.

3.4.1.1 Metodologia

O experimento ocorreu no mês de novembro por meio de testes individuais com os voluntários. Os testes foram realizados com os dispositivos dos próprios usuários, onde foi disponibilizado o apk do aplicativo para realizar a instalação do mesmo. Cada usuário recebeu um questionário de perfil, uma lista de tarefas e um questionário de usabilidade, que estão disponíveis no Apêndice D.

3.4.1.2 Aplicação do teste

Inicialmente, os participantes foram orientados sobre o que é o mapeamento corporal e sobre o objetivo do aplicativo e dos testes de usabilidade. Em seguida, cada voluntário preencheu um questionário de perfil de usuário. A lista de tarefas foi composta por quatorze tarefas, e buscou contemplar as principais funcionalidades disponíveis no aplicativo. Ao final de cada tarefa, o usuário foi orientado a informar se foi possível executar o que foi solicitado, podendo complementar através de observações.

O questionário de usabilidade foi composto de dezoito perguntas fechadas e uma pergunta aberta. As perguntas procuraram obter as impressões do usuário sobre o aplicativo, sobre a facilidade de utilização e a importância das funcionalidades disponíveis. Ao responder o questionário os voluntários também puderam deixar suas críticas e sugestões sobre o aplicativo. Os resultados deste experimento são apresentados na próxima seção.

3.4.1.3 Análise e interpretação dos dados coletados

A primeira análise foi realizada sobre os dados coletados através do questionário de perfil de usuário. No Quadro 23 são apresentados os perfis dos usuários envolvidos no experimento.

Quadro 23 – Perfil dos usuários envolvidos no teste de usabilidade

Sexo	83% masculino 27% feminino
Idade	75% entre 18 e 25 anos 8% entre 25 e 35 anos 17% mais de 35 anos
Profissão	42% estudante 17% programador 8% motorista 8% negociador 8% estilista 8% costureira 8% técnico em informática
Uso de celular	100% todos os dias
Câncer de pele na família	8% sim 92% não
Já realizou exame dermatológico	50% sim 50% não
Costuma guardar os exames médicos	50% sim 50% não

A partir das respostas de sexo, idade e profissão, é possível perceber que foram entrevistadas pessoas de perfis variados, possibilitando a avaliação do aplicativo sob diferentes pontos de vista. É possível verificar que todos os usuários utilizam seus celulares todos os dias e, que dos entrevistados 50% já realizou exames dermatológicos. Apenas 50% dos entrevistados costuma guardar seus exames médicos e, somente 8% dos voluntários possui histórico de câncer de pele na família.

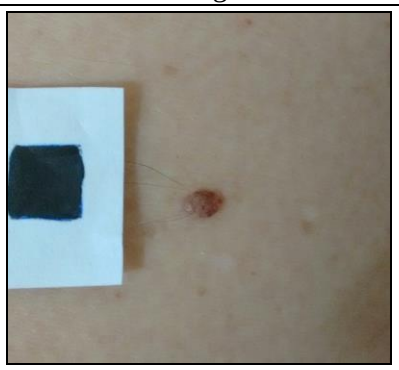
3.4.1.3.1 Análise dos resultados da lista de tarefas

Após a análise de perfil dos usuários, foi realizada a avaliação dos resultados obtidos a partir da lista de tarefas (Apêndice D). Todos os usuários conseguiram instalar o aplicativo com facilidade. O cadastro do paciente também foi realizado por todos os voluntários, alguns sugeriram que a opção para o cadastro poderia estar mais visível na tela inicial. Foi relatado que em alguns dispositivos o caractere de barra (/) para a data de nascimento não era apresentado no teclado.

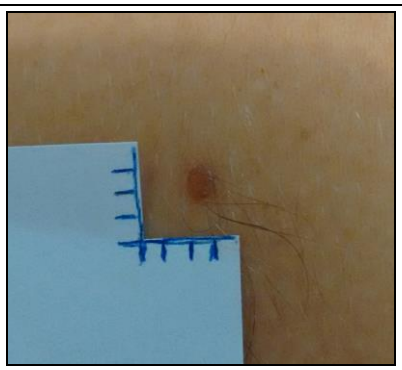
Após o cadastro e seleção do paciente, foi realizada a tarefa de criação do gabarito. Nos quatro primeiros questionários foi aplicado um tipo de gabarito, composto pelo desenho de um quadrado de cor escura (Figura 30 item a). Todos os usuários conseguiram criar o

gabarito. O aplicativo deveria realizar a detecção do gabarito e não permitir que a foto fosse salva caso o gabarito não estivesse presente. Em apenas um dos casos o aplicativo reconheceu o gabarito mesmo não estando presente. Isso se deve a presença excessiva de sombra na imagem.

Figura 30 – Gabaritos utilizados



(a) Gabarito preenchido



(b) Gabarito simulando régua

Contudo, para os questionários seguintes decidiu-se alterar o tipo de gabarito utilizado para um que poderia ser mais útil e que facilitaria a compreensão dos usuários (Figura 30 item b). O aplicativo não foi preparado para reconhecer este segundo tipo de gabarito, portanto, apenas orienta-se o usuário sobre a importância das imagens retiradas das pintas estarem acompanhadas do gabarito. No Quadro 24, são apresentados os resultados relacionados à captura e vinculação da imagem com a região corporal.

Quadro 24 – Respostas quanto às tarefas de captura e vinculação da imagem

Tarefas/Respostas	**Sim**	**Sim, mas com dificuldade**	**Não**
Capturar uma imagem com o uso do gabarito	83%	17%	
Vincular imagem: seleção da região corporal	83%	17%	
Vincular imagem: selecionar local específico da pinta em uma região inválida.	92%		8%
Vincular imagem: selecionar local específico da pinta	83%	17%	
Informar dados sobre a pinta	92%	8%	

A partir do Quadro 24, percebe-se que todos os usuários conseguiram realizar a captura da imagem, porém alguns tiveram dificuldades quando alteravam a orientação do dispositivo da vertical para a horizontal e vice-versa. Foi reportado que nessas situações o aplicativo parava de funcionar. Também foi relatado que é difícil segurar o gabarito e ao mesmo tempo capturar a imagem, o que dificulta o autoexame.

Quanto à vinculação da imagem com uma região corporal, todos os usuários realizaram a tarefa, no entanto, 17% relataram problemas na identificação da região por eles selecionada. Quando selecionavam uma região, o aplicativo apresentava outra. Este problema ocorreu devido à resolução dos dispositivos utilizados por estes usuários ser diferente dos

demais. Além disso, alguns usuários sugeriram inverter o lado direito e esquerdo da imagem do corpo humano, pois se confundiram pelo fato de a imagem estar invertida em relação ao usuário.

Foi solicitado que os usuários selecionassem uma região inválida quando fossem selecionar o local específico da pinta. O objetivo era validar o uso correto das *bounding boxes* na verificação se o local selecionado é válido ou não. Neste experimento, apenas um usuário conseguiu selecionar um local inválido, no restante dos casos o aplicativo validou corretamente. Na seleção de um local válido, houve dificuldades apenas em relação ao lado direito/esquerdo assim como na tarefa de seleção da região corporal.

Todos os usuários conseguiram informar os dados sobre a pinta, solicitados na tela de confirmação do local selecionado. Entretanto, um usuário relatou problemas em visualizar todos os campos pelo fato de a tela não ter *scroll*.

Após a vinculação da primeira imagem, seguiram as tarefas relacionadas à consulta e manutenção do mapeamento corporal, os resultados são apresentados no Quadro 25.

Quadro 25 – Tarefas relacionadas à consulta e manutenção do mapeamento corporal

Tarefas/Respostas	**Sim**	**Sim, mas com dificuldade**	**Não**
Consultar mapeamento corporal	92%	8%	
Vincular mais de uma imagem ao mesmo grupo (local)	75%	25%	
Consultar mapeamento corporal com mais de uma imagem da mesma pinta	92%		8%
Visualizar imagens de forma ampliada	100%		
Vincular uma imagem a outra região corporal	100%		

A partir do Quadro 25, observa-se que os usuários conseguiram realizar a consulta do mapeamento e visualizar a imagem capturada anteriormente, selecionando o local da pinta. Um usuário teve dificuldades em selecionar a pinta para visualizar a imagem, o problema foi atribuído à diferença na resolução do dispositivo em relação aos demais.

Foi solicitado que os usuários vinculassem uma nova imagem ao mesmo local da imagem capturada anteriormente. Todos conseguiram realizar a tarefa, porém alguns relataram dificuldades em selecionar exatamente o mesmo local. Após vincular mais de uma imagem ao mesmo local, os usuários deveriam ser capazes de visualizar as duas imagens através da consulta ao mapeamento corporal, apenas um usuário não conseguiu realizar essa tarefa, o mesmo atribuiu o problema à resolução do dispositivo.

Outra tarefa constante do questionário era poder visualizar as imagens capturadas de forma ampliada. Todos os usuários conseguiram executar essa tarefa com sucesso. Por fim, a última tarefa era vincular uma imagem a outra região corporal. Assim como na tarefa anterior,

todos os usuários conseguiram vincular uma nova imagem a outra região do corpo e visualizá-la através da consulta ao mapeamento.

3.4.1.3.2 Análise quanto à usabilidade e funcionalidades

Foi aplicado um questionário fechado com nove perguntas relacionadas à usabilidade do aplicativo e nove perguntas referentes às funcionalidades e interesse dos usuários pelo mapeamento corporal. O Quadro 26 apresenta as respostas das perguntas referentes à usabilidade.

Quadro 26 – Avaliação de usabilidade do protótipo

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Não concordo nem discordo**	**Discordo parcialmente**	**Discordo totalmente**
1. Foi fácil encontrar as informações que você precisou	34%	50%	8%	8%	
2. O design da interface do aplicativo é atraente	34%	66%			
3. A organização dos menus e comandos de ação (como botões e links) é lógica, permitindo encontrá-los facilmente na tela	58%	34%		8%	
4. Foi fácil navegar nos menus e telas do aplicativo	66%	34%			
5. É fácil lembrar como fazer as coisas no aplicativo	75%	25%			
6. Os símbolos e ícones são claros e intuitivos	67%	25%		8%	
7. A interface é semelhante dos demais aplicativos	25%	50%	25%		
8. Às vezes eu não sei o que fazer com este aplicativo	8%	16%	16%	25%	35%
9. Você precisaria do apoio de uma pessoa para usar este aplicativo	34%	34%	8%	8%	16%

Conforme as respostas do Quadro 26, a maioria dos usuários achou fácil encontrar as informações necessárias e acharam o design da interface do aplicativo atraente. Concordaram que foi fácil navegar nos menus e telas. Alguns sugeriram adicionar no aplicativo um “Como fazer” (*How to*) para orientar o usuário no primeiro uso.

Quanto aos símbolos e ícones, a maioria achou os mesmos claros e intuitivos. Alguns concordaram que a interface é semelhante a dos demais aplicativos, outros não concordaram nem discordaram. Alguns dos usuários assinalaram que, às vezes, não sabiam o que fazer com o aplicativo. Porém a maioria não teve essa dificuldade.

Mais da metade dos usuários admitiram que precisariam do apoio de alguém para usar o aplicativo, o motivo principal é a dificuldade em segurar o gabarito e ao mesmo tempo capturar a imagem.

Na sequência são analisadas as respostas das questões relacionadas às funcionalidades do aplicativo e sua eficácia em relação ao mapeamento corporal. O Quadro 27 exibe as respostas obtidas.

Quadro 27 – Respostas relacionadas às funcionalidades e à eficácia do aplicativo

Perguntas / Critérios de avaliação	**Concordo totalmente**	**Concordo parcialmente**	**Não concordo nem discordo**	**Discordo parcialmente**	**Discordo totalmente**
1. Você conseguiu compreender que o gabarito serve para acompanhar a evolução (aumento de tamanho) da pinta	50%	8%	25%	17%	
2. Você se sente mais motivado para fazer o acompanhamento das suas pintas	34%	34%	16%		16%
3. É importante que o aplicativo armazene os dados, prevendo a recuperação dos mesmos caso você perca o seu celular	100%				
4. Você achou importante ver a pinta de forma ampliada	42%	34%	34%		
5. Você achou importante que o aplicativo mostre a imagem da pinta atual do lado de uma capturada anteriormente para fins de comparação	75%	25%			
6. O aplicativo em algum momento parou inesperadamente	66%				34%
7. Você usaria novamente o aplicativo caso houvesse alguma falha no primeiro manuseio	66%	17%	17%		
8. Você usaria este aplicativo no dia a dia	8%	43%	8%	25%	16%
9. Você recomendaria este aplicativo para outras pessoas.	34%	34%	16%	16%	

Pouco mais da metade dos usuários conseguiram compreender a utilidade do gabarito em relação ao acompanhamento da evolução do tamanho da pinta. 68% dos usuários disseram se sentir mais motivados para fazer o acompanhamento de suas pintas. Alguns não souberam dizer e outros disseram não se sentir motivados.

Todos os usuários consideraram importante a possibilidade de armazenar os dados do aplicativo externamente, prevendo a recuperação dos mesmos se necessário. Todos concordaram com a importância de exibir a imagem da pinta atual ao lado de uma imagem da mesma pinta capturada anteriormente, pois facilita a comparação.

O aplicativo parou de funcionar em algum momento para mais da metade dos usuários. A maioria relatou que isso aconteceu na captura da imagem, quando alteravam a orientação do dispositivo do modo retrato para paisagem e vice-versa. Contudo, a maioria disse que usaria o aplicativo novamente no caso de haver uma falha no primeiro manuseio.

Apenas metade dos usuários disse que usaria o aplicativo no dia a dia e cerca de 40% disseram que não usariam e, 68% dos usuários relataram que indicariam o aplicativo para outras pessoas, dos 32% restantes, metade não soube dizer se indicaria e a outra metade acha que não indicaria.

3.4.2 Experimento de backup dos dados e compatibilidade

O experimento de backup foi realizado exportando os dados de um dispositivo e importando-os no mesmo dispositivo onde foi realizada a exportação e em um segundo aparelho. Nos dois casos o backup foi realizado corretamente.

Observou-se que a exportação das imagens pode ser lenta, dependendo da quantidade de imagens cadastradas e da velocidade da rede de internet utilizada. Contudo, o *upload* é realizado em background, por isso não trava a interface e o usuário pode continuar o uso do aplicativo durante a exportação.

Já o experimento de compatibilidade foi realizado utilizando doze dispositivos Android, possibilitando avaliar a compatibilidade do aplicativo em diferentes *hardwares* e versões do sistema operacional Android. Os dispositivos utilizados estão listados no Quadro 28. A maioria eram *smartphones*, sendo apenas um *tablet*.

Quadro 28 – Dispositivos utilizados no experimento

Quantidade	**Dispositivo**	**Versão do Android**	**Tamanho da tela**	**Resolução**
3	Motorola Moto G 2014 (dispositivo de desenvolvimento)	5.0.2	5	(1280x720)
2	Motorola Moto G 2014	4.4	5	(1280x720)
1	Motorola Moto G 2013	4.3	4.5	(1280x720)
1	Motorola Moto X 2013	5.1	4.7	(1280x720)
1	LG G3	5.0.1	5.5	(2560x1440)
1	Samsung Galaxy S4 Mini	4.4.2	4.3	(960x540)
1	Sony Xperia T3	4.4.4	5.3	(1280x720)
1	Galaxy Tab GT-P5110	4.1.2	10.1	(1280x800)
1	Não informado pelo usuário	-	-	

O aplicativo foi instalado com sucesso em todos os aparelhos e todas as funcionalidades puderam ser utilizadas. Entretanto, os usuários mencionaram problemas na

funcionalidade de vinculação da imagem com a região corporal nos dispositivos LG G3, Samsung Galaxy S4 Mini e Galaxy Tab. Isso se deve à resolução diferenciada destes dispositivos em relação ao de desenvolvimento. Além disso, o aplicativo mostrou-se compatível com versões do Android acima de 4.1.

3.4.3 Comparação com os trabalhos relacionados

O Quadro 29 apresenta um comparativo entre o aplicativo desenvolvido e os trabalhos relacionados. Os dados utilizados para comparação são baseados nos experimentos realizados.

Quadro 29 - Comparação com os trabalhos relacionados

Características / trabalhos	Pro-Cal (2014)	Foto Finder (2010a)	Canfield Scientific (2015)	Barbosa (2014)	Lima (2015)
Plataforma	iOS	iOS	Desktop	Android	Android
Backup	Não	Sim	Sim	Não	Sim
Padronização da captura	Sim	Sim	Sim	Não	Sim
Mais de um paciente	Não	Sim	Sim	Sim	Sim
Visualizar duas imagens simultaneamente	Sim	Sim	Sim	Não	Sim
Pode ser utilizado pelos pacientes	Sim	Não	Não	Sim	Sim

Com base no Quadro 29 é possível perceber que o aplicativo desenvolvido se difere do protótipo desenvolvido por Barbosa (2014) por apresentar opção de realizar backup dos dados, oferecer uma forma de padronizar a captura das imagens e permitir visualizar duas imagens simultaneamente.

A vantagem do aplicativo desenvolvido em relação ao aplicativo da Pro-Cal (2014) é a possibilidade de criar backup dos dados e de cadastrar mais de um paciente. Com relação ao aplicativo da Foto Finder (2010a), as diferenças são a plataforma na qual o aplicativo foi desenvolvido e o fato de o Foto Finder (2010a) ser para uso de médicos e não de pacientes.

O trabalho da Canfield Scientific (2015) é o que mais se difere do aplicativo desenvolvido, pois se trata de um software desktop para uso exclusivo de médicos. Enquanto o aplicativo desenvolvido neste trabalho pode ser utilizado pelos pacientes no autoexame e está disponível para a plataforma Android.

4 CONCLUSÕES

Este trabalho propôs o desenvolvimento de um aplicativo Android que permitisse realizar a manutenção do mapeamento corporal de lesões cutâneas. O mapeamento é realizado a partir da captura e vinculação de imagens a regiões corporais específicas.

Para o desenvolvimento do aplicativo foi utilizada a linguagem de programação Java no ambiente de desenvolvimento Android Studio. Não foram encontrados problemas na utilização das ferramentas.

Os resultados alcançados foram satisfatórios. O aplicativo atingiu o objetivo de realizar o mapeamento corporal, permitindo a captura de imagens através da câmera do dispositivo e possibilitando a vinculação da imagem à determinada região corporal. A possibilidade do usuário visualizar lado a lado duas imagens capturadas em diferentes períodos proporciona melhor comparação das mesmas, facilitando a identificação de mudanças ocorridas no intervalo entre um exame e outro. A funcionalidade de exportação e importação de backup dos dados fornece ao usuário a garantia de que seus dados não serão perdidos, podendo ser recuperados em casos como a perda do dispositivo.

Como limitação, destacam-se os problemas encontrados na vinculação da imagem com a região corporal em dispositivos que possuem resolução diferenciada em relação à resolução do dispositivo utilizado para o desenvolvimento do aplicativo.

É possível concluir que o desenvolvimento deste trabalho possibilitou uma alternativa para a realização do mapeamento corporal, que vem sendo considerado cada vez mais importante no diagnóstico precoce de doenças da pele, principalmente do melanoma. Por fim, este trabalho pode servir como base para futuros trabalhos em áreas como processamento de imagens voltado para o diagnóstico de doenças da pele.

4.1 EXTENSÕES

Para extensões deste trabalho propõem-se:

a) adicionar um algoritmo de classificação das imagens para o diagnóstico das lesões;
b) incluir um algoritmo capaz de verificar a presença do gabarito nas imagens;
c) realizar o cálculo do diâmetro das pintas;
d) incluir funcionalidades que guiem o usuário na manutenção do mapeamento corporal, avisando quando uma pinta precisa ser fotografada novamente;
e) estudar outras formas de padronizar a distância de captura das imagens, com o objetivo de facilitar o uso evitando a criação de um gabarito;
f) realizar um pré-processamento das imagens, visando a retirada de ruídos e

melhoria da qualidade;

g) efetuar melhorias de usabilidade.

REFERÊNCIAS

BARBOSA, Zuraica Jung. **Mapeamento corporal de lesões cutâneas.** 2014. 57 f. Trabalho de Conclusão de Curso (Bacharelando 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_2_zuraica-jung-barbosa-neta_monografia.pdf>. Acesso em: 11 fev. 2015.

CASTRO, Luís Guilherme Martins. **Mapeamento corporal de nevos ajuda a reconhecer precocemente o melanoma**. [S.l.], 2008. Disponível em: <http://www.fleury.com.br/medicos/educacao-medica/revista-medica/materias/Pages/mapeamento-corporal-de-nevos-ajuda-a-reconhecer-precocemente-o-melanoma.aspx>. Acesso em: 20 nov. 2015.

_____. **Mapeamento corporal acompanha a evolução dos nevos e evita cirurgias desnecessárias**. [S.l.], 2009. Disponível em: <http://www.fleury.com.br/medicos/educacao-medica/revista-medica/materias/Pages/mapeamento-corporal-acompanha-a-evolucao-dos-nevos-e-evita-cirurgias-desnecessarias.aspx>. Acesso em: 20 nov. 2015.

CANFIELD SCIENTIFIC. **DermaGraphix:** Manage high risk patients for melanoma with state-of-the-art body mapping software for tracking pigmented lesions. [S.l.], 2015. Disponível em: < http://www.canfieldsci.com/imaging-systems/mirror/#dermagraphix/>. Acesso em: 20 mar. 2015.

FOTO FINDER. **Handyscope - Mobile dermatoscope:** Application**.** 2010a. Disponível em: <http://www.handyscope.net/en/application.html>. Acesso em: 12 mar. 2015.

_____. **Handyscope - Mobile dermatoscope:** Device**.** 2010b. Disponível em: <http://www.handyscope.net/en/device.html>. Acesso em: 12 mar. 2015.

_____. **Handyscope - Mobile dermatoscope:** FotoFinder Hub**.** 2010c. Disponível em: <http://www.handyscope.net/en/fotofinder-hub.html>. Acesso em: 12 mar. 2015.

GOULART, Jacqueline M. et al. Dermoscopy in Skin Self-examination: A Useful Tool for Select Patients. **Archives of Dermatology**, [S.l.], v.147, n. 1, Jan. 2011. Disponível em: <http://archderm.jamanetwork.com/article.aspx?articleid=426469&resultClick=3>. Acesso em: 29 mar. 2015.

INSTITUTO NACIONAL DO CÂNCER. **Estimativa 2014**. [S.l.], 2014. Disponível em: <http://www.inca.gov.br/estimativa/2014/sintese-de-resultados-comentarios.asp>. Acesso em: 03 abr. 2015.

_____. **Pele melanoma.** [S.l.], 2015a. Disponível em: <http://www2.inca.gov.br/wps/wcm/connect/tiposdecancer/site/home/pele_melanoma>. Acesso em: 20 nov. 2015.

_____. **Pele não melanoma.** [S.l.], 2015b. Disponível em: <http://www2.inca.gov.br/wps/wcm/connect/tiposdecancer/site/home/pele_nao_melanoma>. Acesso em: 20 nov. 2015.

MALVEHY, Josep; PUIG, Susana. Follow-up of Melanocytic Skin Lesions with Digital Total-Body Photography and Digital Dermoscopy: A Two-Step Method. **Clinics in Dermatology**, [S.l.], v. 20, n. 3, Jun. 2002. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0738081X02002201#>. Acesso em: 29 mar. 2015.

MAYER, Jonathan E. et al. Screening, early detection, education, and trends for melanoma: Current status (2007-2013) and future directions: Part I. Epidemiology, high-risk groups, clinical strategies, and diagnostic technology. **Journal of the American Academy of Dermatology**, [S.l.], v. 71, n. 4, Out. 2014. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0190962214015345>. Acesso em: 30 mar. 2015.

PEREIRA, Gustavo A. **Câncer de pele.** São Paulo, 2012a. Disponível em: <http://www.cancerdepele.net.br/cancer-de-pele>. Acesso em: 10 nov. 2015.

_____. **Mapeamento corporal e dermatoscopia digital.** São Paulo, 2012b. Disponível em: <http://www.cancerdepele.net.br/mapeamento>. Acesso em: 12 mar. 2015.

_____. **Melanoma.** São Paulo, 2012c. Disponível em: <http://www.cancerdepele.net.br/melanoma>. Acesso em: 10 nov. 2015.

PRO-CAL. **Mole Monitor App - Skin Cancer Symptom Checker - Mole Monitor | Skin Cancer Symptom Checker.** 2014. Disponível em: <http://www.mole-monitor.com/mole-monitor-app.html>. Acesso em: 19 mar. 2015.

SALERNI, Gabriel et al. Benefits of Total Body Photography and Digital Dermoscopy ("Two-Step Method of Digital Follow-Up") In The Early Diagnosis Of Melanoma In High-Risk Patients. **Journal of the American Academy of Dermatology**, [S.l.], v. 67, n. 1, Jul. 2012. Disponível em: < http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3215791/>. Acesso em: 30 mar. 2015.

SAVORY, Stephanie. **Total Body Photography and its Use in the HighRisk Melanoma Patient**. Dallas, 2015. Disponível em: <https://www.dermquest.com/expert-opinions/clinical-updates/2015/tbp/>. Acesso em: 10 nov. 2015.

SOUZA, Reynaldo J. S. P. et al. Estimativa do custo do tratamento de câncer de pele tipo melanoma no Estado de São Paulo – Brasil. **Anais Brasileiros de Dermatologia**. Ribeirão Preto, v. 84, n. 3, 2009. Disponível em: <http://www.scielo.br/pdf/abd/v84n3/v84n03a04.pdf>. Acesso em: 03 abr. 2015.

TYAGI, Abhilasha; MILLER, Kimberly ; COCKBURN, Myles. **e-Health tools for targeting and improving melanoma screening:** a review**.** Los Angeles, 2012. Disponível em: <http://www.hindawi.com/journals/jsc/2012/437502>. Acesso em: 01 abr. 2015.

APÊNDICE A – Detalhamento dos casos de uso

A seguir é apresentado um detalhamento dos casos de uso, com descrição dos cenários.

O caso de uso UC01 - Cadastrar Paciente descreve a relação entre o Usuário e a funcionalidade que permite o cadastro de novos pacientes. Os detalhes deste caso de uso são apresentados no Quadro 30.

Quadro 30 – Caso de uso UC01 - Cadastrar Paciente

Número	01
Caso de Uso	Cadastrar paciente
Descrição	Este caso de uso tem por objetivo efetuar o cadastro de um novo paciente no aplicativo.
Ator	Usuário
Pré-condições	Estar na tela inicial do aplicativo.
Cenário principal	1. O Usuário seleciona a opção cadastrar novo paciente; 2. O aplicativo abre a tela de cadastro de pacientes; 3. O Usuário informa o nome, CPF, data de nascimento e o sexo do paciente; 4. O Usuário pressiona o botão Salvar; 5. O aplicativo salva os dados do paciente e apresenta uma mensagem de sucesso e volta a exibir a tela de seleção de paciente.
Fluxo Alternativo 1	No passo 4, o Usuário pode voltar à tela anterior sem salvar os dados

O caso de uso UC02 – Capturar imagem descreve a relação entre o Usuário e a funcionalidade que permite a captura de imagens através da câmera do dispositivo. Detalhes deste caso de uso estão descritos no Quadro 31.

Quadro 31 – Caso de uso UC02 - Capturar imagem

Número	02
Caso de Uso	Capturar imagem
Descrição	Este caso de uso tem por objetivo capturar a imagem de uma pinta
Ator	Usuário
Pré-condições	Estar no painel principal do aplicativo ou na tela de histórico da pinta
Cenário principal	1. O Usuário seleciona a opção capturar imagem; 2. O aplicativo abre a câmera do dispositivo; 3. O Usuário posiciona a câmera de tal forma que a pinta e o gabarito estejam na região de interesse; 4. O Usuário toca a tela para capturar a fotografia; 5. O aplicativo corta a fotografia para salvar apenas a região de interesse; 6. O aplicativo apresenta a foto para o Usuário e aguarda sua confirmação. 7. O Usuário confirma a fotografia. 8. O aplicativo salva a imagem.
Fluxo Alternativo1	No passo 7, o Usuário pode optar por descartar a imagem, o fluxo retorna ao passo 2.

O caso de uso UC03 – Vincular imagem a região do corpo descreve a relação entre o Usuário e a funcionalidade que realiza a vinculação da imagem a uma determinada região do corpo. Detalhes deste caso de uso estão descritos no Quadro 32.

Quadro 32 – Caso de uso UC03 – Vincular imagem a região do corpo

Número	03
Caso de Uso	Vincular imagem a região do corpo
Descrição	Este caso de uso tem por objetivo vincular uma imagem a uma determinada região do corpo.
Ator	Usuário
Pré-condições	UC02 - Capturar imagem
Cenário principal	1. O aplicativo apresenta uma tela com o desenho do corpo humano; 2. O Usuário seleciona a parte do corpo com a qual deseja vincular a imagem; 3. O aplicativo apresenta uma nova tela com os detalhes da parte do corpo escolhida no passo anterior; 4. O Usuário seleciona a região específica que a imagem deve ser vinculada; 5. O aplicativo desenha um ponto sobre o local selecionado e solicita confirmação da localização da pinta; 6. O Usuário confirma a localização; 7. O aplicativo salva os dados e retorna à tela principal.
Fluxo Alternativo	No passo 6, se o Usuário não confirmar a localização da pinta, o fluxo retorna ao passo 4.

O caso de uso UC04 – Visualizar imagens descreve a relação entre o Usuário e a funcionalidade que permite a visualização das imagens capturadas através da navegação pelo diagrama corporal. Detalhes deste caso de uso estão descritos no Quadro 33.

Quadro 33 – Caso de uso UC04 - Visualizar imagens

Número	04
Caso de Uso	Visualizar imagens
Descrição	Este caso de uso tem por objetivo visualizar as imagens previamente capturadas.
Ator	Usuário
Pré-condições	UC03 – Vincular imagem a região do corpo
Cenário principal	1. O Usuário seleciona a opção Consultar diagrama corporal; 2. O aplicativo abre a tela com os desenhos das partes do corpo; 3. O usuário navega pelas partes do corpo e seleciona a parte que deseja consultar; 4. O aplicativo amplia a parte do corpo escolhida pelo usuário; 5. O usuário seleciona de qual pinta que deseja ver as imagens; 6. O aplicativo apresenta uma tela onde é possível consultar as imagens da pinta selecionada e informações sobre a mesma.

O caso de uso UC05 – Fazer backup dos dados descreve a relação entre o Usuário e a funcionalidade que permite realizar backup dos dados, exportando-os para o Google Drive. Detalhes deste caso de uso estão descritos no Quadro 34.

Quadro 34 – Caso de uso UC05 - Fazer backup dos dados

Número	05
Caso de Uso	Fazer backup dos dados
Descrição	Este caso de uso tem por objetivo realizar backup dos dados do aplicativo
Ator	Usuário
Pré-condições	
Cenário principal	1. O Usuário seleciona a opção Criar backup; 2. O aplicativo abre a tela do Google Drive para o usuário selecionar a pasta em que os dados serão salvos; 3. O Usuário seleciona a pasta onde deseja salvar os dados; 4. O aplicativo faz *upload* dos dados para a pasta selecionada pelo usuário.
Fluxo Alternativo	No passo 2, caso seja a primeira vez que o Usuário realiza a ação de backup, o Google Drive irá solicitar que ele informe qual conta do Google deseja utilizar.

O caso de uso UC06 – Importar backup dos dados descreve a relação entre o Usuário e a funcionalidade que permite importar um backup dos dados realizado anteriormente. Detalhes deste caso de uso estão descritos no Quadro 35.

Quadro 35 – Caso de uso UC06 - Importar backup dos dados

Número	06
Caso de Uso	Importar backup dos dados
Descrição	Este caso de uso tem por objetivo importar um backup dos dados do aplicativo
Ator	Usuário
Pré-condições	UC05 - Fazer backup dos dados
Cenário principal	1. O Usuário seleciona a opção Importar backup; 2. O aplicativo abre a tela do Google Drive para o usuário selecionar a pasta onde estão os dados a serem importados; 3. O Usuário seleciona a pasta onde estão os dados; 4. O aplicativo faz *download* dos dados, salvando-os novamente no dispositivo.

O caso de uso UC07 – Criar Gabarito descreve a relação entre o Usuário e a funcionalidade que auxilia o usuário na aquisição do gabarito a ser utilizado na captura das imagens. Detalhes deste caso de uso estão descritos no Quadro 36.

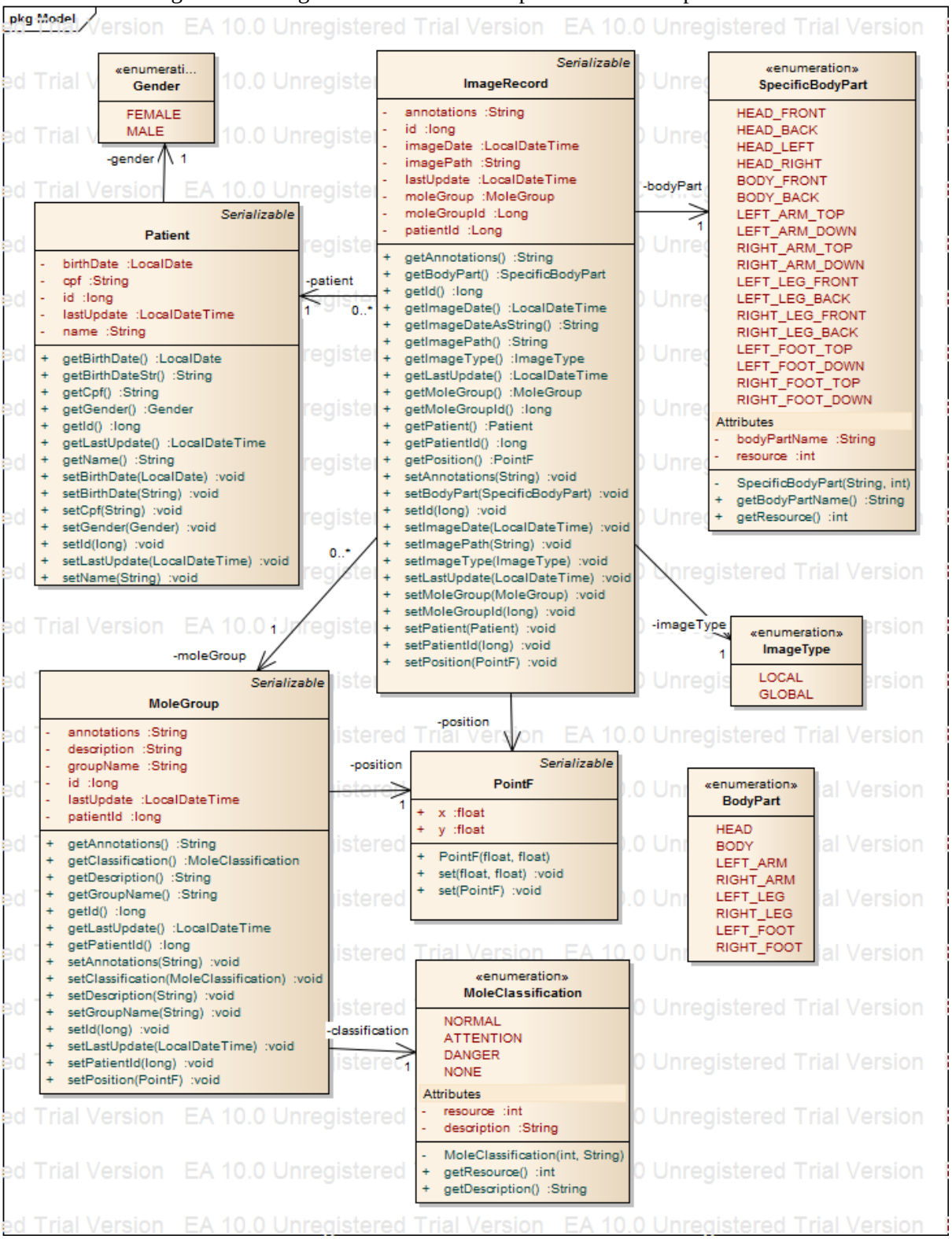
Quadro 36 – Caso de uso UC07 - Criar gabarito

Número	07
Caso de Uso	Criar gabarito
Descrição	Este caso de uso tem por objetivo criar o gabarito a ser utilizado na captura das imagens
Ator	Usuário
Pré-condições	
Cenário principal	1. O Usuário seleciona a opção Gabarito; 2. O aplicativo abre a tela de aquisição do gabarito; 3. O Usuário segue os passos descritos da tela.

APÊNDICE B – Diagramas de classes do aplicativo

Neste apêndice estão os diagramas de classes do aplicativo desenvolvido. Primeiramente são apresentados os diagramas de classe dos sub-pacotes do pacote de dados. A Figura 31 apresenta o diagrama de classes de modelo do pacote de dados.

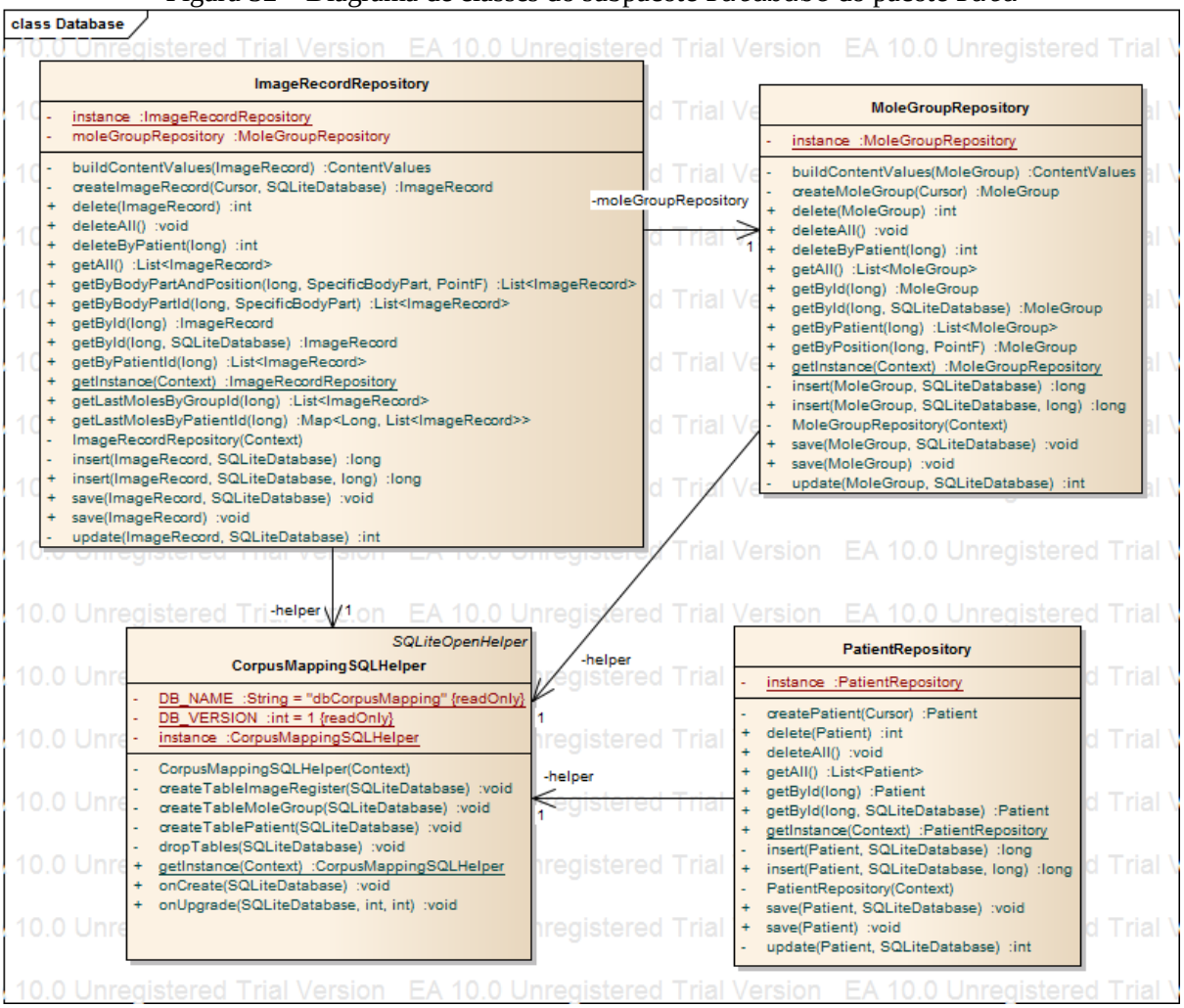
Figura 31 – Diagrama de classes do subpacote `Model` do pacote `Data`



A classe `Patient` (Figura 31) representa um paciente, esta possui uma associação com a enumeração `Gender`, que representa o sexo do paciente. A classe `ImageRecord` representa os dados da imagem de uma pinta, possui associação com as classes `Patient` e `MoleGroup`, e com as enumerações `SpecificBodyPart` e `ImageType` que representam as partes do corpo humano de forma detalhada e o tipo da imagem respectivamente. A classe `MoleGroup` representa o grupo de imagens de uma pinta, possui associação com a classe `PointF`, que representa um ponto com as coordenadas `xy` utilizadas para armazenar a posição de uma pinta. Possui também associação com a enumeração `MoleClassification`, utilizada para a classificação de risco de uma pinta. Há ainda a enumeração `BodyPart` que representa as partes do corpo humano.

Na sequência, a Figura 32 apresenta o diagrama das classes responsáveis pelo acesso à base de dados.

Figura 32 – Diagrama de classes do subpacote `Database` do pacote `Data`

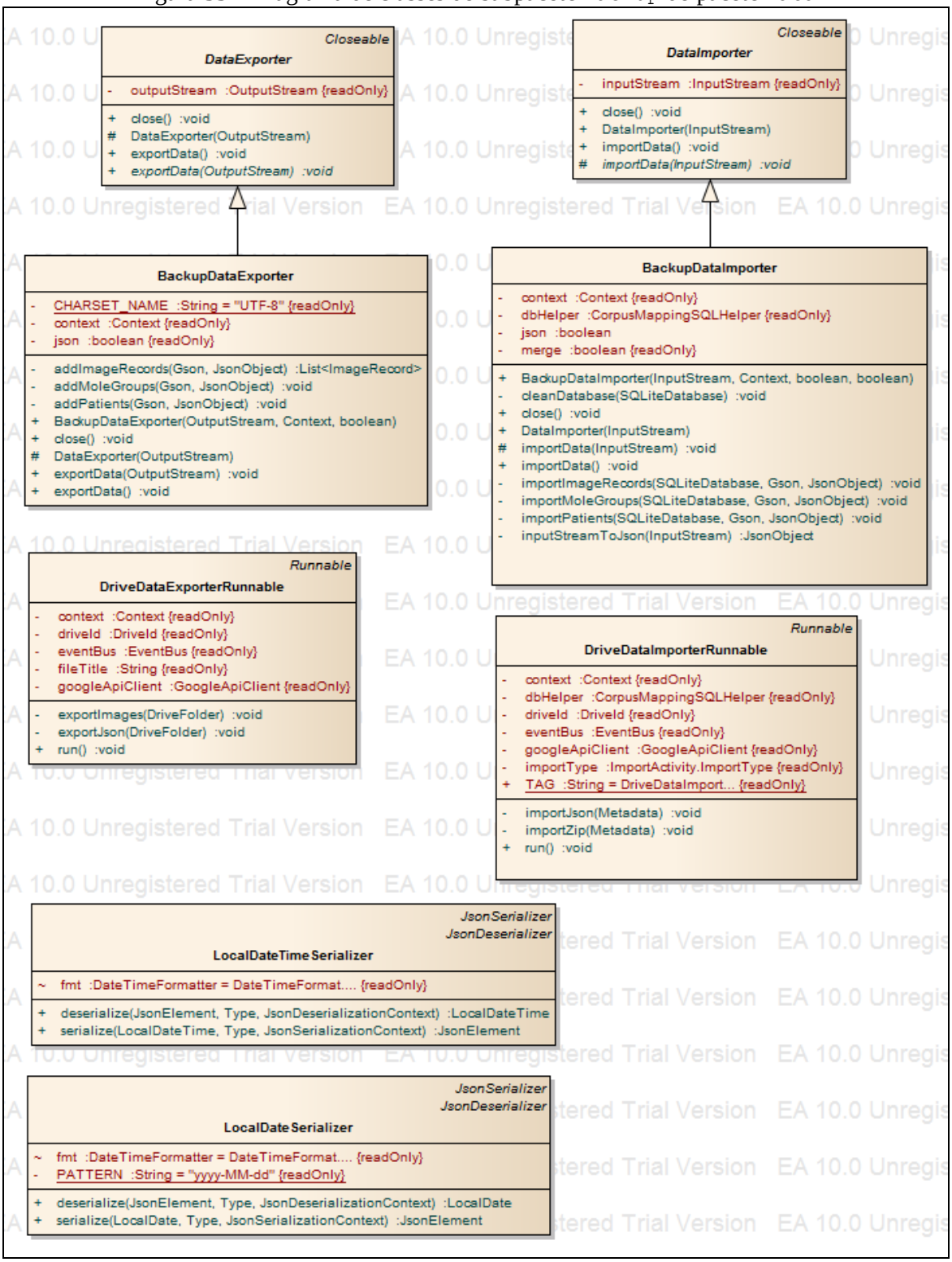


A classe `CorpusMappingSQLHelper` estende a classe `SQLiteOpenHelper` do Android e é responsável pela criação e versionamento da base de dados. As classes

`PatientRepository`, `ImageRecordRepository` e `MoleGroupRepository` são responsáveis por obter, inserir e alterar dados das tabelas que armazenam os dados de pacientes, imagens e grupos.

A Figura 33 exibe o diagrama das classes que realizam a exportação e importação de dados de backup.

Figura 33 – Diagrama de classes do subpacote `Backup` do pacote `Data`

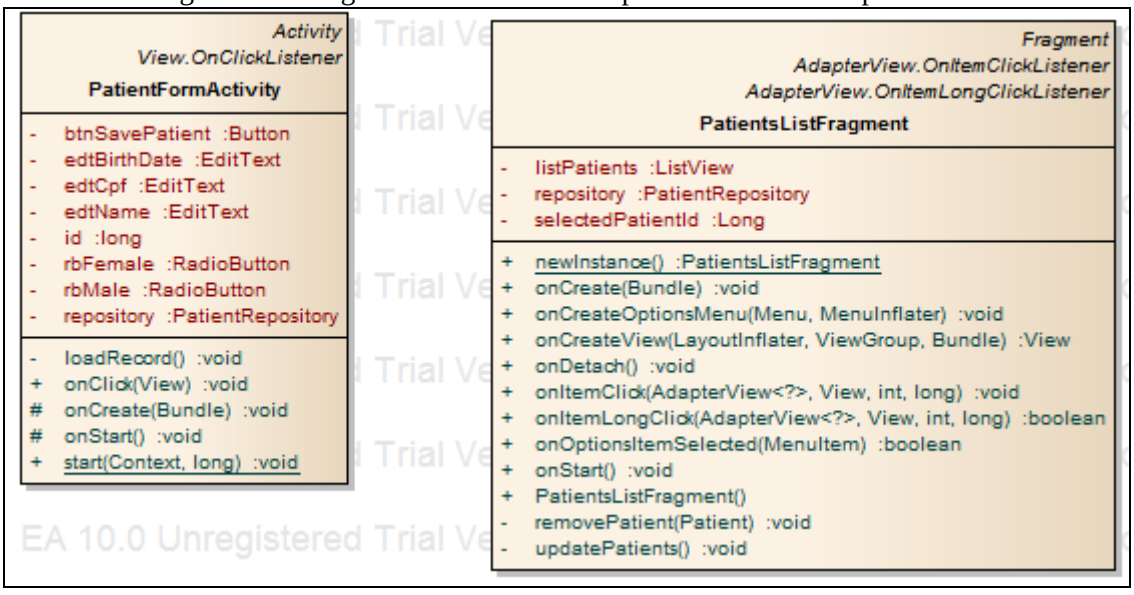


A classe `BackupDataExporter` (Figura 33) implementa a interface `DataExporter` e é responsável por ler os dados da base de dados, transformá-los para o formato JSON, ler as imagens do cartão de memória, compactá-las em um arquivo zip e escrever os dados em um `OutputStream` recebido da classe `DriveDataExporterRunnable`. A classe `DriveDataExporterRunnable` implementa a interface `Runnable` do Java, ela é executada após o usuário selecionar em qual diretório do Google Drive os dados devem ser salvos. Essa classe chama a classe `BackupDataExporter` informando o `OutputStream` onde os dados devem ser escritos e após esse procedimento realiza o *upload* para o Drive.

A classe `BackupDataImporter` implementa a interface `DataImporter`, sua responsabilidade é ler o `InputStream` que contém os dados em JSON e o arquivo zip com as imagens, e realizar a importação dessas informações para a base de dados do aplicativo. Sua execução é acionada pela classe `DriveDataImporterRunnable`, que implementa a interface `Runnable` do Java e é executada após o usuário selecionar em qual diretório do Google Drive estão os dados a serem importados.

A seguir, são apresentados os diagramas de classe dos sub-pacotes do pacote interface do usuário. A Figura 34 apresenta as classes responsáveis pelo cadastro e seleção de pacientes.

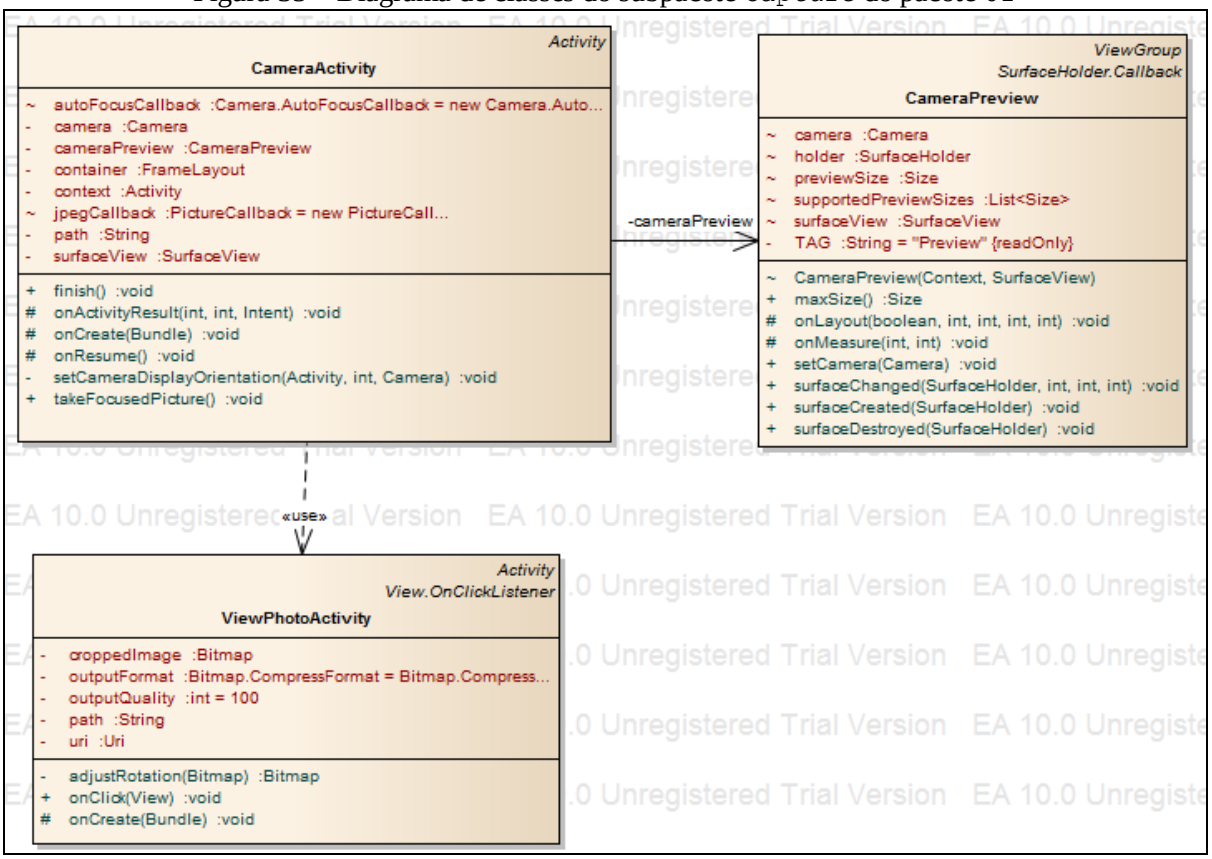
Figura 34 – Diagrama de classes do subpacote `Patient` do pacote `UI`



A classe `PatientFormActivity` (Figura 34) é responsável por apresentar uma tela com os campos necessários ao cadastro do paciente. A classe `PatientListFragment` apresenta os pacientes cadastrados em um componente `ListView`, e é utilizada pela classe `MainActivity` do pacote `Main`.

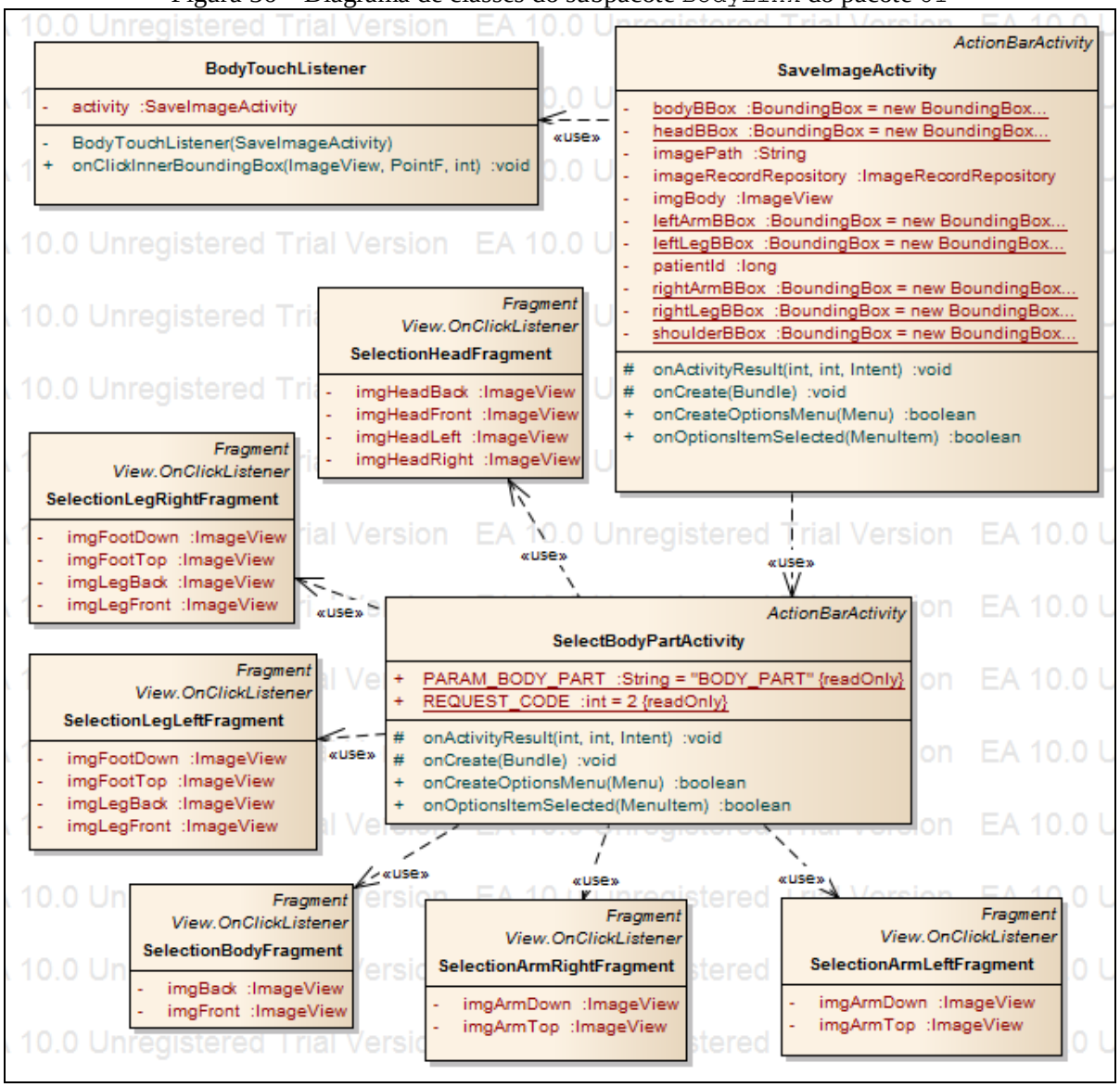
O diagrama das classes que realizam a captura de imagens é apresentado na Figura 35. A classe `CameraActivity` é responsável por abrir a câmera do dispositivo e, em conjunto com a classe `CameraPreview`, define onde será realizada a renderização do *preview* da câmera. Após a captura da imagem, a classe `CameraActivity` chama a abertura da `ViewPhotoActivity`. A classe `ViewPhotoActivity` realiza um processamento na imagem capturada para posteriormente apresentá-la ao usuário.

Figura 35 – Diagrama de classes do subpacote `Capture` do pacote `UI`



Na Figura 36 são apresentadas as classes responsáveis pela vinculação da imagem capturada com uma parte do corpo.

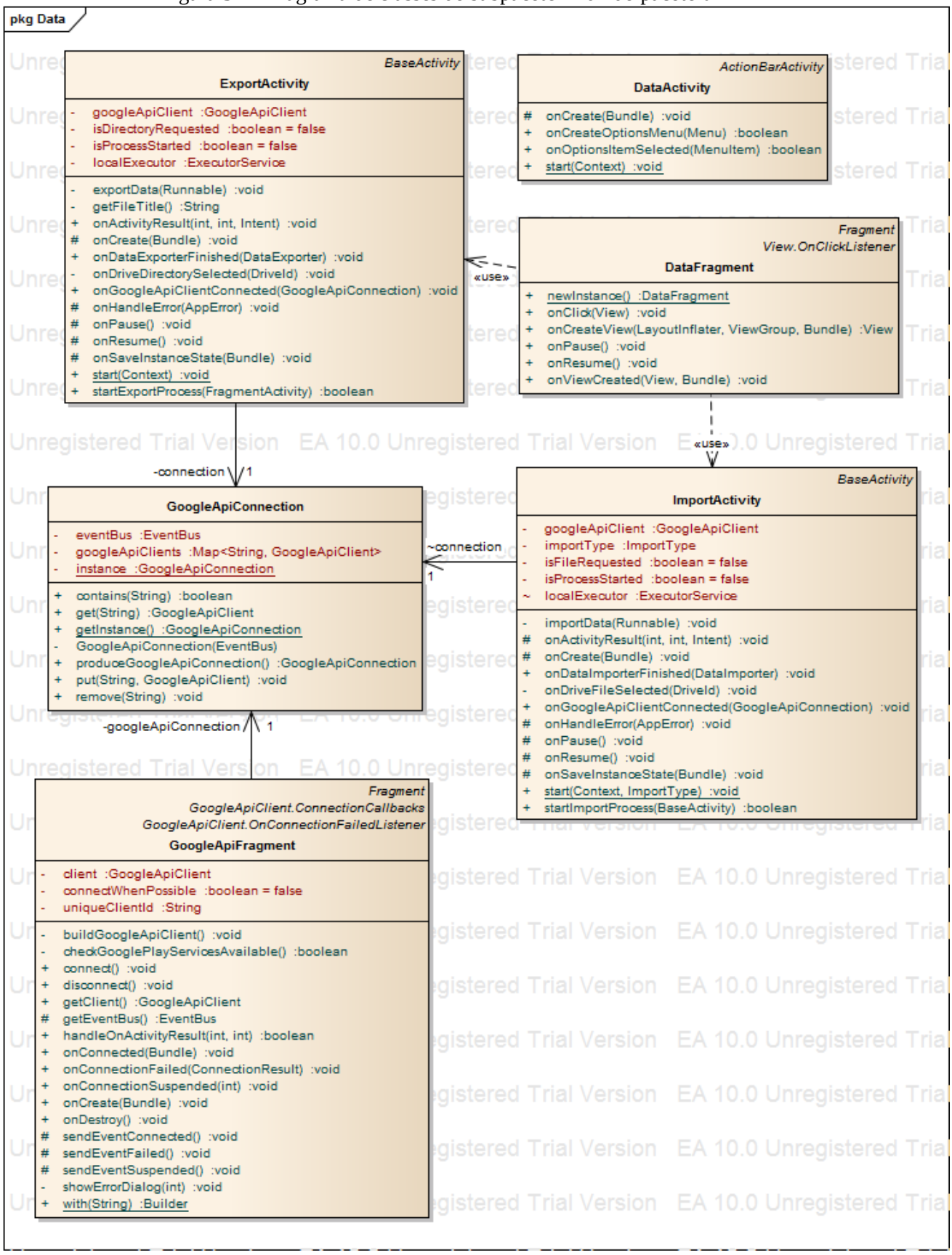
Figura 36 – Diagrama de classes do subpacote `BodyLink` do pacote `UI`



A classe `SaveImageActivity` (Figura 36) é invocada pela `ViewPhotoActivity` do pacote `Capture` quando o usuário confirma a utilização da imagem. Sua responsabilidade é salvar a imagem capturada. A classe `SelectBodyPartActivity`, dependendo da parte do corpo selecionada na `SaveImageActivity`, adiciona um *fragment* diferente na tela. Se for selecionada a região da cabeça, é adicionado o `SelectionHeadFragment`. Se a região selecionada for o tronco, é utilizado o `SelectionBodyFragment`. Caso a região selecionada for um dos braços, é apresentado o *fragment* correspondente com o lado selecionado, que pode ser o `SelectionArmRightFragment` para o braço direito, ou o `SelectionArmLeftFragment` para o braço esquerdo. O mesmo para a região das pernas, será apresentado o `SelecionLegRightFragment` para a perna direita, e para a perna esquerda o `SelecionLegLeftFragment`.

As classes que tratam da visualização das imagens capturadas estão demonstradas na Figura 37.

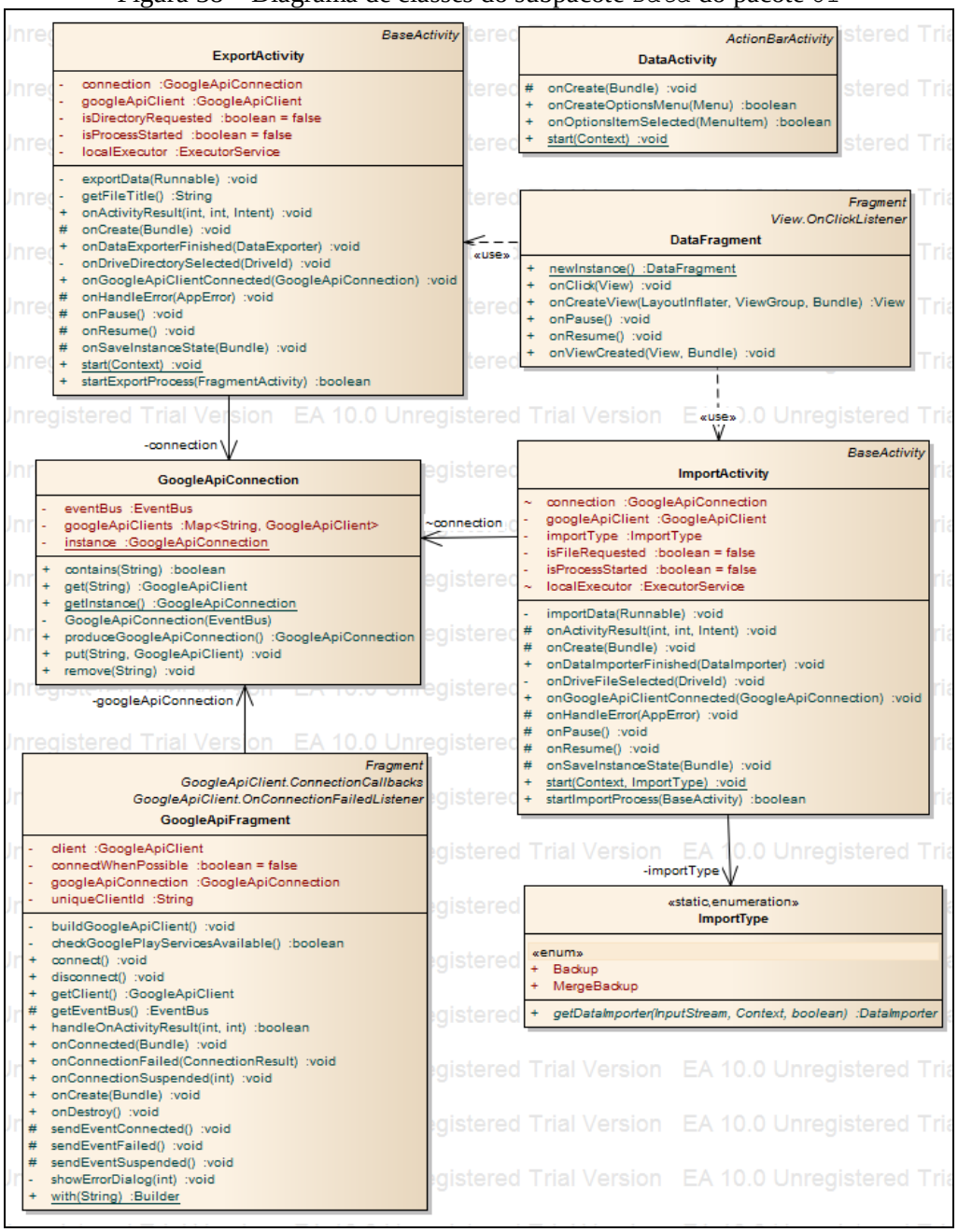
Figura 37 – Diagrama de classes do subpacote `View` do pacote `UI`



A classe `ViewBodyDiagramActivity` (Figura 37) utiliza a classe `BodyDiagramTabsAdapter` para apresentar as partes do corpo na forma de abas, cada aba contém um *fragment* que apresenta uma parte do corpo. Na aba da cabeça é utilizado o `ViewHeadMolesFragment`. Na aba do tronco é utilizado o `ViewBodyMolesFragment`. A próxima aba é a dos braços, que utiliza o `ViewArmMolesFragment`. E, por fim, a aba das pernas utiliza o `ViewLegMolesFragment`. A classe `MoleImageSliderActivity` é responsável por apresentar o histórico de imagens de uma pinta. A classe `MoleDetailedImageSliderActivity` apresenta uma imagem de forma ampliada.

A Figura 38 exibe o diagrama das classes que tratam as opções de backup.

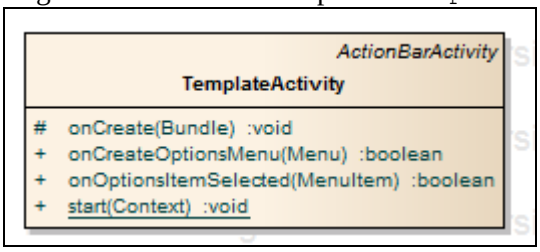
Figura 38 – Diagrama de classes do subpacote `Data` do pacote `UI`



A classe `DataActivity` (Figura 38) juntamente com a classe `DataFragment`, é responsável por apresentar as opções de criar backup, importar backup e limpar e restaurar os dados. Para a opção de criar backup, é chamado a abertura da `ExportActivity`, que inicia o processo de exportação dos dados para o Google Drive. Para as opções de importar backup e limpar e restaurar os dados é chamado a abertura da `ImportActivity`, que inicia o processo de importação dos dados do Google Drive para o aplicativo. As classes GoogleApiFragment e GoogleApiConnection são utilizadas pelas classes `ExportActivity` e `ImportActivity` para gerenciar a conexão com os serviços do Google.

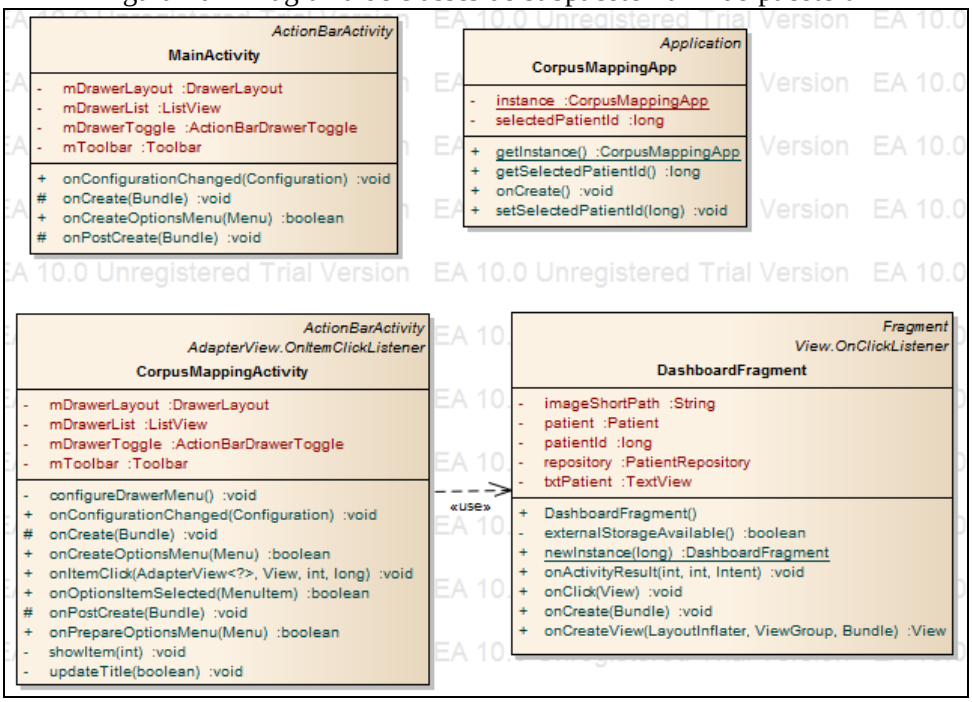
Na Figura 39, é apresentado o diagrama de classes do sub-pacote `Template`, do pacote de interface do usuário. A classe `TemplateActivity` (Figura 39) é responsável por apresentar ao usuário uma tela com orientações de como criar o gabarito a ser utilizado na captura das imagens.

Figura 39 – Diagrama de classes do subpacote `Template` do pacote `UI`



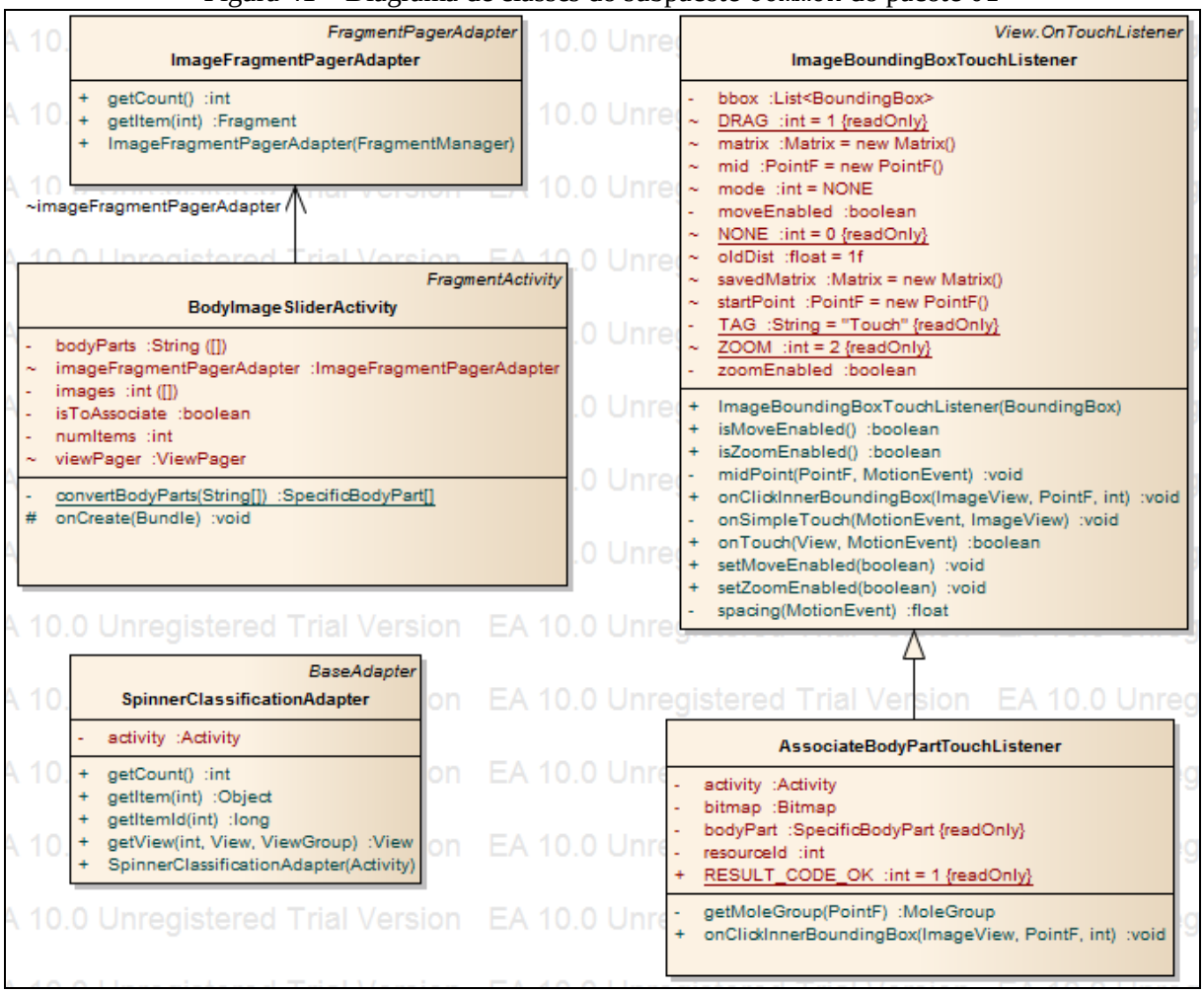
A Figura 40 apresenta o diagrama do sub-pacote `Main`. A classe `MainActivity` é a primeira tela aberta no aplicativo, ela utiliza o `PatientListFragment` do pacote `Patient` para listar os pacientes e solicita que o usuário selecione um paciente para continuar.

Figura 40 – Diagrama de classes do subpacote `Main` do pacote `UI`



A seguir, a **Erro! Autoreferência de indicador não válida.** apresenta o diagrama do pacote `Common`. A classe `ImageBoundingBoxTouchListener` é responsável por verificar se um toque do usuário foi realizado dentro de alguma *bounding box*. A classe `AssociateBodyPartTouchListener` estende a classe `ImageBoundingBoxTouchListener` e é utilizada tanto na vinculação da imagem com uma região corporal quanto na visualização das imagens na consulta do mapeamento corporal.

Figura 41 – Diagrama de classes do subpacote `Common` do pacote `UI`



A classe `BodyImageSliderActivity` apresenta os desenhos das partes do corpo de forma ampliada, sendo utilizada na vinculação e na consulta do mapeamento. Por fim, a classe `SpinnerClassificationAdapter` disponibiliza as opções de classificação de risco de uma pinta, sendo utilizada na vinculação, no momento de salvar uma imagem e na visualização, quando é permitido alterar os dados da pinta.

Por fim, a Figura 42 apresenta o diagrama de classes do pacote `Util`, que contém classes auxiliares utilizadas nos outros pacotes da aplicação.

Figura 42 – Diagrama de classes do pacote `Util`

A classe `ImageUtils` contém métodos para manipulação de imagens, como recorte e obtenção do caminho onde uma imagem está salva no cartão de memória. A classe `ImageDrawer` disponibiliza funções para desenhar pontos que representam as pintas nos desenhos das partes do corpo.

A classe `ZipManager` contém um método para compactar uma lista de arquivos em um arquivo ZIP. A classe `BoudingBox` guarda os valores mínimos e máximos de uma *bounding box*. Por fim, a classe `BodyImagesUtil` disponibiliza métodos para obter as *bounding boxes* de cada região corporal.

APÊNDICE C – Exemplo de arquivo JSON gerado no backup

Neste apêndice é demonstrado o formato JSON do arquivo com os dados do aplicativo que é gerado no backup (Quadro 37).

Quadro 37 – JSON gerado no backup


```
{
   "patients":[
      {
         "birthDate":"1987-08-11",
         "cpf":"08905632087",
         "gender":"MALE",
         "name":"Ezequiel",
         "id":1
      },{
         "birthDate":"1993-06-02",
         "cpf":"08819508907",
         "gender":"MALE",
         "name":"Janaina",
         "id":2
      }
   ],
   "moleGroups":[
      {
         "classification":"NORMAL",
         "groupName":"Braço direito",
         "position":{ "x":90.12232,"y":183.69887},
         "id":2,
         "patientId":1
      },{
         "classification":"NORMAL",
         "groupName":"Pescoço",
         "position":{ "x":211.12346,"y":348.7414},
         "id":3,
         "patientId":1
      }
   ],
   "imageRecords":[
      {
         "annotations":"cor diferente",
         "bodyPart":"BODY_FRONT",
         "imageDate":"2015-10-18 15:11:37",
         "imagePath":"1_Ezequiel/20151018_150857.jpg",
         "imageType":"LOCAL",
         "moleGroupId":1,
         "patientId":1,
         "position":{
            "x":61.876595,
            "y":75.78848
         },
         "id":1
      },
   ]
}
```

APÊNDICE D – Lista de tarefas e questionário de avaliação 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 38 – Questionário de perfil do usuário

Corpus Mapping – Aplicativo para o mapeamento corporal de lesões cutâneas

Olá!

Você está sendo convidado a participar de uma avaliação do aplicativo de mapeamento corporal de lesões cutâneas Corpus Mapping. Este aplicativo tem como principal objetivo, disponibilizar uma forma de realizar o mapeamento corporal através de dispositivos móveis.

O mapeamento corporal é utilizado por dermatologistas para acompanhar mudanças que ocorrem nas pintas dos pacientes ao longo do tempo, permitindo o diagnóstico precoce de doenças da pele como o Melanoma, tipo de câncer de pele agressivo que pode ser fatal se descoberto em um estágio mais avançado.

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

Profissão:_____

Você utiliza celular com qual frequência?

( ) todos os dias
( ) 2 a 3 vezes por semana
( ) 1 vez por semana
( ) A cada 15 dias
( ) 1 vez ao mês

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

( ) alarme
( ) calendário
( ) foto
( ) áudio
( ) vídeos
( ) agenda
( ) outros aplicativos ( ) facebook ( ) whatsapp. Especifique quais _____

Você tem conhecimento de câncer de pele na sua família?

( ) sim ( ) não

Você já realizou algum exame dermatológico?

( ) sim ( ) não

Você costuma guardar os seus exames médicos?

( ) sim ( ) não

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

( ) celular ( ) tablet modelo: _____ versão do Android: _____ tela:_____

Quadro 39 – Lista de tarefas a serem executadas pelos avaliadores

LISTA DE TAREFAS

A seguir, é apresentada uma lista com 13 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) Instalação do aplicativo
   Baixe o aplicativo e faça sua instalação.
   A tarefa foi executada? Sim, não? Por quê?
   _____

2) Cadastrar um novo paciente.
   Ao abrir o aplicativo, navegue pela tela inicial e tente cadastrar um paciente.
   A tarefa foi executada? Sim, não? Por quê?
   _____

   Acessando o paciente cadastrado.

   Após o fim do cadastro do paciente, navegue pela tela inicial e acesse o paciente que foi criado na tarefa anterior.
   A tarefa foi executada? Sim, não? Por quê?
   _____

3) Criar o gabarito a ser utilizado na captura das imagens.
   Através do menu lateral do aplicativo, acesse a opção Gabarito e siga as instruções que serão exibidas na tela.
   A tarefa foi executada? Sim, não? Por quê?
   _____
   _____

4) Capturar uma imagem.
   Na tela principal do aplicativo, acesse a opção de capturar imagem. Posicione o gabarito próximo à pinta que você deseja bater a foto e capture a imagem. O aplicativo exibirá uma tela para você escolher se quer salvar ou descartar a imagem. Escolha salvar.
   A tarefa foi executada? Sim, não? Por quê?
   _____
   _____

5) Vincular imagem: seleção da região corporal.
   Após a captura da imagem, o aplicativo exibirá uma tela onde você deverá informar em qual região corporal está a pinta cuja imagem foi capturada.
   A tarefa foi executada? Sim, não? Por quê?
   _____
   _____

6) Vincular imagem: selecionar local específico da pinta em uma região inválida.
Após a escolha da região corporal, o aplicativo exibe desenhos da parte do corpo escolhida. Você deve selecionar um dos desenhos. O aplicativo deve ampliar a parte do corpo selecionada. Tente tocar fora do desenho da parte do corpo, neste momento não deve acontecer nada no aplicativo.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

7) Vincular imagem: selecionar local específico da pinta.
Agora toque dentro do desenho da parte do corpo, no local que deseja vincular a imagem da pinta. O aplicativo deve solicitar a confirmação do local selecionado.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

8) Informar dados sobre a pinta.
Na tela de confirmação do local da pinta, o aplicativo apresenta alguns campos para inserção de informações sobre a pinta: nome para pinta, anotações e classificação de risco. Informe os dados e confirme a ação.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

9) Consultar mapeamento corporal
Na tela principal do aplicativo, acesse a opção de consultar o mapeamento corporal. Navegue até encontrar a parte do corpo com a qual você vinculou a imagem nas tarefas anteriores. Selecione a parte do corpo, o aplicativo deve estar exibindo uma marca no local que você vinculou a imagem, toque sobre esta marca. Deve ser apresentada uma tela com o título Histórico da pinta exibindo a imagem que foi capturada.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

10) Vincular mais de uma imagem ao mesmo grupo (local).
Volte para a tela principal do aplicativo e realize a captura de uma nova imagem da mesma pinta que foi fotografada anteriormente, com o uso do gabarito. Quando o aplicativo apresentar a tela para selecionar o local da pinta, escolha exatamente o mesmo local que escolheu na outra imagem.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

11) Consultar mapeamento corporal com mais de uma imagem da mesma pinta.
Acesse a consulta do mapeamento corporal e selecione a parte do corpo, o aplicativo deve continuar exibindo apenas uma marca no local onde você vinculou as duas imagens capturadas. Toque sobre a marca do local da pinta. Na tela de consulta do histórico você deve ser capaz de ver as duas imagens capturadas, uma ao lado da outra, e também a data e hora do registro e as anotações que foram inseridas.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

12) Visualizar imagens de forma ampliada.
Toque sobre uma das imagens apresentadas. Ela deverá ser ampliada, tente navegar através de gestos na tela para visualizar também a outra imagem de forma ampliada.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

13) Vincular uma imagem a outra região corporal
Em sua última tarefa, você deve ser capaz de capturar uma nova imagem e vincular a mesma a outra parte do corpo. Logo após deve conseguir visualiza-la na consulta do mapeamento corporal.
A tarefa foi executada? Sim, não? Por quê?

_____

_____

Quadro 40 – 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**
Foi fácil encontrar as informações que você precisou					
O design da interface do aplicativo é atraente					
A organização dos menus e comandos de ação (como botões e links) é lógica, permitindo encontrá-los facilmente na tela					
Foi fácil navegar nos menus e telas do aplicativo					
É fácil lembrar como fazer as coisas no aplicativo					
Os símbolos e ícones são claros e intuitivos					
A interface é semelhante dos demais aplicativos					
Você conseguiu compreender que o gabarito serve para acompanhar a evolução (aumento de tamanho) da pinta					
Você se sente mais motivado para fazer o acompanhamento das suas pintas					
É importante que o aplicativo armazene os dados, prevendo a recuperação dos mesmos caso você perca o seu celular					
Você achou importante ver a pinta de forma ampliada					
Você achou importante que o aplicativo mostre a imagem da pinta atual do lado de uma capturada anteriormente para fins de comparação					
Às vezes eu não sei o que fazer com este aplicativo					
Você precisaria do apoio de uma pessoa para usar este aplicativo					
O aplicativo em algum momento parou inesperadamente					
Você usaria novamente o aplicativo caso houvesse alguma falha no primeiro manuseio?					
Você usaria este aplicativo no dia a dia					
Você recomendaria este aplicativo para outras pessoas.					

Qual é a sua opinião sobre o aplicativo quanto ao seu uso e funcionalidades? Fique a vontade para fazer críticas e sugestões.

_____

_____

Obrigado pela sua participação!