INSTITUTO FEDERAL DO ESPÍRITO SANTO
CURSO SUPERIOR DE SISTEMAS DE INFORMAÇÃO

CLAUDIO MAGNO BARBIERI JUNIOR

APLICATIVO ANDROID PARA CORREÇÃO DE FOLHAS DE CARTÃO-RESPOSTA

Serra
2025

CLAUDIO MAGNO BARBIERI JUNIOR

APLICATIVO ANDROID PARA CORREÇÃO DE FOLHAS DE CARTÃO-RESPOSTA

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Sistemas de Informação do Instituto Federal do Espírito Santo, Campus Serra, como requisito parcial para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Daniel Trindade

Serra
2025

Dados Internacionais de Catalogação na Publicação (CIP)

B236a 2025

Barbieri Junior, Claudio Magno
Aplicativo Android para correção de folhas de cartão-resposta / Claudio Magno Barbieri Junior - 2025.
72 f.; il.; 30 cm

Orientador: Prof. Dr. Daniel Trindade.

Monografia (graduação) - Instituto Federal do Espírito Santo, Coordenadoria do Curso de Bacharelado em Sistemas de Informação, 2025.

1. Cartão-resposta. 2. Processamento de imagens. 3. Aplicativos móveis. 4. Android. I. Trindade, Daniel. II. Instituto Federal do Espírito Santo. III. Título.

CDD 004

Bibliotecário: Valmir Oliveira de Aguiar - CRB6/ES 566

CLAUDIO MAGNO BARBIERI JUNIOR

APLICATIVO ANDROID PARA CORREÇÃO DE FOLHAS DE CARTÃO-RESPOSTA

Trabalho de Conclusão de Curso apresentado como parte das atividades para obtenção do título de Bacharel em Sistemas de Informação, do curso de Bacharelado em Sistemas de Informação do Instituto Federal do Espírito Santo.

Aprovado em 07 de fevereiro de 2025.

COMISSÃO EXAMINADORA

Prof. Msc. Daniel Ribeiro Trindade (Orientador)
Instituto Federal do Espírito Santo - Campus Serra

Profª Drª Kelly Assis de Souza Gazolli
Instituto Federal do Espírito Santo - Campus Serra

Prof. Dr. Richard Junior Manuel Godinez Tello
Instituto Federal do Espírito Santo - Campus Serra

FOLHA DE APROVAÇÃO-TCC Nº 1/2025 - SER - CCTMSI (11.02.32.01.08.02.09)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 07/02/2025 16:01)
DANIEL RIBEIRO TRINDADE
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
SER - CCTMSI (11.02.32.01.08.02.09)
Matrícula: 2277933

(Assinado digitalmente em 10/02/2025 17:18)
KELLY ASSIS DE SOUZA GAZOLLI
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
SER-CGEN (11.02.32.01.08.02)
Matrícula: 1344568

(Assinado digitalmente em 07/02/2025 16:43)
RICHARD JUNIOR MANUEL GODINEZ TELLO
PROFESSOR DO ENSINO BASICO TECNICO E TECNOLOGICO
SER-CGEN (11.02.32.01.08.02)
Matrícula: 2280760

*Visualize o documento original em https://sipac.ifes.edu.br/documentos/ informando seu número: **1**, ano: **2025**, tipo: **FOLHA DE APROVAÇÃO-TCC**, data de emissão: **07/02/2025** e o código de verificação: **dafcaef7bd***

Á minha esposa
Aos meus pais
Ao meu irmão
À todos aqueles que me acompanharam nessa jornada

AGRADECIMENTOS

Agradeço a minha família por todo apoio e amor que me deram, por terem me motivado nessa caminhada.

Agradeço ao meu orientador Prof.Dr. Daniel Trindade por toda paciência, dedicação e auxílio.

Por fim, agradeço a todos os colegas e professores que estiveram comigo durante essa jornada no curso.

RESUMO

A utilização de testes objetivos nas avaliações dos alunos é prática comum entre professores de todo o país. Uma das principais características desse tipo de avaliação é ter respostas mais precisas, fazer um julgamento sem subjetividade, além de ser de fácil correção. Porém, quando há uma grande quantidade de testes, há um tempo maior de correção e também uma maior probabilidade de erros por cansaço do examinador. Com o avanço das tecnologias móveis, os dispositivos pessoais têm ganhado cada vez mais destaque e importância no dia a dia das pessoas, utilizando-os para a realização de diversas tarefas diárias. Com base nisso, este trabalho propõe a criação de uma aplicação *móvel* que utiliza a câmera do dispositivo como principal ferramenta para correção de folhas de cartão resposta. Por meio da câmera, o aplicativo utiliza algoritmos e técnicas de processamento digital de imagens para identificar a folha e retirar as questões preenchidas pelo participante da prova. Uma vez reconhecidas as respostas do usuário, é feita uma comparação com a prova cadastrada pelo professor para entrega do resultado da avaliação. Dentre as funcionalidades do aplicativo, duas principais se destacam, sendo elas o cadastro e a correção de provas.

Palavras-chave: Avaliação. Gabarito. Correção. Processamento digital de imagens. Aplicativo. Mobile.

ABSTRACT

The use of objective tests in student assessments is common practice among teachers across the country. One of the main characteristics of this type of assessment is having more precise answers, making a judgment without subjectivity, in addition to being easy to correct. However, when there is a massive amount of tests, we have a longer correction time as well as a greater probability of errors due to examiner fatigue. With the advancement of mobile technologies, personal slides have become increasingly prominent and important in people's daily lives, using them to carry out various daily tasks. Based on this, this work proposes to create a *mobile* application that uses the device's camera as the main tool for correcting template sheets. Through the camera, the application uses algorithms and digital image processing techniques to identify the sheet and remove the questions highlighted by the test participant. Once the user's responses are recognized, a comparison is made with the exam registered by the teacher in order to deliver the evaluation result. Among the application's features, two main ones stand out, being the registration of personalized exams with number of questions, number of question alternatives and final grade.

Keywords: Assessment. Template. Correction. Digital Image Processing. Application. Mobile.

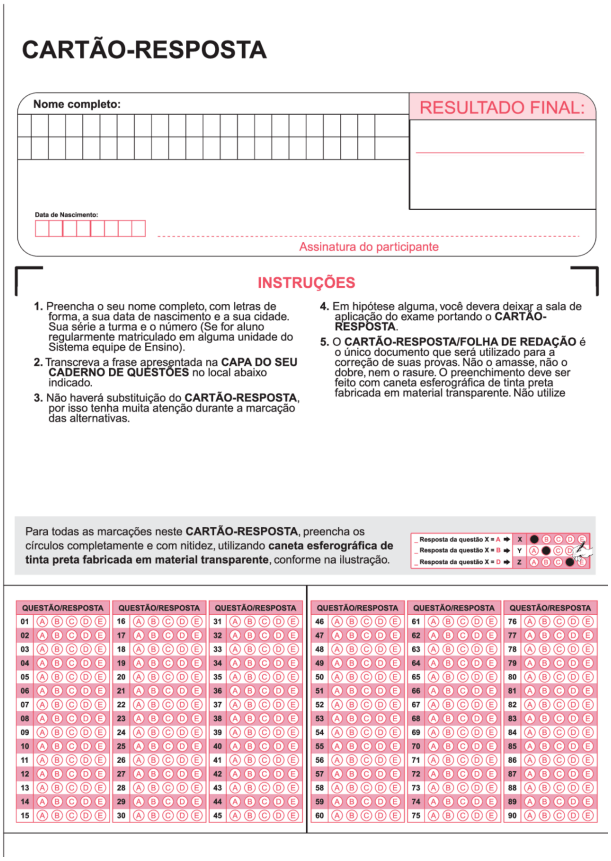
LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Uma das formas utilizadas para facilitar a correção de provas objetivas é a utilização de um modelo de cartão-resposta com as opções dispostas em um formato de múltipla escolha, um exemplo é apresentado na Figura 1. O uso dos cartões-resposta permite que provas sejam corrigidas de maneira mais rápida e precisa, visto que as informações necessárias para a correção estão dispostas de maneira sequencial, o que também minimiza a possibilidade de erros. Por essas vantagens, o uso de cartões-resposta se torna mais justificável em provas com um grande número de participantes, como por exemplo no Enem (Exame Nacional do Ensino Médio).

Figura 1 – Exemplo de cartão-resposta e gabarito.



Fonte: Print (2020).

Em 1937 surgiu o primeiro *hardware* feito pela IBM denominado *Type 805 Test Scoring Machine* que podia realizar a correção de até 750 questões por folha de resposta utilizando comandos elétricos para determinar se a área pré-determinada no cartão-resposta estava pree por grafite, método que seria alterado por volta de 1960 para a utilização da tecnologia OMR (do termo em inglês Optical Mark Recognition) (IBM, 2012). OMR é uma tecnologia que permite a leitura de documentos por meio de marcações utilizando sensores ópticos. O documento é gerado em um modelo padrão que, após ser capturado pelo sensor, é digitalizado, onde, através de um *software*, a imagem é processada e são extraídas

as informações requeridas. As vantagens dos sistemas OMR são a alta velocidade de reconhecimento e menos erros em comparação com a análise tradicional (HUSSMANN; DENG, 2005). Com isso surgiram *softwares* que fazem o uso dessa tecnologia para automatizar a correção de cartões-resposta.

Entretanto, apesar dessas soluções, o uso de um sistema para a correção de cartões-resposta requer um investimento para as instituições de ensino, seja para adquirir o equipamento de *hardware* ou a licença de *software*. Com base nisso, o objetivo desse trabalho é fornecer gratuitamente uma solução em um serviço de distribuição digital, além de deixar o seu código fonte em plataforma de hospedagem de forma pública, possibilitando o amplo acesso por professores e instituições de ensino.

1.1 A PROPOSTA

Uma das soluções para a correção automática de gabaritos é a tecnologia OMR, que permite que os gabaritos sejam digitalizados e, em seguida, corrigidos automaticamente por meio de *software* especializado. No entanto, essa solução requer a compra de equipamentos ou licenças de *software*, o que pode ser um obstáculo para muitas escolas e professores.

Com base nisso, a proposta deste trabalho é a criação de uma aplicação *mobile* para sistema Android disponibilizada de forma gratuita no Play Store e de forma aberta em repositório público, capaz de utilizar a tecnologia OMR em conjunto com a câmera nativa do dispositivo para realizar correções de cartões-resposta, objetivando ser uma solução acessível em termos financeiros e mais prática, não necessitando de equipamentos de *hardware* periféricos.

A aplicação tem como objetivo fornecer ao usuário uma interface interativa para criação de múltiplas folhas de cartões-resposta com diferentes quantidades de questões e valor da prova. Para cada correção feita, a aplicação se propõe a salvar o nome e a nota do exame, através da inserção de dados do usuário. Esses dados poderão ser editados ou excluídos posteriormente.

1.2 OBJETIVOS

1.2.1 Objetivo geral

Desenvolver uma aplicação *mobile* para o sistema Android que permita a correção de cartões-resposta baseados em um gabarito, de forma automatizada, utilizando a câmera, associando cada cartão e seu resultado a um aluno.

1.2.2 Objetivos Específicos

Os objetivos específicos identificados para se atingir o objetivo geral proposto são:

- Criação de *layout* para uma aplicação *mobile*.
- Desenvolvimento de uma aplicação *mobile* para Android com base no *layout* proposto.
- Implementação de um banco de dados nativo para persistência de turmas, alunos e gabaritos criados.
- Implementação de interface que permita a criação de um novo cartão resposta customizável.
- Permitir o acesso ao gabarito criado para posterior impressão.
- Implementação de técnicas de processamento de imagem baseando-se na tecnologia OMR para reconhecimento da folha de cartão resposta por meio da câmera nativa em tempo real.
- Avaliação do gabarito reconhecido, permitindo salvar o resultado vinculado ao aluno e prova correspondentes.

1.3 ESTRUTURA DO TEXTO

Este trabalho está dividido em cinco capítulos. Este primeiro capítulo trouxe uma introdução ao problema estudado, a contextualização do tema, a justificativa para a sua realização e os objetivos pretendidos. O Capítulo 2 apresenta detalhes sobre a tecnologia OMR e como pode ser aplicada, processamento de imagem e desenvolvimento mobile para Android. Em seguida, o Capítulo 3 traz o desenvolvimento da aplicação e o Capítulo 4 traz os resultados dos testes realizados. Por fim, no Capítulo 5, são tecidas as considerações finais e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados conceitos de engenharia de software além das tecnologias, técnicas e bibliotecas usadas neste trabalho, como processamento de imagem, OpenCV e desenvolvimento móvel.

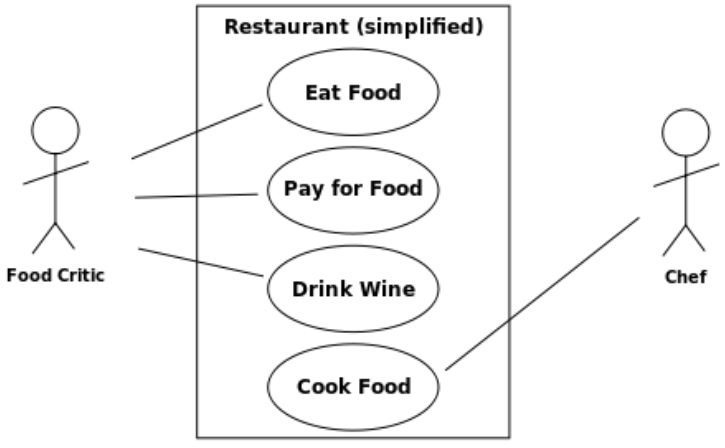
2.1 Engenharia de Software

Engenharia de software é uma disciplina de engenharia cujo foco está em todos os aspectos da produção de software, desde os estágios iniciais da especificação do sistema até sua manutenção, quando o sistema já está sendo usado (SOMMERVILLE, 2011). Neste trabalho serão apresentadas algumas ferramentas de engenharia de software utilizadas para desenvolver a aplicação proposta.

2.1.1 Diagramas de Caso de Uso

Casos de uso são usados para descrever os requisitos externos de um sistema, eles são usados na fase de análise de requisitos de um projeto e contribuem para planos de teste e guias do usuário (FIRESMITH, 2003). Em um caso de uso é representado interatividade do sistema por meio de ações do usuário. No diagrama representamos os atores que realizam ações no sistema representadas por linhas que conectam o autor a ação, exemplificando a interação do usuário com o sistema.

Figura 2 – Diagrama de casos de uso.



Fonte: Wikipedia (2020).

2.1.2 Requisitos Funcionais

Requisitos funcionais são declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações (SOMMERVILLE, 2011). Esses requisitos são essenciais para

definir as expectativas do cliente e orientar o desenvolvimento, fornecendo uma base clara para o design e implementação do software. Ao abordar aspectos como entrada de dados, processamento e saída, os requisitos funcionais garantem que a solução atenda de maneira precisa e eficiente às necessidades do usuário, contribuindo para o sucesso do projeto de engenharia de software.

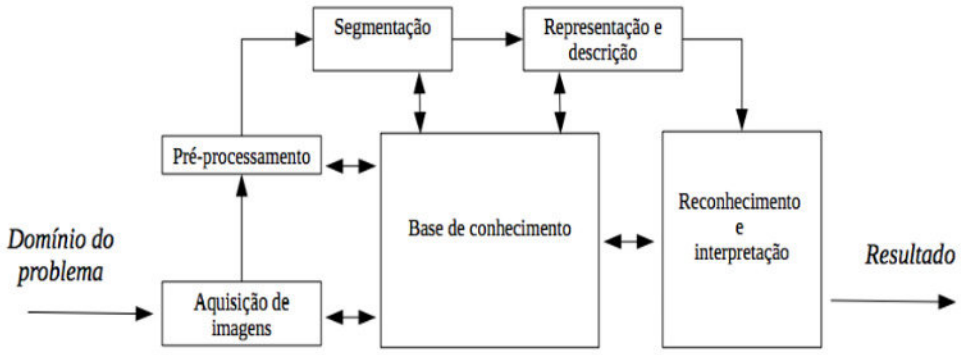
2.1.3 Requisitos Não Funcionais

Requisitos não funcionais são restrições aos serviços ou funções oferecidos pelo sistema. Incluem restrições de *timing*, restrições no processo de desenvolvimento e restrições impostas pelas normas. (SOMMERVILLE, 2011). Esses requisitos são fundamentais para moldar a experiência do usuário e garantir a eficácia do sistema em situações diversas. Exemplos comuns incluem requisitos de escalabilidade, segurança, manutenibilidade e eficiência, fornecendo diretrizes abrangentes que permeiam todo o ciclo de vida do desenvolvimento de software.

2.2 Processamento de Imagem

O processamento de imagem é uma área da ciência da computação que vem crescendo fortemente desde 1964, dedicando-se na resolução de problemas por meio de processamento de imagens digitais. Por meio de várias técnicas e algoritmos, esse método permite a manipulação, transformação e a extração de dados de uma imagem. (GONZALEZ; WOODS, 2000). Essa área engloba uma série de processos, como filtragem, segmentação, reconhecimento de padrões e visão computacional, que têm como objetivo extrair informações relevantes contidas em uma imagem. No contexto deste trabalho, o processamento de imagem desempenha um papel fundamental na análise e tratamento das imagens capturadas, a rotina esperada objetivando o resultado final da avaliação de um cartão resposta está disposta na Figura 3.

Figura 3 – Exemplo fluxo de processamento de imagem.



Fonte: Gate (2023).

Na representação acima, a aquisição da imagem será feita pela câmera de um dispositivo móvel onde cada *frame* será submetido às etapas de avaliação apresentadas no fluxo

objetivando primeiramente o reconhecimento das características pré-determinadas da folha de gabarito. Após a validação positiva, o presente *frame* será enviado à próxima etapa do processamento, onde será realizada a extração da região de interesse possibilitando a análise e interpretação das questões dispostas na folha de resposta. Ao fim dessa etapa, será possível obter o resultado final da avaliação do aluno por meio da contagem das questões preenchidas. Para o auxílio da utilização das técnicas de processamento de imagem, o presente trabalho faz o uso de uma biblioteca *open-source* denominada OpenCV. Nesta seção são apresentados algumas técnicas de processamento de imagem que foram utilizados na aplicação proposta.

2.2.1 Conversão de cor

A conversão de cor é o processo de transformar as informações de cor em uma imagem de um espaço de cores para outro, por meio de algoritmos que calculam a intensidade de cada pixel e modificam o valor de acordo com a transformação pretendida (OPENCV, 2023a). No contexto deste trabalho o algoritmo é utilizado principalmente para converter as imagens obtidas no filtro RGB (*Red, Green e Blue*) para uma escala de cinza, como exemplificado na Figura 4.

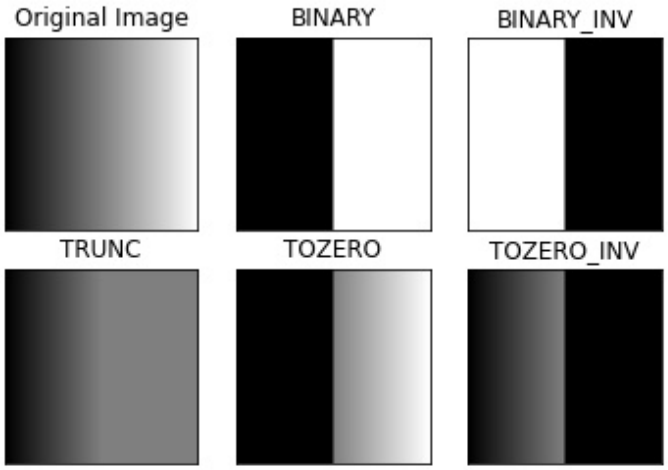
Figura 4 – Exemplo conversão de RGB para escala de cinza.

Fonte: Lindevs (2023)

2.2.2 Binarização

O processo de binarização também conhecido como **thresholding** faz a classificação binária dos *pixels* de uma imagem (OPENCV, 2023d). Nesse algoritmo, cada pixel é comparado a um valor de intensidade definido de 0 a 255, onde cada valor abaixo é convertido para 0 e cada valor maior ou igual ao valor de intensidade é convertido para o valor 1. Esse algoritmo pode ser utilizado em diversas aplicações tais como, detecção de bordas, classificação de objetos, reconhecimento de caracteres e muito mais. A função **thresholding** possui diversos métodos para avaliação e segmentação da imagem que podem ser usados de formas distintas a depender do resultado esperado, conforme apresentado na Figura 5.

Figura 5 – Exemplo de métodos de conversão do threshold.



Fonte: Documentation (2023a)

2.2.2.1 Método Otsu

O método otsu é um método binarização que faz a classificação binária dos *pixels* de uma imagem porém diferente do método **thresholding** apresentados anteriormente, esse método tem como objeto de facilitar a busca de uma valor de intensidade, também conhecido como limiar (OPENCV, 2023d). O método otsu através de uma imagem em escala de cinza reconhece todos os limiares e por meio deles realiza um cálculo para obter o valor de limiar que contém a menor variação, maximizando a assertividade no processo de binarização da imagem.

2.2.3 Morfologia matemática

A morfologia matemática é uma técnica de processamento de imagem baseada em operações denominadas: dilatação, erosão, abertura e fechamento, sendo utilizada geralmente em imagens binárias (OPENCV, 2023e). Esse algoritmo tem o objetivo de realizar a limpeza de ruídos na imagem percorrendo as formas e realizando operações com base em um valor pré definido denominado *kernel*, onde a cada conjunto de *pixel* a depender do operador utilizado será realçado ou desvalorizado convertendo os valores para 0 ou 1. No presente trabalho é utilizada a operação de fechamento onde a imagem é submetida a dois processos em sequência, dilatação e erosão. A operação morfológica denominada dilatação tem por objetivo realçar as bordas de um objeto, o funcionamento com base em uma área pré-definida (*kernel*) converterá os valores de pixel para 1 se ao menos um do conjunto de *pixels* ao redor for igual a 1. A operação morfológica denominada erosão tem por objetivo desvalorizar as bordas de um objeto, o funcionamento com base em uma área pré-definida (*kernel*) converterá os valores de pixel para 1 apenas se todo o conjunto de *pixels* ao redor for igual a 1. Ao fim, o transformação morfológica de fechamento tem por

objetivo diminuir o ruído da imagem como demonstrado na Figura 6.

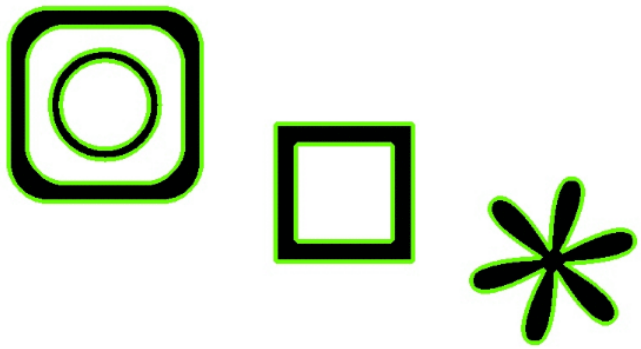
Figura 6 – Exemplo da utilização da transformação morfológica de fechamento.

Fonte: Documentation (2023b)

2.2.4 Região Conectada

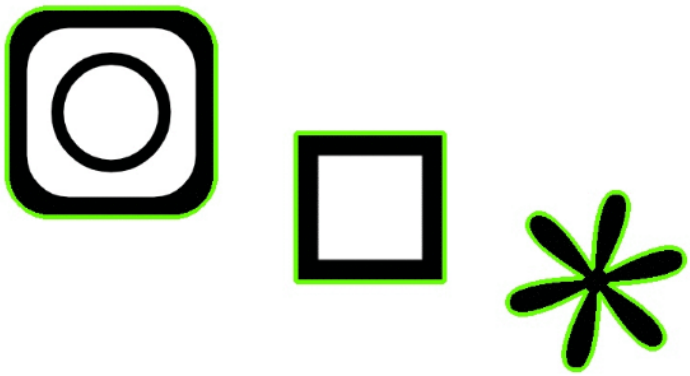
Os algoritmos de região conectada são projetados para identificar e agrupar conjuntos de *pixels* em sequência com cor e intensidade semelhantes obtendo objetos na imagem, chamados de contornos (OPENCV, 2023b). Tais algorítimos oferecem diferentes modos de detecção de contornos, como detecção de contornos externos, contornos internos ou todos os contornos presentes na imagem. No contexto deste trabalho, é utilizado um algoritmo para retornar todos os contornos presentes na imagem conforme ilustrado na Figura 7 e os contornos externos conforme ilustrado na Figura 8.

Figura 7 – Exemplo da detecção de todos os contornos.



Fonte: Shaikh (2020a)

Figura 8 – Exemplo da detecção de contornos externos.

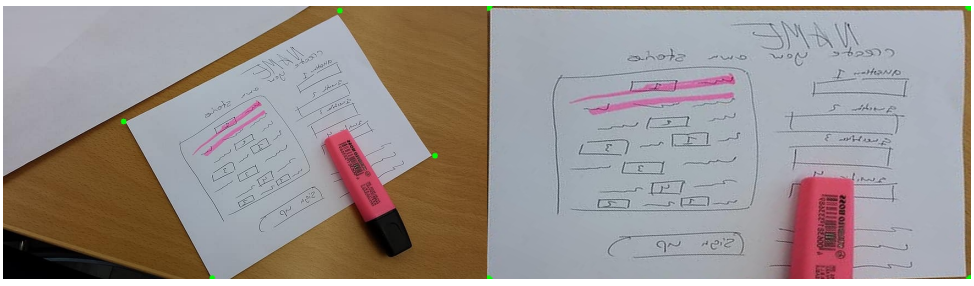


Fonte: Shaikh (2020a)

2.2.5 Transformação de Perspectiva

A transformação de perspectiva é uma técnica que permite ajustar a representação visual de uma imagem, alterando sua perspectiva. Ao selecionar pontos-chave em uma imagem original e associá-los a posições correspondentes em uma imagem transformada, uma matriz é calculada para modelar a transformação geométrica necessária (OPENCV, 2023c). Essa matriz é então aplicada a cada pixel da imagem original, reposicionando-os para criar uma nova visualização com perspectiva modificada, conforme apresentado na Figura 9. Essa técnica é frequentemente utilizada para corrigir distorções em imagens causadas por ângulos de visão não ideais. A transformação de perspectiva desempenha um papel fundamental em diversas aplicações, desde a correção de imagens de câmeras inclinadas até a projeção de objetos virtuais em ambientes tridimensionais.

Figura 9 – Exemplo de transformação de perspectiva.



Fonte: Shaikh (2020b)

2.3 OpenCV

A biblioteca OpenCV[1] (*Open Source Computer Vision Library*) é uma ferramenta amplamente utilizada no desenvolvimento de aplicações que utilizam como recurso o processamento de imagem. Por meio dessa, é possível ter acesso a uma gama de ferramentas entre algoritmos e funções especializadas em análise e processamento de imagens capazes de

[1] https://opencv.org/

realizar tarefas em tempo real. A biblioteca é distribuída gratuitamente, foi desenvolvida pela Intel e possui mais de 500 funções e algoritmo s (MARENGONI; STRINGHINI, 2009). Neste trabalho foi utilizado a versão 4.8.0 da biblioteca OpenCV.

2.4 Desenvolvimento Móvel

O desenvolvimento móvel refere-se ao processo de criação de aplicativos destinados a dispositivos móveis, como *smartphones* e *tablets*. Essa área da engenharia de software abrange tanto o desenvolvimento de aplicativos nativos, projetados para plataformas específicas como Android ou iOS, quanto o desenvolvimento de aplicativos multiplataforma, que podem ser executados em diferentes sistemas operacionais utilizando *frameworks* como React Native ou Flutter. Com a crescente uso dos dispositivos móveis, o desenvolvimento móvel tornou-se essencial para atender às demandas da sociedade contemporânea. Ele envolve uma combinação de habilidades em programação, design de interface do usuário, experiência do usuário, testes e integração com serviços em nuvem. O ecossistema móvel dinâmico exige uma abordagem adaptativa e ágil, considerando as constantes atualizações de sistemas operacionais e a diversidade de dispositivos. O desenvolvimento móvel desempenha um papel crucial na criação de aplicativos inovadores que aprimoram a produtividade, fornecem entretenimento, e facilitam a comunicação e o acesso à informação em qualquer lugar e a qualquer momento. No contexto do trabalho, foi escolhido o desenvolvimento nativo para Android para desenvolver a aplicação, utilizando a linguagem Kotlin adotada como linguagem oficial para desenvolvimento Android pela Google desde 2017.

2.5 TRABALHOS RELACIONADOS

Um trabalho relacionado que possui objetivo semelhante ao proposto foi o TCC de Silva (2018), onde foi desenvolvido um sistema chamado Minha Prova que possui uma interface web onde é realizado o cadastro de docentes, turmas e provas que podem ser consultadas pelos sistemas *mobile* através de uma API. Como solução para a correção, o trabalho faz o uso de uma aplicação Android desenvolvida em Java que faz o uso da câmera em conjunto com a biblioteca OpenCV para a análise e processamento de imagem, além de um outra aplicação feita para estudantes desenvolvida em *React Native* para dar suporte aos sistemas operacionais Android/iOS que possibilita a visualização dos detalhes das provas de cada aluno individualmente.

O sistema Minha Prova possui o processo criação de provas separado do sistema *mobile* que faz a correção, além de ser pago possuindo um valor de mensalidade no plano para instituições de ensino e anuais ou semestrais para professores independentes.

Outra solução se trata do aplicativo EvalBee, que possui todas as funcionalidades principais

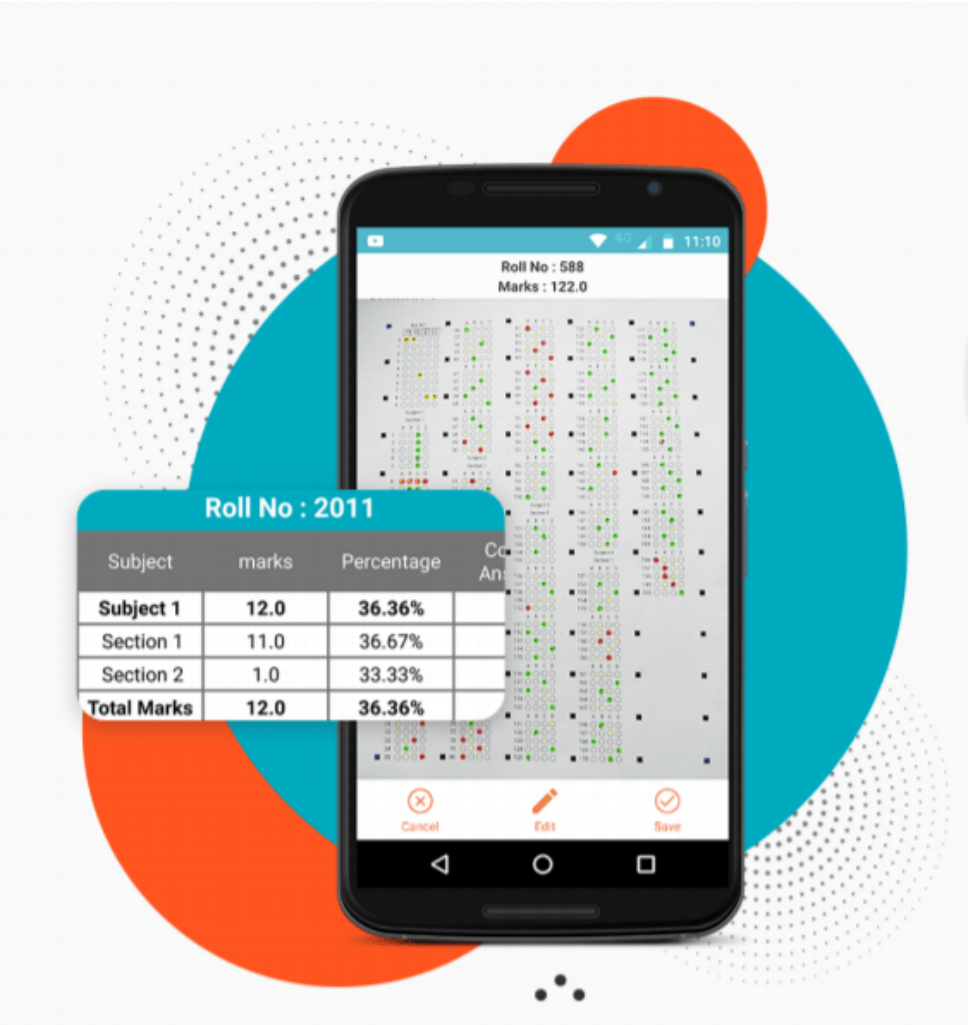
Figura 10 – Minha Prova.

Fonte: https://www.minhaprova.com.br/ (2023)

do projeto proposto como cadastro de alunos e turmas, criação de folhas de cartão resposta e correção automatizada, sendo possível também customizar o cartão resposta com diferentes seções de questões que podem ter quantidade de questões e valores diferentes. Desenvolvido para Android e para o iOS o aplicativo possui diferentes planos de uso que variam de licença gratuitas onde usuário está sujeito a vídeos promocionais e até empresariais onde possui a possibilidade de uso de vários professores na mesma conta.

Por se tratar de uma aplicação mobile com uma versão desenvolvida nativamente para Android o aplicativo EvalBee é uma das principais inspirações do presente trabalho, focando em uma solução de uso gratuito sem anúncios promocionais.

Figura 11 – EvalBee.



Fonte: https://evalbee.com/ (2023)

3 DESENVOLVIMENTO

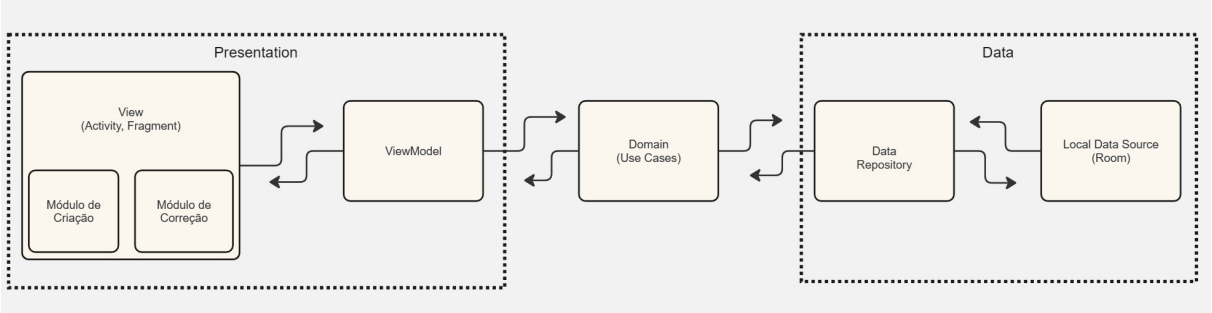
Neste trabalho foi desenvolvida uma aplicação nativa para Android para criação e correção de folhas de cartão-resposta, cujo o objeto é prover uma forma simples e gratuita de automatização para correção de exames escolares. A aplicação foi desenvolvida com foco nos docentes das instituições de ensino fornecendo interfaces customizadas de criação de folhas de cartão-resposta, além de opções para correção através da câmera do dispositivo utilizando técnicas de processamento de imagem, cadastro de alunos e geração de um arquivo de imagem da folha de respostas para eventual impressão.

O presente trabalho possui como fonte de inspiração a aplicação *mobile* EvalBee, por ter uma boa avaliação nas lojas de aplicativos e ser uma aplicação *mobile* com versão nativa para Android. A aplicação construída neste trabalho foi desenvolvida utilizando a linguagem de programação Kotlin em conjunto com o Android SDK (*Software Development Kit*), além de utilizar a biblioteca OpenCV para o processamento de imagem. As próximas seções mostram as metodologias utilizadas na construção das diferentes partes do sistema proposto.

3.1 Arquitetura Geral da Aplicação

Na Figura 12 é apresentada a arquitetura geral da aplicação e como essa foi construída utilizando conceitos nativos de arquitetura Android, objetivando a separação do código em camadas deixando-o mais legível e fácil de ser testado (DEVELOPER, 2023c).

Figura 12 – Arquitetura geral da aplicação.



Fonte: Imagem elaborada pelo autor.

Cada camada segue o principio de responsabilidade única, sendo independente umas das outras. As seções a seguir descrevem cada uma delas:

1. *Data* é a parte responsável por realizar conexões com partes externas ao app como por exemplo comunicação com banco de dados e API's (*Application Programming Interface*) através dos repositórios. Os repositórios são classes que centralizam o acesso a uma fonte de dados, funcionando como uma ponte entre a fonte externa de dados e a aplicação. Ela contém a lógica de negócio e entrega os dados mapeados para a

aplicação (DEVELOPER, 2023b). No aplicativo essa camada é utilizada apenas para a comunicação com o banco de dados, sendo executada para acessar e salvar os dados. Na Figura 12 é indicado na camada *data* a comunicação do repositório (*data repository*) com o banco local (*local data source*).

2. *Domain* é a parte responsável pela transposição dos dados para um modelo de domínio, facilitando a manipulação dos objetos no projeto. Essa camada tem como principal função executar lógicas de chamadas da camada de data e realizar o mapeamento dos dados de retorno para um objeto de domínio. Essas chamadas são feitas por *use cases* que armazenam toda a lógica para realizar as chamadas e mapear o retorno.

3. *Presentation* é a parte responsável por toda a apresentação das interfaces visuais (*views*) do aplicativo, além da lógica de visualização. Essa camada é onde acessamos todos os recursos visuais e coletamos os dados de *input* do usuário. As *views* são resposáveis pelas interfaces gráficas demonstradas no app enquanto as *view models* são as classes responsáveis por conter toda a lógica relativa a interface e, caso necessário, realizar a comunicação com a camada de domínio. É nessa camada que temos demonstradas as principais funcionalidades do app:

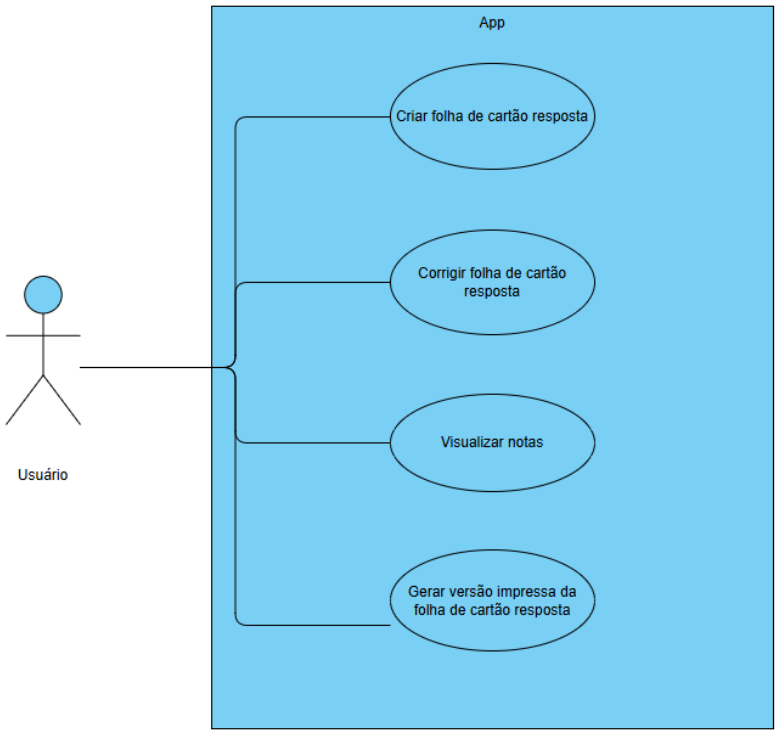
   a) Módulo de Correção é o módulo responsável por realizar a correção da folha de cartão-resposta onde a aplicação extrai a informação da câmera do dispositivo e realiza a comparação com os dados salvos para corrigir a prova.

   b) Módulo de Criação é o módulo responsável por criar a folha de cartão-resposta através da manipulação de um arquivo XML (*Extensible Markup Language*) de acordo com as informações cadastradas pelo usuário no momento de criação da prova.

3.2 Diagrama de Caso de Uso

Além da disposição dos elementos que compõem o sistema, para o seu desenvolvimento foram desenhados alguns cenários utilizando diagramas de caso de uso representando possíveis ações do usuário ao utilizar a aplicação.

Figura 13 – Casos de uso.



Fonte: Imagem elaborada pelo autor.

3.3 Requisitos

Nessa seção são apresentados os requisitos funcionais e não funcionais da aplicação. As tabelas possuem 3 colunas: Requisito, Descrição e Prioridade. A coluna Prioridade pode conter duas classificações possíveis: *Should*, que demonstra um requisito que a aplicação poderia ter; e *Must*, que demonstra os requisitos que a aplicação deve conter.

Tabela 1 – Tabela de Requisitos Funcionais

Requisito	**Descrição**	**Prioridade**
RF01 - Tela de câmera	O aplicativo deve ter uma tela de câmera.	Must
RF02 - Tela de criação de prova	O aplicativo deve ter uma tela para criar uma nova prova contendo os campos necessários como título, descrição, quantidade de opções, quantidade de questões e nota máxima	Must
RF03 - Tela de questões corretas	O aplicativo deve ter uma tela para selecionar as opções corretas para cada questão	Must
RF04 - Tela de provas cadastradas	O aplicativo deve ter uma tela para buscar as provas cadastradas	Must
RF05 - Tela de prova	O aplicativo deve ter uma tela com as informações de prova cadastrada Provas corrigidas	Must
RF06 - Imprimir prova	O aplicativo deve ter a funcionalidade de imprimir a prova cadastrada	Must
RF07 - Excluir prova	O aplicativo deve permitir a exclusão de prova	Must
RF08 - Tela de correção	O aplicativo deve ter uma tela com os detalhes da correção	Must
RF09 - Salvar correção	O aplicativo deve permitir salvar correção	Must
RF10 - Editar correção	O aplicativo deve permitir alterar o nome e a nota do exame corrigido	Must
RF11 - Editar prova criada	O aplicativo deve permitir editar uma prova criada	Should
RF12 - Prova com seção	O aplicativo deve permitir separar as questões por seção	Should
RF13 - Notas por seção	O aplicativo deve permitir atribuir notas diferentes para cada seção	Should

Fonte: Próprio autor.

3.4 Requisitos Não Funcionais

Tabela 2 – Tabela de Requisitos Não Funcionais

Requisito	**Descrição**	**Prioridade**
Desempenho	O aplicativo deve ter uma performance aceitavel independente da versão do dispositivo	Must
Câmera	O aplicativo deve ter acesso a câmera do dispositivo	Must
Plataforma	O aplicativo deve possuir uma versão para Android e para iOS	Should

Fonte: Próprio autor.

3.5 Tecnologias Utilizadas

Nessa seção são apresentadas as tecnologias utilizadas para o desenvolvimento da aplicação.

3.5.1 Android SDK

O Android SDK (*Software Development Kit*) é um conjunto de ferramentas e recursos para o desenvolvimento de aplicativos para o sistema operacional Android. Composto por uma variedade de componentes, como emuladores, depuradores e bibliotecas, o SDK oferece um ambiente de desenvolvimento com várias ferramentas que auxiliam a construção de um projeto Android. Ele fornece aos desenvolvedores a capacidade de construir aplicativos inovadores e funcionais, aproveitando ao máximo os recursos oferecidos pela plataforma Android, desde a criação de interfaces de usuário intuitivas até a integração de funcionalidades avançadas, garantindo uma excelente experiência para os usuários (DEVELOPER, 2023a). Para a aplicação construída neste trabalho foi utilizada a versão do Android Sdk 34.

3.5.2 ROOM

ROOM é uma biblioteca de persistência de dados para aplicativos Android. Desenvolvida pelo Google, ela simplifica o processo de armazenamento e acesso a dados locais em bancos de dados SQLite. Ao adotar uma abordagem baseada em anotações, o ROOM elimina a necessidade de escrever código repetitivo, permitindo que os desenvolvedores definam facilmente entidades de dados, consultas e manipulação de dados com menos esforço. Além disso, o ROOM oferece suporte a conceitos fundamentais como relacionamentos entre entidades, consultas assíncronas e manipulação de transações, garantindo uma interação suave e eficiente com o banco de dados (DEVELOPER, 2023d). Para a aplicação construída neste trabalho foi utilizada a versão 2.5.2.

Listing 3.1 – Exam

```
1     @Entity
2     data class Exam(
3         @PrimaryKey(autoGenerate = true)
4         val id: Int = 0,
5         val title: String,
6         val description: String,
7         val options: Int,
8         val questions: Int,
9         val examAnswers: List<Int>,
10        val maxScore: Int
11    )
```

Em 3.1 temos um exemplo de uma classe utilizando a linguagem Kotlin. A classe é anotada com a *key-word* @Entity na linha 1, que sinaliza ao Room que essa classe será transformada em uma tabela de dados. Na linha 3 temos a anotação @PrimaryKey que sinaliza que o campo id da classe será a chave primária da tabela.

Listing 3.2 – ExamDao

```
1  @Dao
2  interface ExamDao {
3
4      @Query("SELECT * FROM exam")
5      fun getAllExams(): List<Exam>
6
7      @Insert(onConflict = OnConflictStrategy.REPLACE)
8      suspend fun saveExam(exam: Exam)
9
10     @Query("DELETE FROM exam where id =:examId")
11     fun deleteExamById(examId : Int)
12
13 }
```

Em 3.2 temos uma interface com a anotação @Dao na linha 1, que sinaliza ao Room que essa interface será convertida para uma objeto de acesso as tabelas. Dentro da interface temos declaradas as funções de consultas, anotadas com a nomenclatura padrão de uma consulta sql, sendo @Query na linha 4 que faz uma consulta de todos os dados cadastrados nessa tabela, @Insert na linha 7 responsável por inserir um novo exame na tabela e uma outra @Query na linha 10 com o comanda para remover um exame que contém um id específico.

3.5.3 Dagger Hilt

A biblioteca Dagger Hilt é uma extensão simplificada e facilitadora para realizar a injeção de dependência nos projetos Android. O princípio da injeção de dependência desempenha

um papel crucial na programação orientada a objetos. Em vez de um módulo criar suas próprias dependências, elas são fornecidas a ele de forma externa, comumente por meio de construtores ou métodos de configuração. Essa abordagem estimula a divisão do código em partes distintas, proporcionando maior flexibilidade e facilitando a troca de componentes, o que torna o código mais compreensível, fácil de testar e de dar manutenção.

Desenvolvido pelo Google em colaboração com a comunidade, o Dagger Hilt foi projetado para simplificar o processo de configuração e utilização da injeção de dependência em projetos Android. Ao fornecer uma série de anotações e classes pré-definidas, o Hilt automatiza tarefas comuns, como a criação de componentes e a definição de escopos de dependência, reduzindo assim a quantidade de código necessário e melhorando a legibilidade e manutenibilidade do código (DEVELOPER, 2024a). Para a aplicação construída nesse trabalho foi utilizado a versão do Dagger Hilt 2.47.

Listing 3.3 – DatabaseDi

```
1  @Module
2  @InstallIn(SingletonComponent::class)
3  object DatabaseDI {
4      .
5      .
6      .
7
8      @Provides
9      @Singleton
10     fun providesExamDao(appDatabase: AppDatabase): ExamDao = appDatabase.
           examDao()
11
12     .
13     .
14     .
15 }
```

Em 3.3 temos a criação de um componente onde declaramos as classes que serão injetadas no projeto, sendo usado como referência a criação do objeto ExamDao demonstrado em 3.2. Na linha 1 temos a anotação @Module sinalizando que essa classe é um objeto que será utilizado pelo Hilt para criar as dependências. Na linha 2 temos a anotação @InstallIn(SingletonComponent::class) que sinaliza para o Hilt em qual módulo da aplicação será criado as dependências, nesse caso a classe interna do Hilt SingletonComponent informa que esse objeto poderá ser acessado por toda a aplicação. Por fim, nas linhas 8 e 9 temos o @Provides e @Singleton, que dizem que a função abaixo irá criar e fornecer a classe ExamDao e que essa será uma instância única para toda a aplicação.

3.5.4 Data Binding

A biblioteca Data Binding é uma ferramenta oferecida para Android que simplifica o desenvolvimento de interfaces de usuário dinâmicas em aplicativos. Com o Data Binding, os desenvolvedores podem vincular componentes da interface do usuário diretamente aos dados do aplicativo, eliminando a necessidade de atualizações manuais repetitivas. Essa abordagem permite uma separação mais clara entre a lógica de negócios e a camada de apresentação, resultando em um código mais limpo, conciso e fácil de manter. Além disso, o Data Binding oferece recursos avançados, como vinculação bidirecional de dados, expressões observáveis e conversores personalizados, proporcionando uma experiência de desenvolvimento mais eficiente e produtiva (DEVELOPER, 2024b). Para a aplicação construída nesse trabalho foi utilizado a versão do Data Binding 8.1.0.

3.6 Desenvolvimento

Nesta seção será demonstrado como o foi o processo de desenvolvimento dos dois principais módulos da aplicação. Inicialmente, é discutido como o padrão XML é utlizado pelo Android para definição de interfaces de usuário.

3.6.1 XML

O XML (*Linguagem de Marcação Extensível*) é uma linguagem amplamente adotada na plataforma Android para definir layouts de interface do usuário. No contexto do desenvolvimento para Android, os arquivos XML desempenham um papel fundamental na especificação da estrutura e do comportamento de itens visuais, como botões, textos e contêineres. Através de sua sintaxe intuitiva e hierárquica, conseguimos definir como os itens devem ser organizados e configurar propriedades como dimensão, aparência e interatividade. Adicionalmente, o XML para *Android* é altamente flexível e extensível, permitindo criar interfaces complexas e definir recursos reutilizáveis, o que apoia a modularidade e a manutenibilidade do código. No trecho de código 3.4, há um exemplo de XML para definição de interface de usuário em Android.

Listing 3.4 – XML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:orientation="vertical">
8
9      <com.google.android.material.appbar.AppBarLayout
10         android:id="@+id/appBarLayout"
```

```
11          android:layout_width="match_parent"
12          android:layout_height="wrap_content"
13          android:background="@android:color/transparent"
14          app:elevation="0dp"
15          app:layout_constraintEnd_toEndOf="parent"
16          app:layout_constraintStart_toStartOf="parent"
17          app:layout_constraintTop_toTopOf="parent">
18
19          <com.google.android.material.appbar.MaterialToolbar
20              android:id="@+id/toolbar"
21              android:layout_width="match_parent"
22              android:layout_height="wrap_content"
23              android:textAlignment="center"
24              app:navigationIcon="@drawable/arrow_back"
25              app:title="@string/title_my_exams"
26              app:titleCentered="true"
27              app:titleTextAppearance="@style/Toolbar.TitleText"
28              app:titleTextColor="@android:color/white" />
29      </com.google.android.material.appbar.AppBarLayout>
30
31      <ScrollView
32          android:layout_width="match_parent"
33          android:layout_height="match_parent"
34          android:fillViewport="true">
35
36          <LinearLayout
37              android:layout_width="match_parent"
38              android:layout_height="wrap_content"
39              android:orientation="vertical">
40
41              <TextView
42                  android:id="@+id/examTitle"
43                  android:layout_width="match_parent"
44                  android:layout_height="wrap_content"
45                  android:layout_marginHorizontal="8dp"
46                  android:layout_marginTop="8dp"
47                  android:ellipsize="end"
48                  android:gravity="center"
49                  android:maxLines="1"
50                  android:textColor="@android:color/white"
51                  android:textSize="20sp"
52                  android:textStyle="bold"
53                  tools:text="Provas" />
54
55              <EditText
56                  android:background="@drawable/bg_white_16dp"
57                  android:id="@+id/searchView"
58                  android:layout_width="match_parent"
```

```
59                  android:hint="  Buscar prova"
60                  android:textColorHint="@color/primaryColor"
61                  android:layout_height="50dp"
62                  android:layout_marginHorizontal="16dp"
63                  android:layout_marginTop="16dp"
64                  android:theme="@style/SearchViewTheme" />
65
66              <androidx.recyclerview.widget.RecyclerView
67                  android:id="@+id/examsRv"
68                  android:layout_width="match_parent"
69                  android:layout_height="wrap_content"
70                  android:layout_marginHorizontal="16dp"
71                  android:layout_marginTop="16dp"
72                  android:clipToPadding="false"
73                  android:paddingVertical="16dp"
74                  app:layoutManager="androidx.recyclerview.widget.
                        LinearLayoutManager"
75                  tools:listitem="@layout/item_exam" />
76
77          </LinearLayout>
78      </ScrollView>
79  </LinearLayout>
```

1. *TextView* é um componente usado para exibir texto na interface do usuário de um aplicativo. Este componente pode ser facilmente personalizado, atribuindo cores, tamanhos e tipos de fonte diferente. Em 3.4 temos a criação de um componente *TextView* da linha 41 até a linha 53.

2. *EditText* é um componente de interface do usuário que permite aos usuários inserir e editar texto. Ele é frequentemente utilizado em formulários e telas de entrada de dados, oferecendo uma ampla gama de recursos para interação com o usuário. Além de aceitar entrada de texto simples, o EditText suporta diferentes tipos de entrada, como números, senhas e endereços de e-mail, entre outras. Em 3.4 temos a criação de um componente *EditText* da linha 55 até a linha 64.

3. *View* é um componente fundamental no desenvolvimento de aplicativos Android, que serve como base para a criação de componentes de interface do usuário. Ela representa um retângulo na tela e é responsável por desenhar e manipular elementos visuais, podendo ser um botão, um campo de texto, uma imagem ou qualquer outro elemento com o qual o usuário pode interagir.

4. *RecyclerView* é uma componente-chave no desenvolvimento Android para exibir grandes conjuntos de dados de maneira eficiente e escalável. Ele oferece uma abordagem flexível

e otimizada para lidar com listas e grades de itens. Esse componente possui um modelo de adaptador que permite que os desenvolvedores forneçam dinamicamente os dados a serem exibidos e personalizem a aparência e o comportamento de cada item. Além disso, ele aproveita ao máximo a técnica de reciclagem de *Views*, reutilizando elementos visuais fora da tela conforme o usuário rola, resultando em uma experiência de rolagem suave e economia de recursos. Em 3.4 temos a criação de um componente *EditText* da linha 66 até a linha 75.

5. *Button* é um dos elementos fundamentais da interface do usuário em aplicativos Android, permitindo aos usuários interagir com o aplicativo por meio de cliques ou toques. Ele é usado para desencadear ações, como enviar um formulário, fazer uma seleção ou iniciar uma operação específica

Figura 14 – Layout.

Fonte: Imagem elaborada pelo autor.

Na Figura 14 temos o layout gerado pelo código apresentado em 3.4.

3.6.2 Módulo de Criação

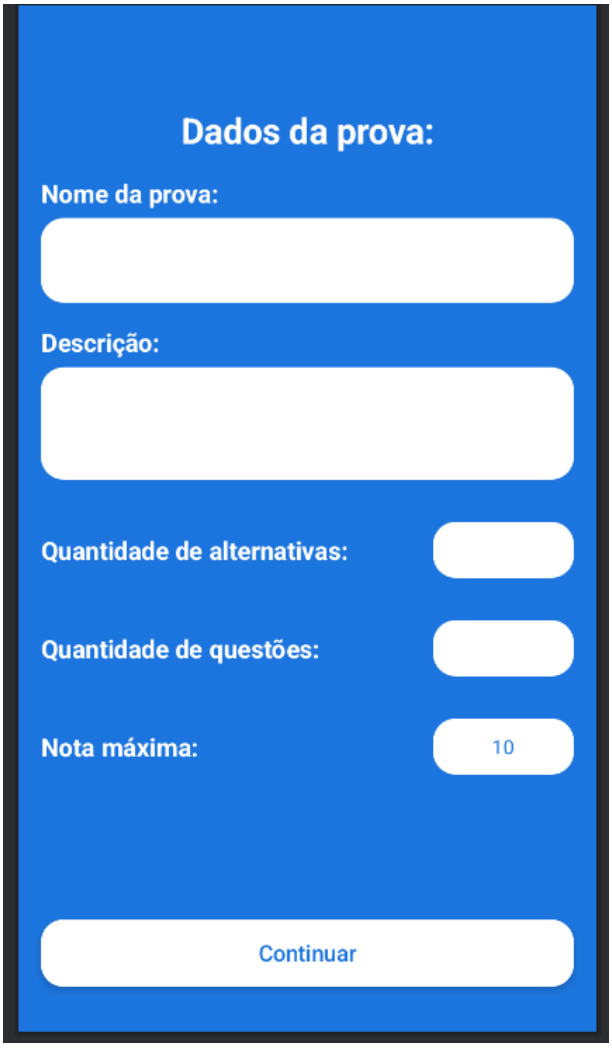
O módulo de criação é a parte da aplicação onde o usuário faz o preenchimento do formulário objetivando a criação de uma folha de cartão-resposta de acordo com as informações inseridas. É nessa parte onde a aplicação realiza o salvamento dos dados preenchidos no banco de dados local utilizando o *framework* ROOM. Esse módulo é parte essencial para o funcionamento correto do módulo de correção, o qual realiza a comparação da folha de acordo com os dados preenchidos pelo usuário.

O formulário de criação é baseado em duas principais telas, sendo elas:

3.6.2.1 Dados da prova

Nessa tela, mostrada na Figura 15, a aplicação cria um formulário utilizando o componente nativo *EditText* para permitir a inserção de dados do usuário, são eles: nome, descrição, quantidade de questões, quantidade de alternativas por questão e valor máximo da avaliação.

Figura 15 – Criação de prova passo 1.

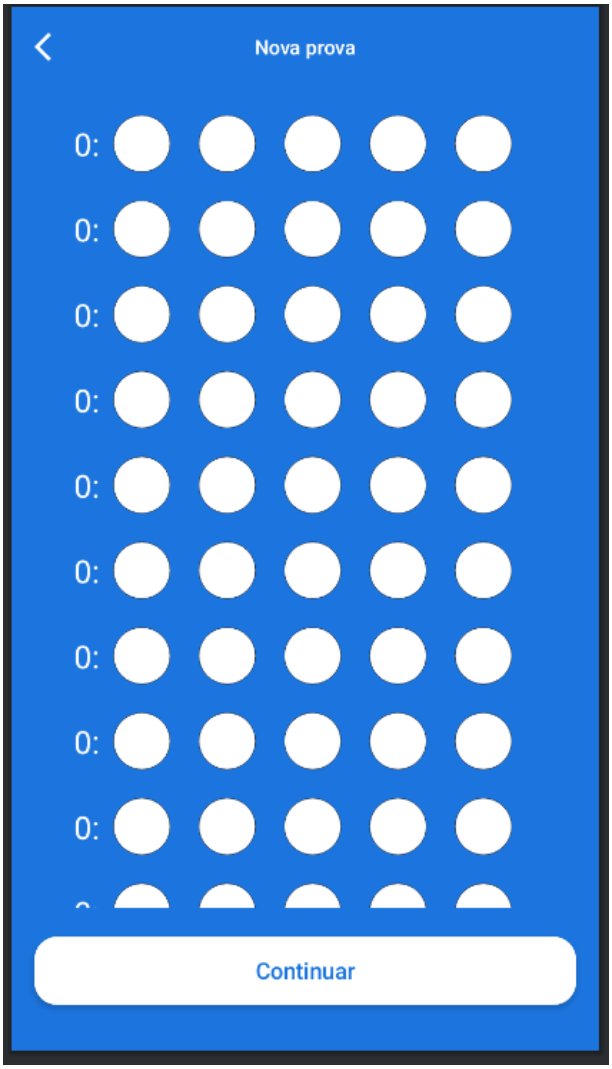


Fonte: Imagem elaborada pelo autor.

3.6.2.2 Questões corretas

Nesta tela, demonstrada na Figura 16, a aplicação usa o componente nativo *RecyclerView* para criar uma lista com quantidade de questões assinaladas pelo usuário na tela anterior. A partir dessa lista, cada linha irá possuir uma outra lista (*RecylerView*) com a quantidade de alternativas escolhidas pelo usuário, sendo cada alternativa uma *View* permitindo a interação ao toque. Com a interface da tela pronta, o usuário fará o preenchimento através do clique em cada alternativa indicando qual será a opção correta.

Figura 16 – Criação de prova passo 2.



Fonte: Imagem elaborada pelo autor.

Ao final do fluxo, a aplicação realiza o recolhimento dos dados preenchidos pelo usuário através da biblioteca *data binding*. Com os dados, a biblioteca *ROOM* é acionada, criando para o exame uma tabela na base de dados com os seguintes atributos:

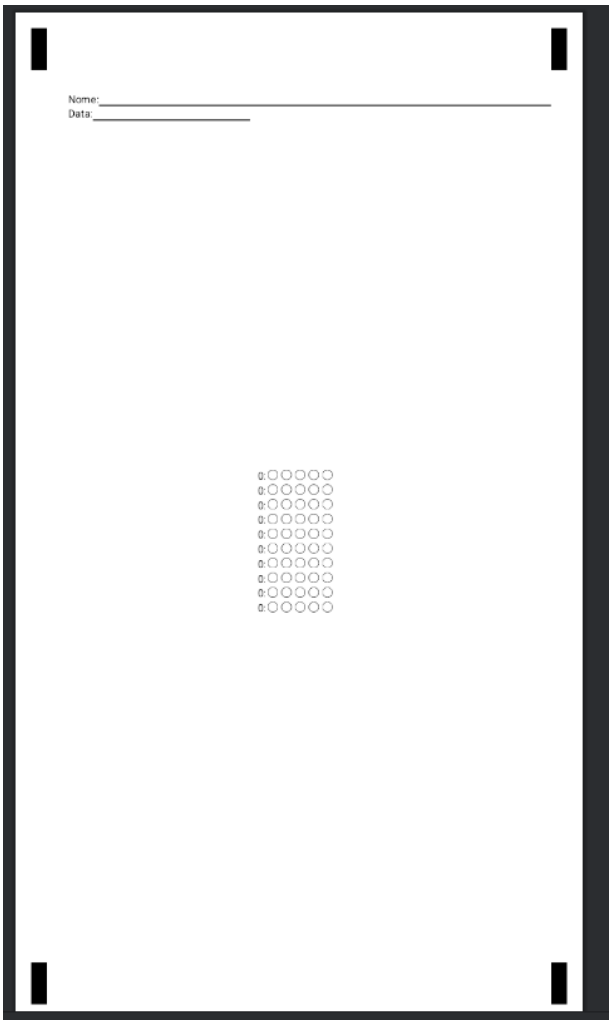
Tabela 3 – Dados de um exame

Atributo	**Descrição**	**Tipo**
id	Identificador único de cada exame cadastrado	Int
title	Nome do exame	String
description	Curta descrição do exame	String
options	Quantidade de alternativas por questão	Int
questions	Quantidade de questões	Int
examAnswers	Uma lista de inteiros, cada posição da lista representa a opção correta para a questão de igual posição	List<Int>
maxScore	Valor máximo do exame	Int

Fonte: Próprio autor.

Após o exame ser criado na base de dados, o usuário consegue gerar um arquivo PDF (*Portable Document Format*) da folha de cartão-resposta. O arquivo é gerado através da manipulação de um arquivo de layout XML contendo os elementos dimensionados para um tamanho de folha A4. Esse arquivo de layout é renderizado pelo sistema operacional criando um objeto denominado *view*, que fica armazenado em memória. Após esse processo, os elementos da *view* são inicializados assim como um componente de lista *RecyclerView*, que é definido de acordo com a quantidade de questões e alternativas cadastradas pelo usuário. Com a *view* carregada em memória e com seus componentes devidamente inicializados é feito um processo de conversão, transformado essa *view* em uma imagem do tipo *bitmap*, com dimensões compatíveis com uma folha A4. Isso é realizado através de funções nativas do Android. Por fim, com o arquivo de *bitmap* gerado é utilizada a biblioteca PdfDocument para gerar um arquivo pdf a partir da imagem.

Figura 17 – Template Exame A4



Fonte: Imagem elaborada pelo autor.

3.6.3 Módulo de correção

O módulo de correção é a parte da aplicação responsável por reconhecer a folha de gabarito criada pelo módulo de criação e realizar a correção das questões. Nesse módulo a aplicação realiza a detecção da folha de resposta usando a câmera do dispositivo móvel, tomando como referência as marcações adicionadas na folha. Uma vez realizada a detecção do gabarito, o *frame* é enviado para uma segunda etapa onde é feita a extração das questões e a correção de cada uma delas. A correção foi dividida em duas etapas objetivando uma melhor performance na aplicação.

Na tela inicial o aplicativo faz o processamento de cada *frame* obtido através da câmera do dispositivo. Cada *frame* é submetido a uma função que roda em *looping* (código 3.5).

Listing 3.5 – Algoritimo Encontra Gabarito

```
1    Algoritimo encontrar_gabarito
2        var imagemEncontrada = false
3    Início
```

```
4           enquanto imagemEncontrada = false faça
5               var frame = frameDaCamera
6               var framePreProcessado = preProcessaFrame(frame)
7               var contornos = pegaContornos(framePreProcessado)
8               var quadrantes = pegaQuadrantes(contornos)
9
10              se quadrantes = 4 faça
11                  var pontosFonte = pegaPontosFontes(quadrantes)
12                  var pontosDeDestino = pegaPontosDeDestino(frame)
13                  var imagemTransformada = transformaPerspectiva(matrizFonte,
                        matrizDestino)
14                  var imagemEncontrada <- true
15                  chamaProximaTela(imagemTransformada)
16              fim-se
17          fim-enquanto
18      Fim
```

A primeira etapa do algorítimo busca pré-processar a imagem com o objetivo de evidenciar de uma forma mais clara os contornos da imagem retornando como resultado uma imagem binária 3.6.

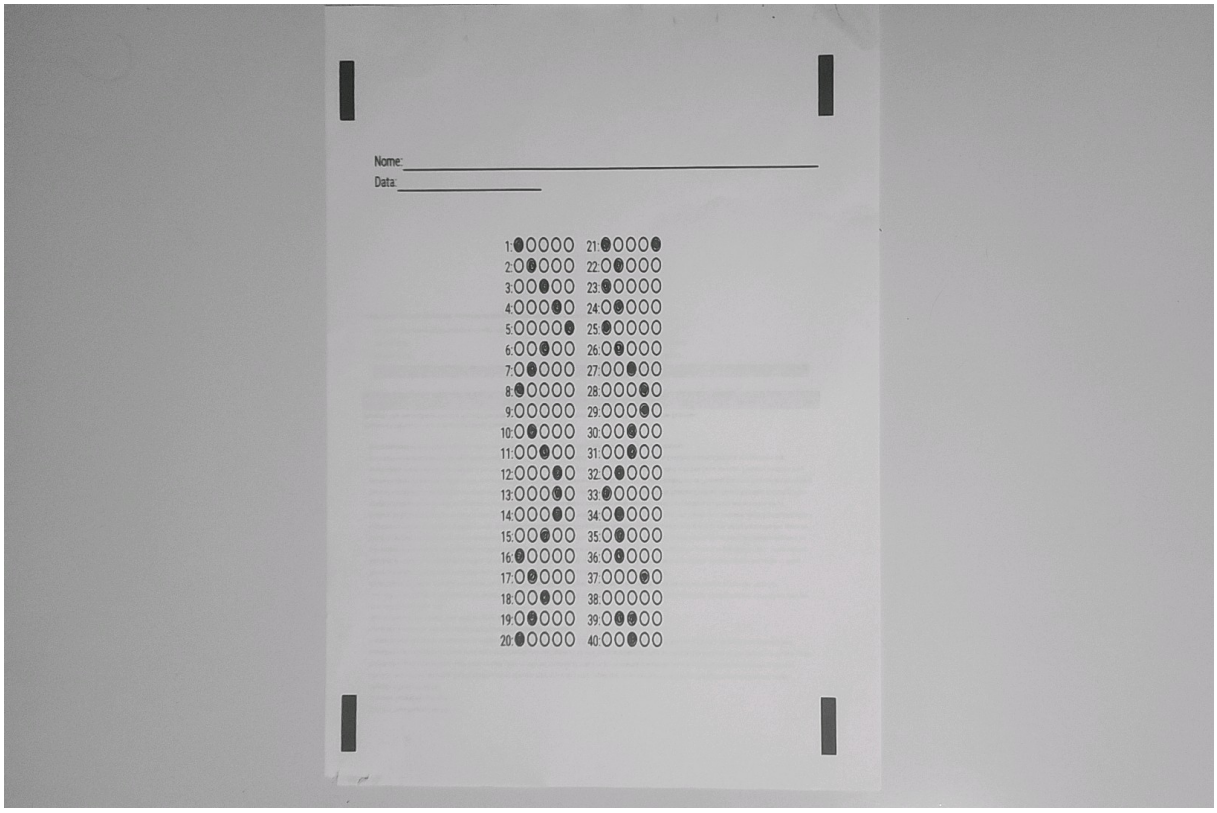
Listing 3.6 – preProcessaFrame

```
1
2   Algoritimo preProcessaFrame
3       var thresh = 100.0
4       var maxVal = 255.0
5       var kernel = Mat.ones(5, 5, CvType.CV_8UC1)
6       var escalaDeCinza = Mat()
7       var imagemBinaria = Mat()
8       var imagemFechada = Mat()
9
10      Início
11          Imgproc.cvtColor(frame, escalaDeCinza, Imgproc.COLOR_BGR2GRAY)
12          Imgproc.threshold(escalaDeCinza, imagemBinaria, thresh, maxVal,
                Imgproc.THRESH_BINARY)
13          Imgproc.morphologyEx(imagemBinaria, imagemFechada, Imgproc.
                MORPH_CLOSE, kernel)
14      Fim
```

A biblioteca do openCV é utilizada para o processamento da imagem. A função preProcessaFrame() realiza a conversão da matriz do frame primeiramente para uma escala de cinza através da função cvtColor() resultando na Figura 18. A imagem convertida é submetida ao processo de *thresholding* através da função threshold(). Esse método recebe um valor de intensidade representado na linha 3 do código 3.6 que será utilizado para realizar a conversão da imagem em escala de cinza em uma imagem binária, resultando na Figura 19.

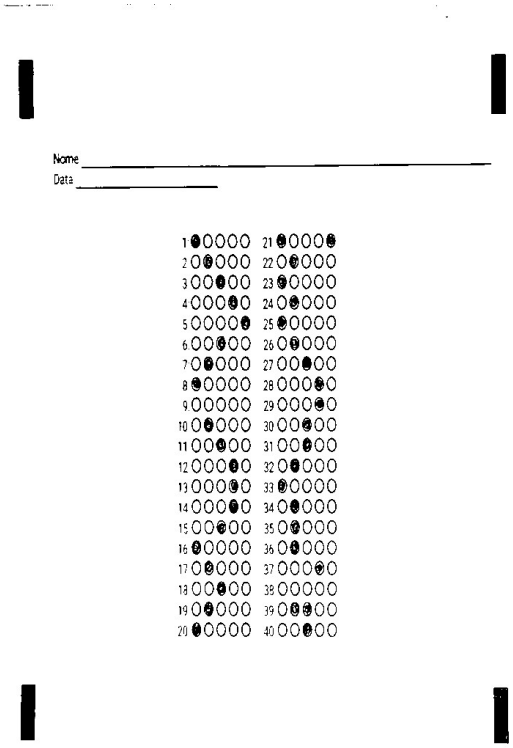
Por fim, a função morphologyEx() faz o processo de fechamento da imagem removendo possíveis ruídos e finalizando a função com a imagem pré processada Figura 20.

Figura 18 – Imagem Cinza



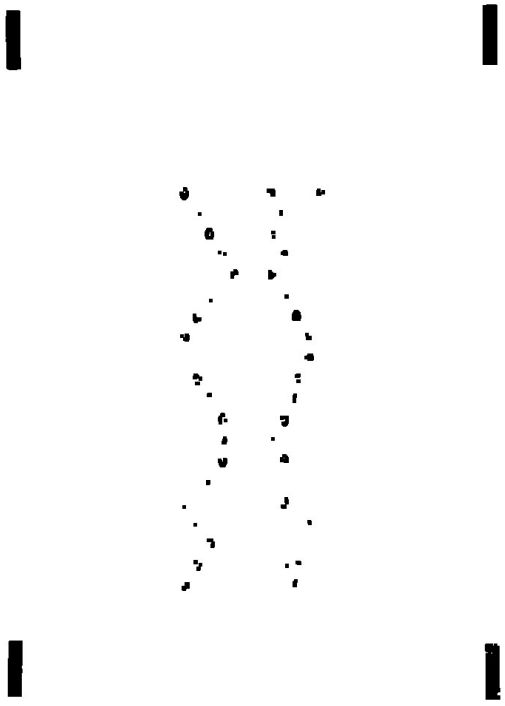
Fonte: Imagem elaborada pelo autor.

Figura 19 – Imagem Binaria



Fonte: Imagem elaborada pelo autor.

Figura 20 – Imagem Fechada



Fonte: Imagem elaborada pelo autor.

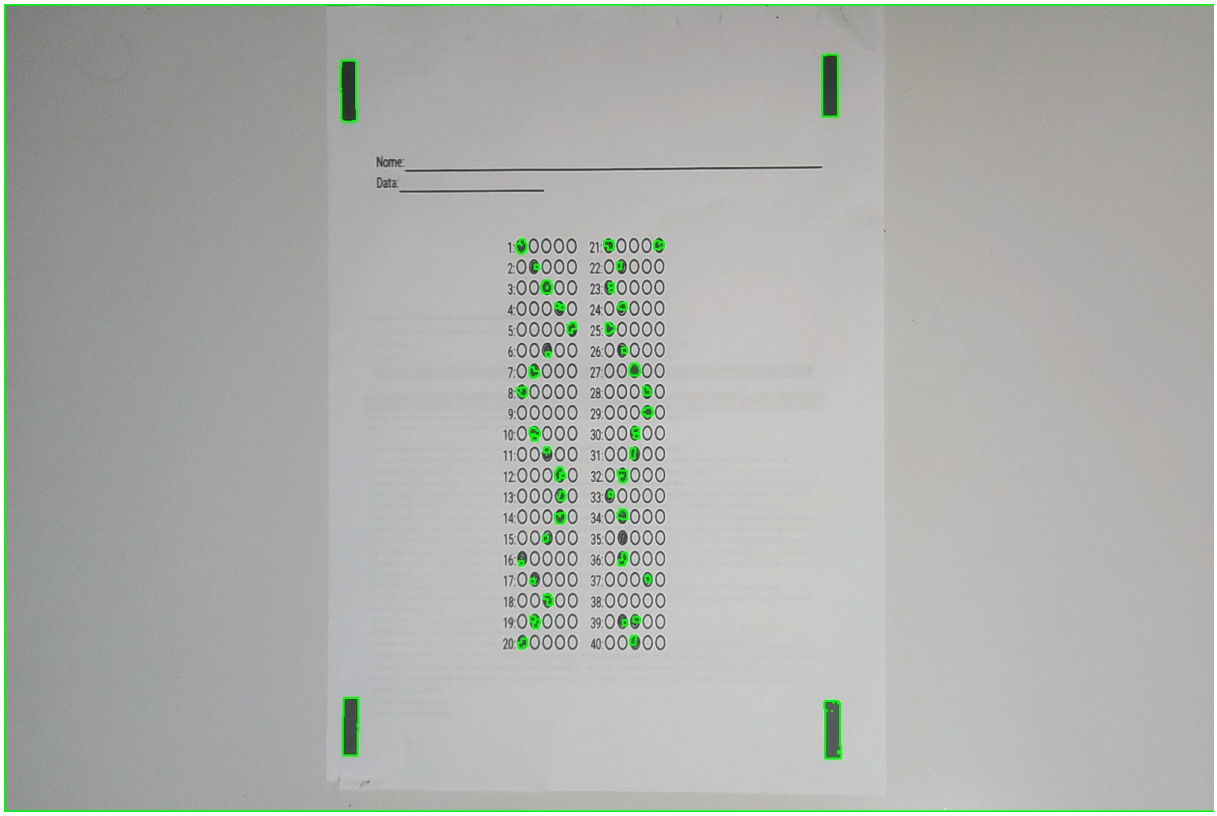
Após a imagem ser pré processada a próxima etapa do algoritmo é realizar a busca dos contornos da imagem.

Listing 3.7 – pegaContornos

```
1
2  Algoritimo pegaContornos
3      var listaDeContornos = []
4      Início
5          Imgproc.findContours(framePreProcessado, listaDeContornos, Mat(),
               Imgproc.RETR_LIST, Imgproc.CHAIN_APPROX_SIMPLE)
6      Fim
```

A função pegaContornos() recebe o *frame* pré processado e retorna uma lista de contornos com cada forma encontrada na imagem Figura 21.

Figura 21 – Imagem com contornos



Fonte: Imagem elaborada pelo autor.

Com a lista de contornos a aplicação passa por cada contorno encontrado e busca achar os quadrantes nas extremidades da folha.

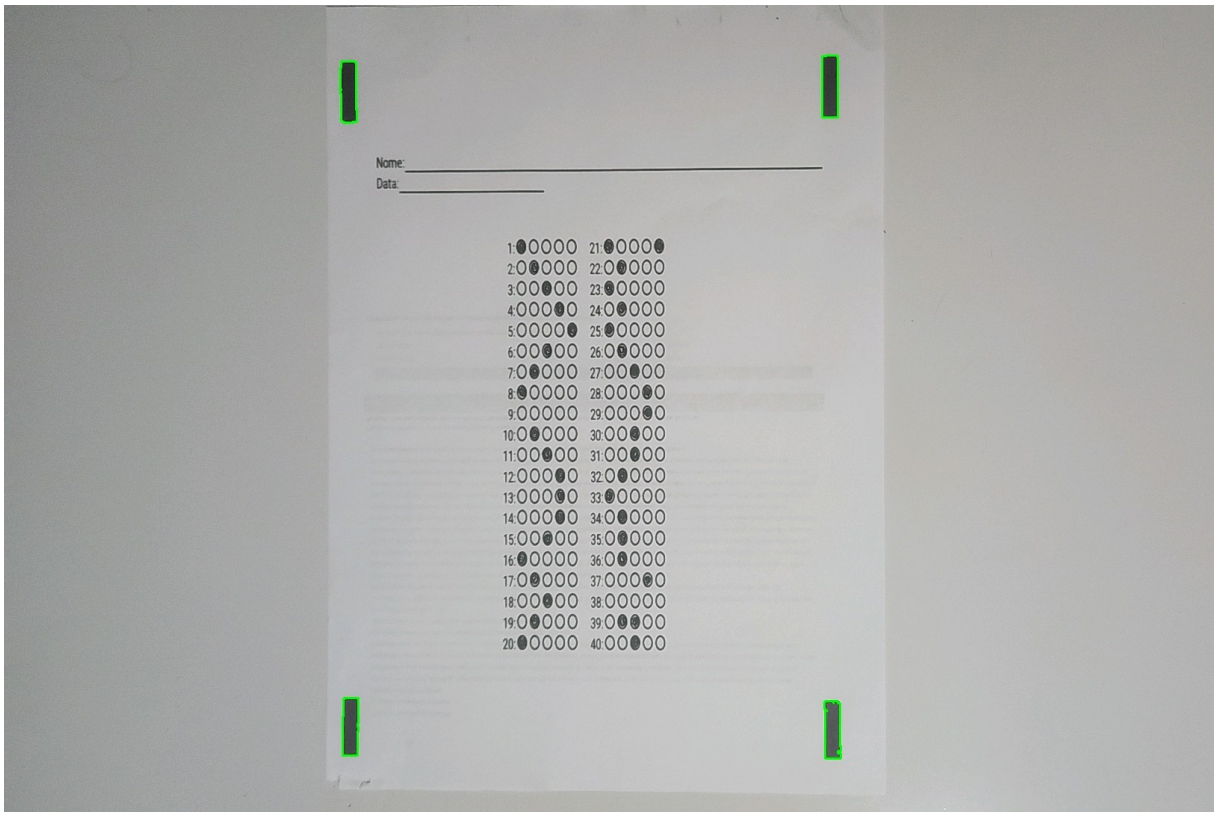
Listing 3.8 – pegaQuadrantes

```
1
2  Algoritimo pegaQuadrantes
3      var maximoAspectRadio = 0.35
```

```
4       var minimoAspectRadio = 0.25
5       var maximoAreaDoContorno = 2200.0
6       var minimaAreaDoContorno = 1400.0
7       var proporcaoPerimetro = 0.02
8       var arestasDeUmQuadrado = 4
9       var listaDeQuadrantes = []
10
11      Início
12          para cada contorno em listaDeContornos faça
13              var perimetro = Imgproc.arcLength(contorno, true)
14              var curvaAproximada = Mat()
15
16              Imgproc.approxPolyDP(contorno, curvaAproximada,
                    proporcaoPerimetro * perimetro, true)
17
18              se curvaAproximada = arestasDeUmQuadrado faça
19                  var area = Imgproc.boundingRect(contorno)
20                  var aspectRatio = area.largura / area.tamanho
21
22                  se aspectRatio entre minimoAspectRadio e maximoAspectRadio
                        faça
23                      se area entre minimaAreaDoContorno e
                            maximoAreaDoContorno faça
24                          listaDeQuadrantes <- listaDeQuadrantes + contorno
25                      fim-se
26                  fim-se
27              fim-se
28          fim-para-cada
29      Fim
```

A função pegaQuadrantes passa por cada contorno encontrado na função anterior e processa cada um separadamente dentro de um *looping*. Nessa função utilizamos o método approxPolyDP() buscando otimizar o contorno e diminuir possíveis arestas geradas por ruídos nos processamentos anteriores. Após esse tratamento, o método valida se o contorno está dentro do tamanho esperado de área e *aspect ratio* e adiciona esse contorno em uma lista de quadrantes.

Figura 22 – Imagem com quadrantes



Fonte: Imagem elaborada pelo autor.

Caso a lista de quadrantes tenha o tamanho igual a quatro a aplicação assume que essa é uma folha de cartão-resposta válida e faz o preparo da imagem para ser enviada para a próxima tela.

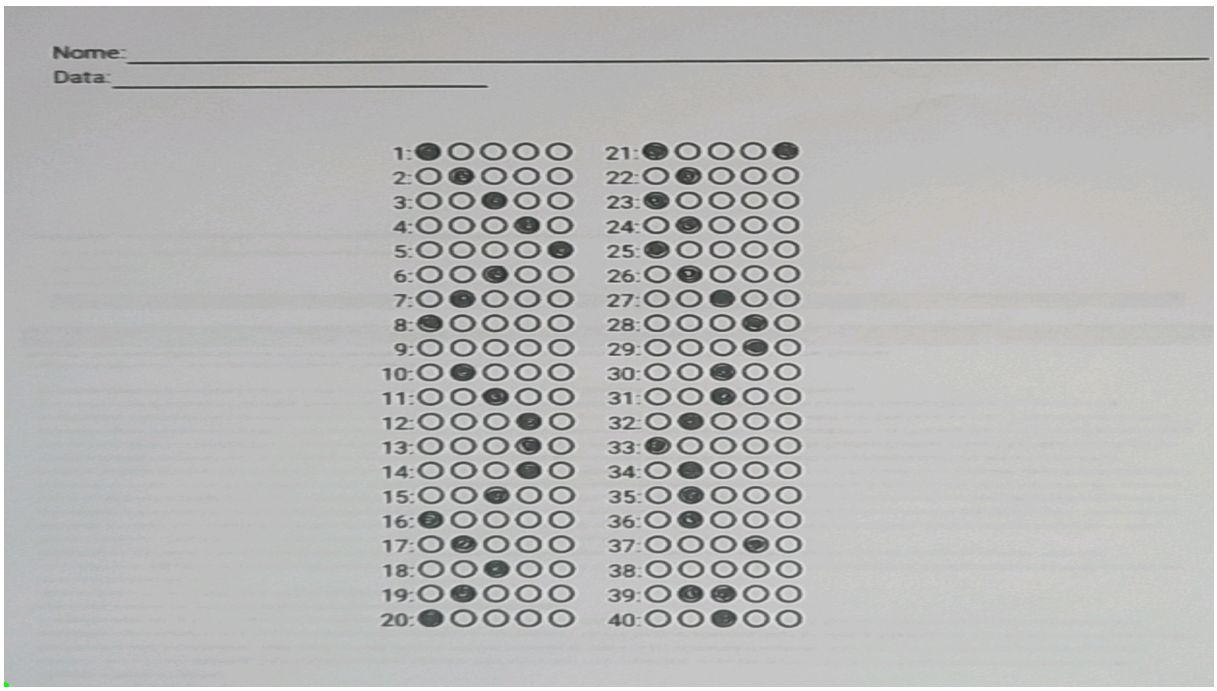
Listing 3.9 – imagemTransformada

```
1
2  Algoritimo imagemTransformada
3      Início
4          val matrizDeTransformacao = Imgproc.getPerspectiveTransform(
               pontosFonte, pontosDeDestino)
5          val imagemTransformada = Mat()
6          Imgproc.warpPerspective(frame, imagemTransformada,
               matrizDeTransformacao, tamanhoDoFrame)
7      Fim
```

Nessa etapa realizamos a transformação de perspectiva na imagem objetivando obter uma imagem apenas com a parte da folha com as questões, essa função primeiramente busca uma matriz resultante através do método getPerspectiveTransform() que recebe uma matriz fonte com os pontos delimitados na área entre os quatro quadrados encontrados e uma matriz de destino que tera seus pontos nas arestas da área total da imagem. Após termos a matriz resultante realizamos a transformação de perspectiva através do método

warpPerspective() gerando a Figura 23.

Figura 23 – Imagem transformada



Fonte: Imagem elaborada pelo autor.

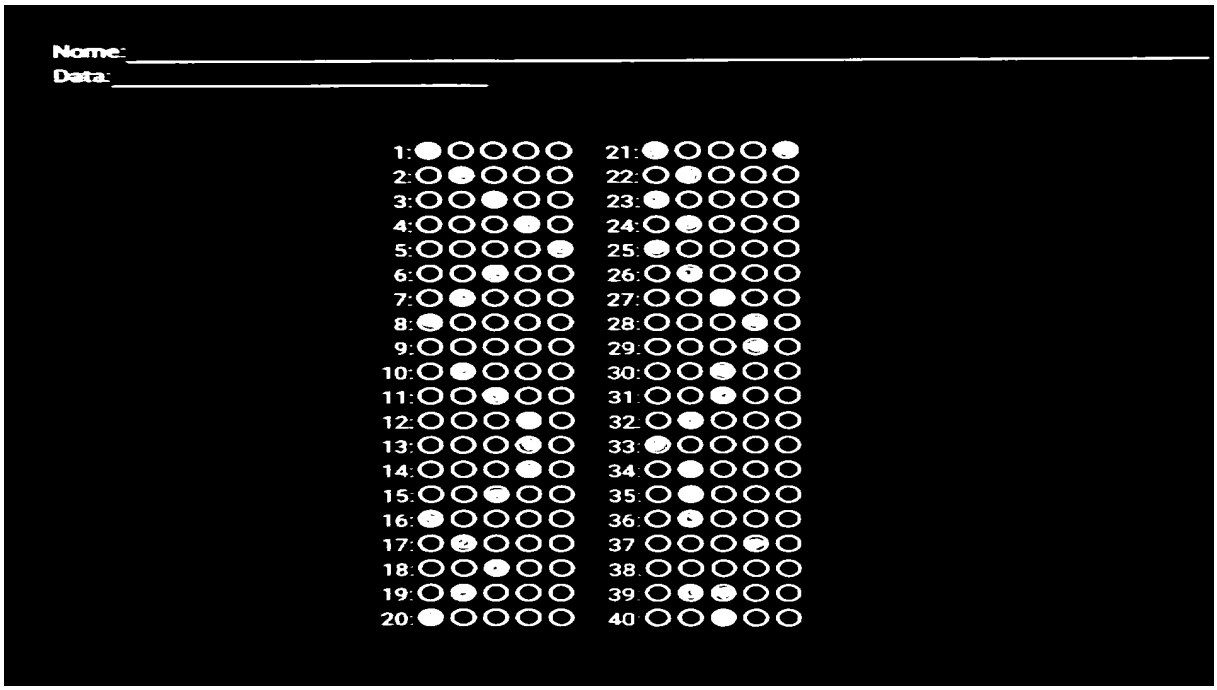
Com a imagem encontrada é feito o envio dessa para a próxima etapa do módulo de correção.

Na segunda etapa a aplicação já possui a imagem pré-selecionada e, em uma outra tela, é feito o processamento da nova imagem através do algorítimo 3.10. A partir da imagem obtida na etapa anterior é feito novamente o pré-processamento através do algorítimo 3.6, com a diferença de que o método threshold utiliza a combinação dos dos métodos otsu e binary_inv para obtermos uma imagem binária invertida e com a intensidade de *pixels* maior e mais evidente, conforme ilustrado na Figura 24.

Listing 3.10 – Algoritimo Corrige Gabarito

```
1   Algoritimo corrige_gabarito
2   Início
3       var frame = frameSelectionado
4       var framePreProcessado = preProcessaFrame(frame)
5       var contornos = pegaContornos(framePreProcessado)
6       var alternativas = pegaAlternativas(contornos)
7       var listaQuestoes = criaListaQuestoes()
8       var listaQuestoesErradas, listaQuestoesCorretas = corrigeQuestoes()
9       desenhaQuestoes()
10  Fim
```

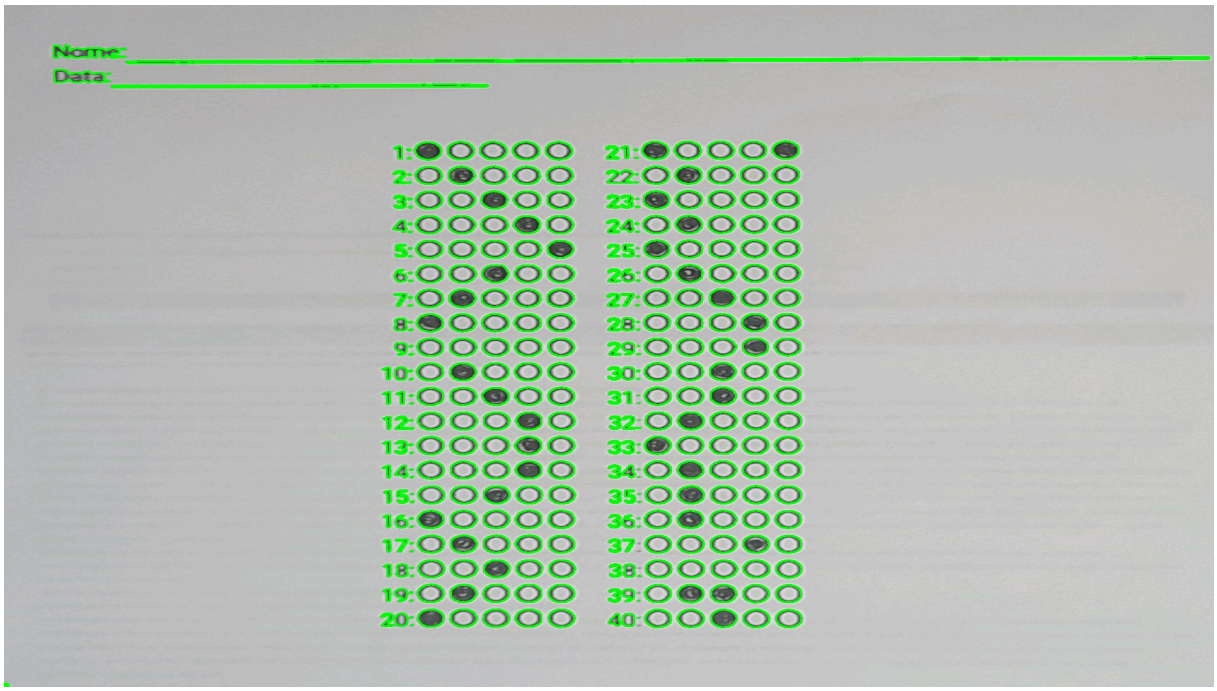
Figura 24 – Imagem Binaria Otsu



Fonte: Imagem elaborada pelo autor.

Após isso, é utilizado o mesmo algorítimo explicado anteriormente 3.7 para obtermos os contornos, conforme a Figura 25.

Figura 25 – Imagem com contornos



Fonte: Imagem elaborada pelo autor.

Com os contornos definidos na listagem, é realizado um *looping* pela lista com o objetivo de encontrar as alternativas baseadas nos valores de areá e *aspect ratio* pré definidos obtendo

o resultado apresentado na Figura 26.

Listing 3.11 – pegaAlternativas

```
1
2  Algoritimo pegaAlternativas
3     var minimoAspectRadio = 1.35
4     var minimaAreaDoContorno = 1000.0
5     var maximaAreaDoContorno = 1600.0
6     var listaDeAlternativas = []
7
8     Início
9         para cada contorno em listaDeContornos faça
10            var area = Imgproc.boundingRect(contorno)
11            var aspectRatio = area.largura / area.tamanho
12
13            se aspectRatio > minimoAspectRadio faça
14                se area entre minimaAreaDoContorno e maximaAreaDoContorno
                      faça
15                    se area entre minimaAreaDoContorno e
                          maximoAreaDoContorno faça
16                        listaDeAlternativas <- listaDeAlternativas +
                              contorno
17                    fim-se
18                fim-se
19            fim-se
20        fim-para-cada
21    Fim
```

Figura 26 – Imagem com as alternativas

Fonte: Imagem elaborada pelo autor.

Com todas as alternativas adicionadas em uma lista a aplicação faz o processo de ordenação exemplificado no algorítimo 3.12 com o objetivo de criar uma lista com as questões ordenadas da esquerda para direita e de cima para baixo. Os valores de colunas, linhas e total de alternativas para cada questão vem das informações fornecidas pelo usuário ao criar o exame. Primeiramente as alternativas são ordenadas pelo valor de y e agrupadas de acordo com a quantidade de opções por linha gerando a lista denominada linhas. Após a ordenação em y é feita a ordenação em x das linhas obtidas. Nesse ponto temos uma nova lista denominada questões com agrupamentos feitos de acordo com a quantidade de alternativas por questão ordenadas da esquerda para direita. Por fim, é feita a ordenação das questões de modo que os agrupamentos sejam sequenciais da esquerda para direita e de cima para baixo.

Listing 3.12 – criaListaQuestoes

```
1
2  Algoritimo ordenaLista
3     var linhas = []
4     var questoes = []
5     var questoesOrdenadas = []
6     var totalDeAlternativasPorColuna = totalColunas * totalAlternativas
7
8     Início
9         linhas = listaDeAlternativas.sortedBy {
10            var area = Imgproc.boundingRect(it)
11            area.y
12        }.chunked(marksByLine)
13
14        para cada linha em linhas faça
15            questoes = linha.sortedBy {
16                var area = Imgproc.boundingRect(it)
17                area.x
18            }.chunked(totalAlternativas)
19
20        fim-para-cada
21
22        para coluna entre 0 ate totalColunas {
23            para linha entre 0 ate totalLinhas) {
24                questoesOrdenadas <- questoesOrdenadas + (questoes[(linha *
                      totalColunas) + coluna])
25            }
26        }
27    Fim
```

Por último, com a listagem das questões ordenadas é feito um novo *looping* com o objetivo de separar a alternativa assinalada de cada questão (Código 3.13). Para cada questão é criado uma imagem com uma alternativa isolada exemplificados nas imagens Figura 27

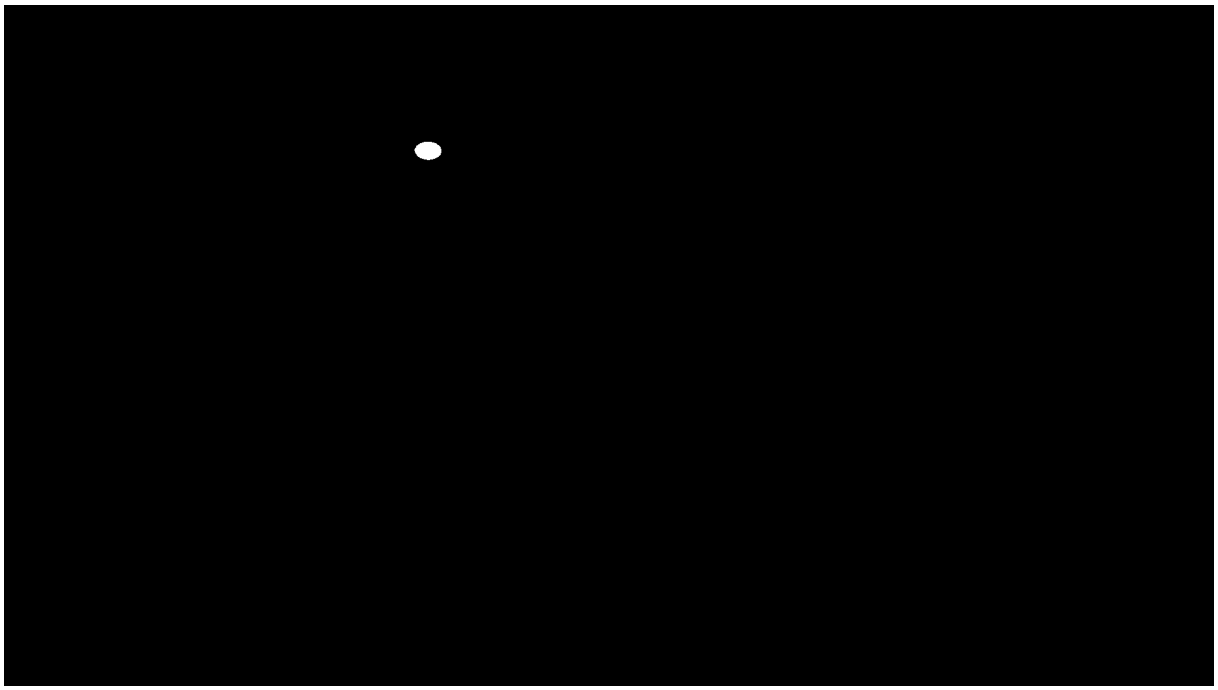
e Figura 28. Nessa imagem é realizada a contagem da intensidade de *pixels* e, caso essa contagem seja maior que a quantidade determinada na variável correspondente, essa alternativa é armazenada. Com a alternativa armazenada é verificado se o valor no index correspondente desse questão na lista de questões é igual ao dessa alternativa armazenada, a depender do resultado, essa alternativa é adicionada na lista de questões corretas ou na lista de questões erradas. No final é feita uma última verificação para saber se uma alternativa preenchida já não havia sido encontrada e, caso tenha, a questão é ignorada e o resultado é apagado da listagem.

Listing 3.13 – corrigeQuestoes

```
1
2  Algoritimo corrigeQuestoes
3     var quantidadeDePixelsQuestaoCorreta = 800
4     var questoesCorretas = []
5     var questoesErradas = []
6
7     Início
8         para cada questao em listaQuestoes faça
9             var repostaEncontrada = null
10            para cada alternativa em questao faça
11                var imageApenasComAlternativa = Mat.zeros(gray.size(),
                      CvType.CV_8UC1)
12                drawContours(imageApenasComUmAlternativa, alternativa, -1,
                      Scalar(255.0), -1)
13
14                val imagemFinal = Mat()
15                Core.bitwise_and(frameOriginal, imageApenasComUmAlternativa
                      , imagemFinal)
16
17                var totalDePixels = Core.countNonZero(imagemFinal)
18
19                if (totalDePixels > quantidadeDePixelsQuestaoCorreta) {
20                        if (repostaEncontrada == null) {
21                            repostaEncontrada = alternativa
22                            if (alternativa.index igual
                                  listaQuestoesCorretas[questao.index]) {
23                                questoesCorretas <- questoesCorretas +
                                      alternativa
24                            } else {
25                                questoesErradas <- questoesErradas +
                                      alternativa
26                            }
27                        } else {
28                            removeAlternativasAdicionadas()
29                            return
30                        }
```

```
31                          }
32                  fim-para-cada
33                  userAnswer = null
34              fim-para-cada
35      Fim
```

Figura 27 – Imagem com mascara questão preenchida



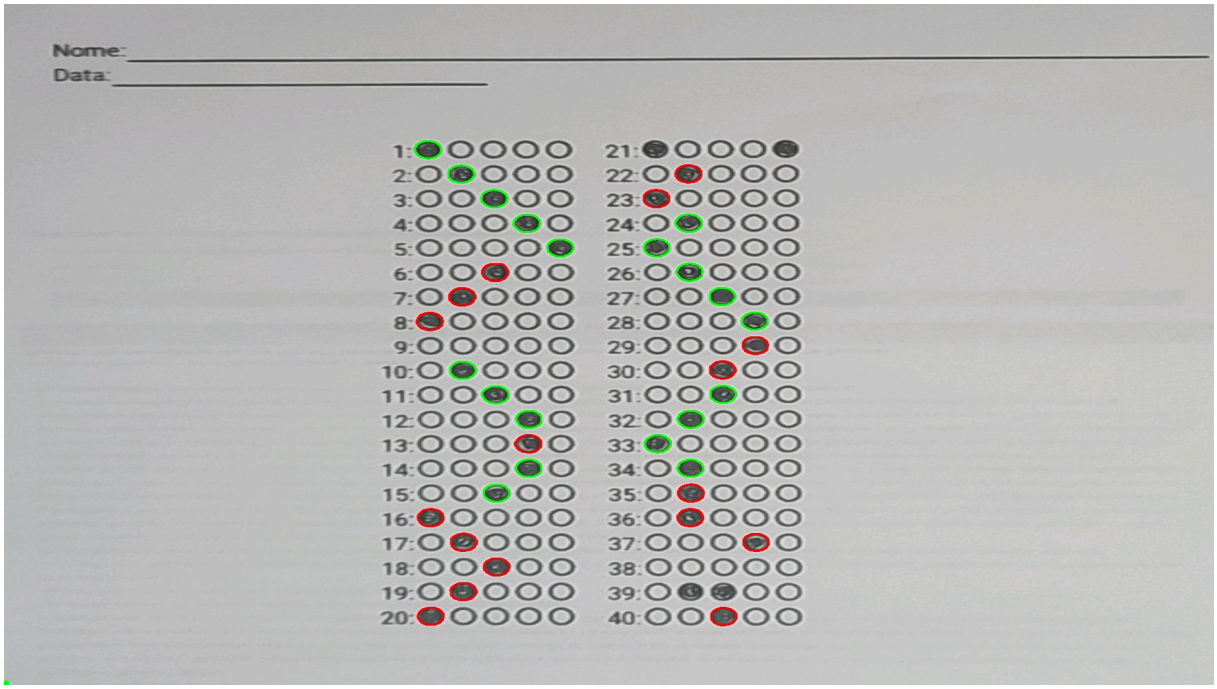
Fonte: Imagem elaborada pelo autor.

Figura 28 – Imagem com mascara questão não preenchida

Fonte: Imagem elaborada pelo autor.

Por fim, é fornecido a imagem com as questões preenchidas pelo usuário circuladas com as cores vermelho que indica o erro e verde que indica o acerto demonstradas na Figura 29. A nota é calculada de acordo com uma expressão matemática que multiplica a quantidade de questões acertadas pelo valor informado de cada questão.

Figura 29 – Imagem folha corrigida



Fonte: Imagem elaborada pelo autor.

4 EXPERIMENTOS, RESULTADOS E DISCUSSÃO

Neste capítulo são apresentados os resultados obtidos no desenvolvimento da aplicação proposta neste trabalho.

Inicialmente, são apresentadas as principais funcionalidades da aplicação, sendo elas: criação da folha de gabarito; listagem dos exames criados; informações de um exame criado, com as funcionalidades de imprimir, corrigir e excluir; listagem de folhas já corrigidas; correção de um exame.

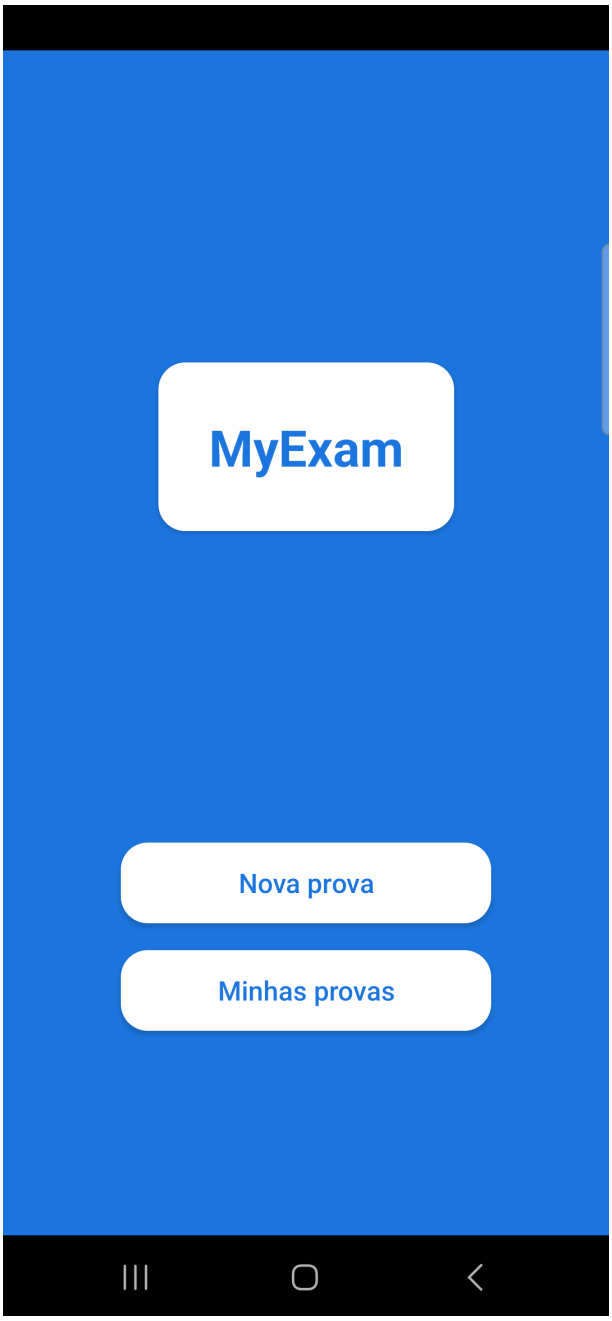
Ao final do capítulo é realizada uma análise do funcionamento da aplicação, mostrando seu comportamento quando são encontrados folhas de cartão-resposta com erros.

4.1 Apresentação das Funcionalidades Principais

Nesta seção serão mostradas as principais funcionalidades da aplicação Android. Para isso, será apresentado um passo a passo de uso da aplicação, indo desde a criação de um folha de cartão-resposta até a correção da mesma.

A tela inicial da aplicação (Figura 30) apresenta o logo do aplicativo myExam. Essa tela possui dois botões utilizados para redirecionar o usuário para as telas de criação de um exame ou de listagem dos exames criados.

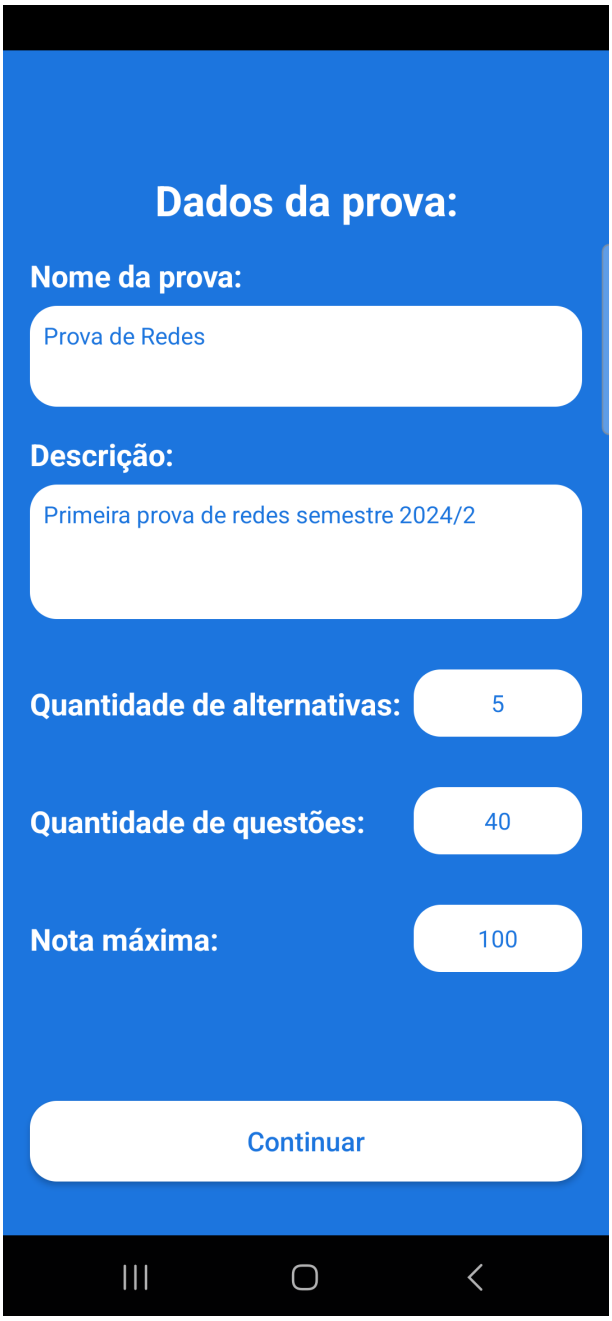
Figura 30 – Tela inicial



Fonte: Imagem elaborada pelo autor.

Ao clicar na opção **"Nova Prova"**, o usuário é redirecionado para tela de cadastro das informações da prova (Figura 31), onde são exibidos os campos de preenchimento para o nome da prova, descrição da prova, quantidade de alternativas por questão, quantidade de questões e nota máxima. Nesta etapa apenas o campo descrição da prova não é de preenchimento obrigatório.

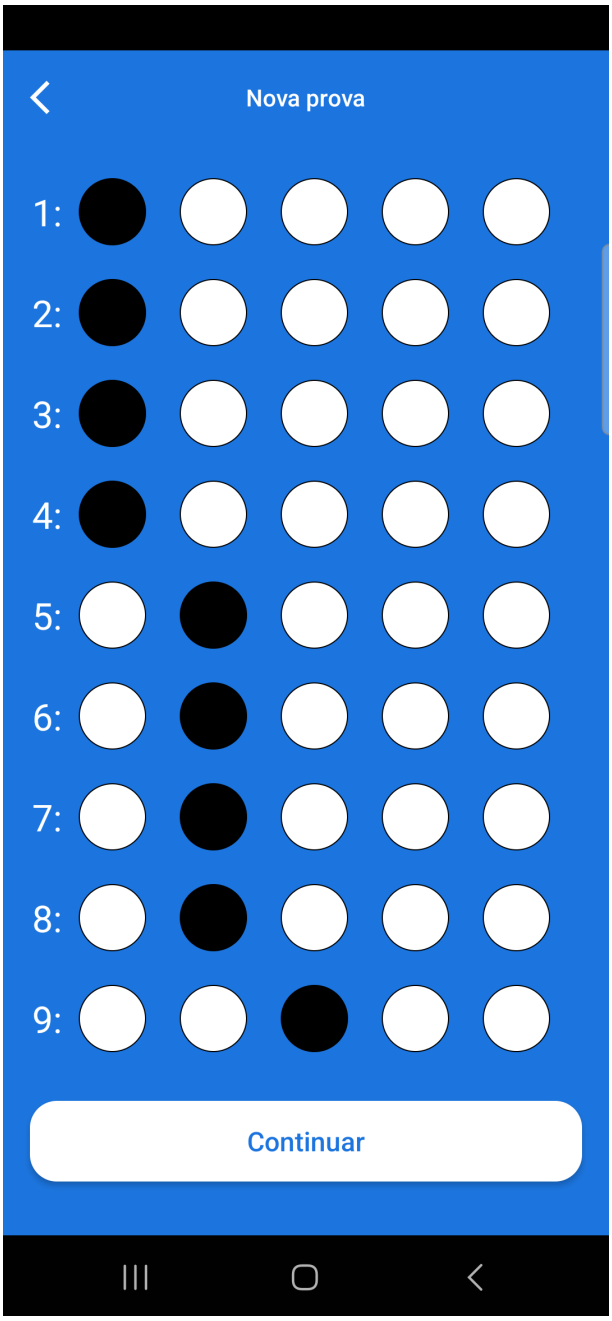
Figura 31 – Tela de cadastro de informações da prova



Fonte: Imagem elaborada pelo autor.

Ao clicar na opção **"Continuar"** o usuário é redirecionado para a tela de preenchimento das alternativas corretas (Figura 32). Nessa tela, é exibida uma listagem de acordo com a quantidade de questões e alternativas informadas na tela anterior. Nesta etapa o usuário só poderá avançar caso preencha cada questão, e a aplicação é responsável por impossibilitar o preenchimento de duas alternativas em uma mesma questão.

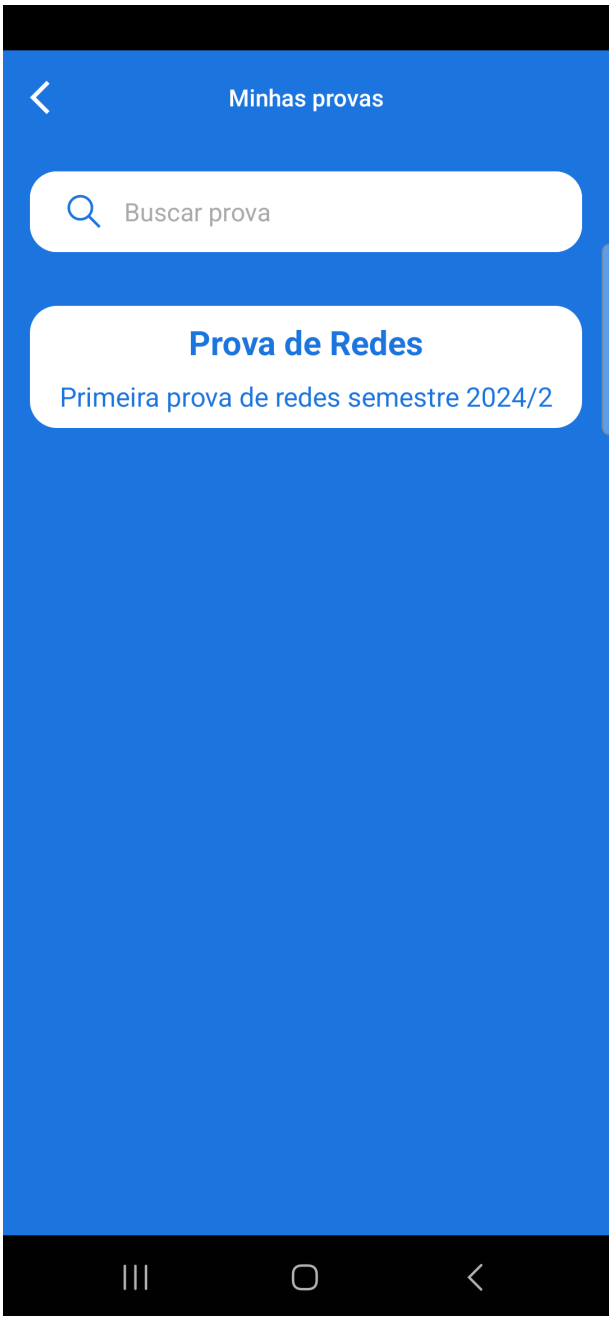
Figura 32 – Tela de cadastro de alternativas corretas



Fonte: Imagem elaborada pelo autor.

Ao clicar na opção **"Continuar"** um exame é criado. O usuário é então redirecionado para a tela de listagem de exames (Figura 33), onde é possível visualizar todos os exames cadastrados anteriormente bem como o atual. Também é possível pesquisar o exame pelo título ao acessar a barra de pesquisa no topo da tela.

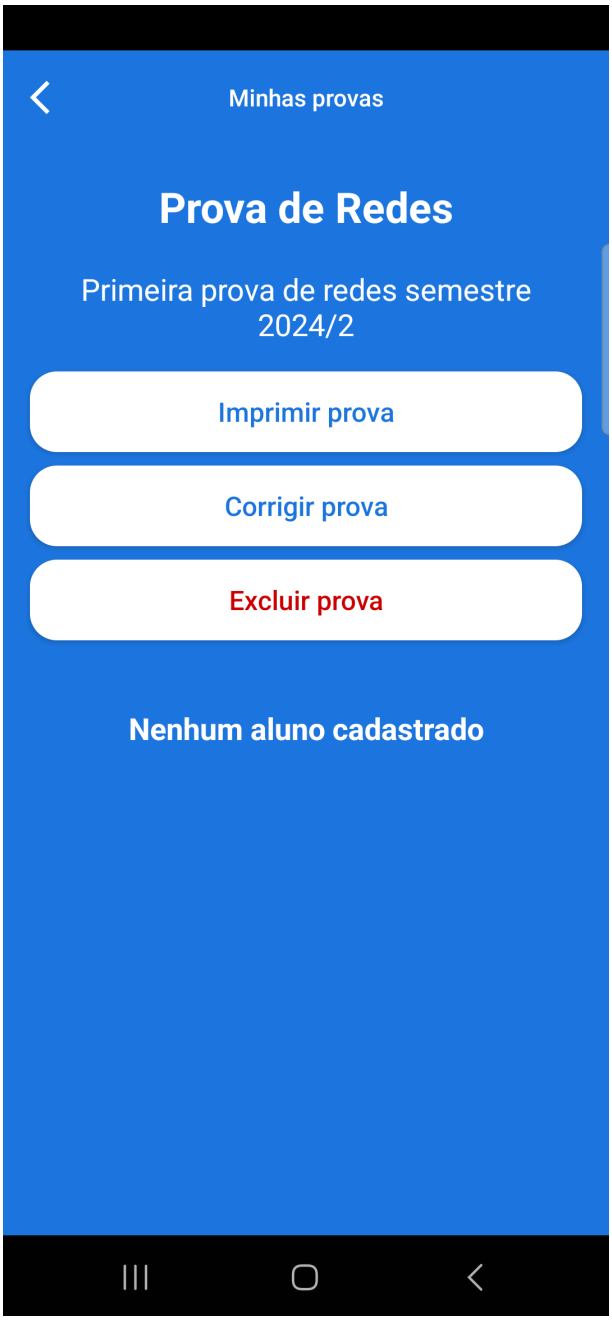
Figura 33 – Tela de listagem de exames



Fonte: Imagem elaborada pelo autor.

Ao selecionar um exame, o usuário é redirecionado para a tela de informações do exame (Figura 34). Nessa tela é mostrado o exame cadastrado com seu título e descrição. Abaixo deles temos as opções de imprimir, corrigir e excluir prova. Também nessa tela é mostrada a lista de alunos que tiveram o exame corrigido. No exemplo dado, porém, não há no momento nenhuma correção realizada.

Figura 34 – Tela de informações do exame



Fonte: Imagem elaborada pelo autor.

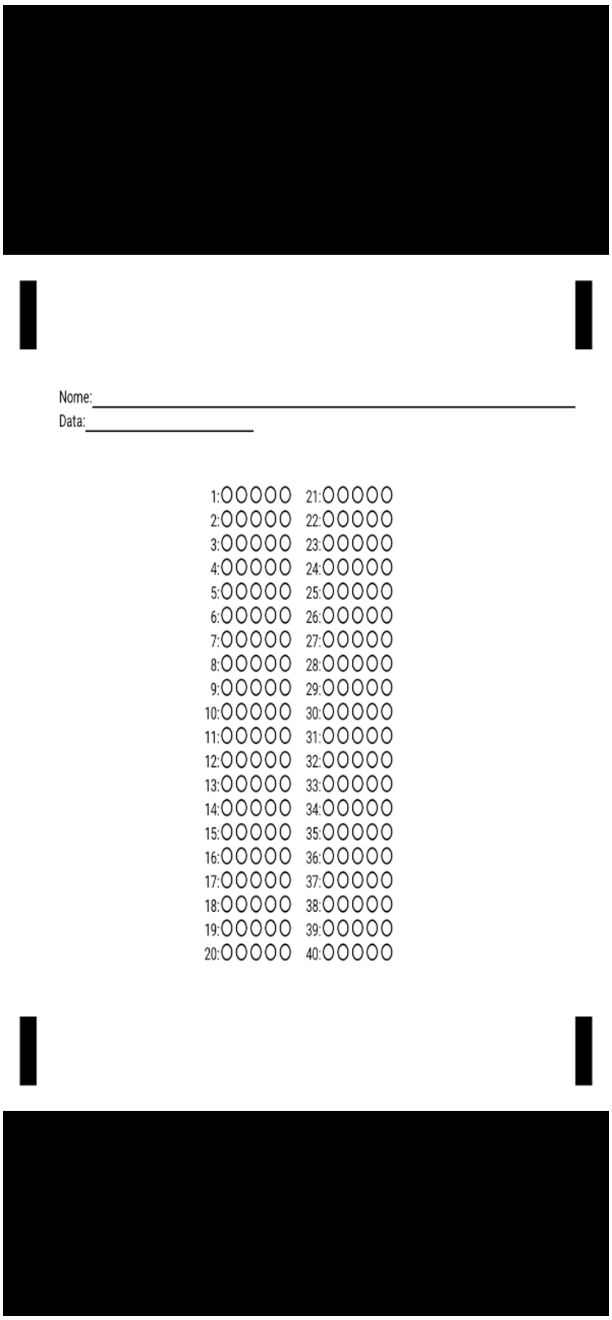
Ao clicar no botão **"Excluir prova"** o aplicativo mostra uma modal pedindo confirmação para a ação (Figura 35). Caso a ação seja confirmada, o exame é excluído e o usuário redirecionado para a tela anterior e caso a ação seja negada o modal desaparece.

Figura 35 – Modal para excluir exame

Fonte: Imagem elaborada pelo autor.

Ao clicar no botão **"Imprimir prova"** o arquivo no formato pdf da folha de cartão-resposta é salvo no dispositivo sendo possível a sua visualização (Figura 36).

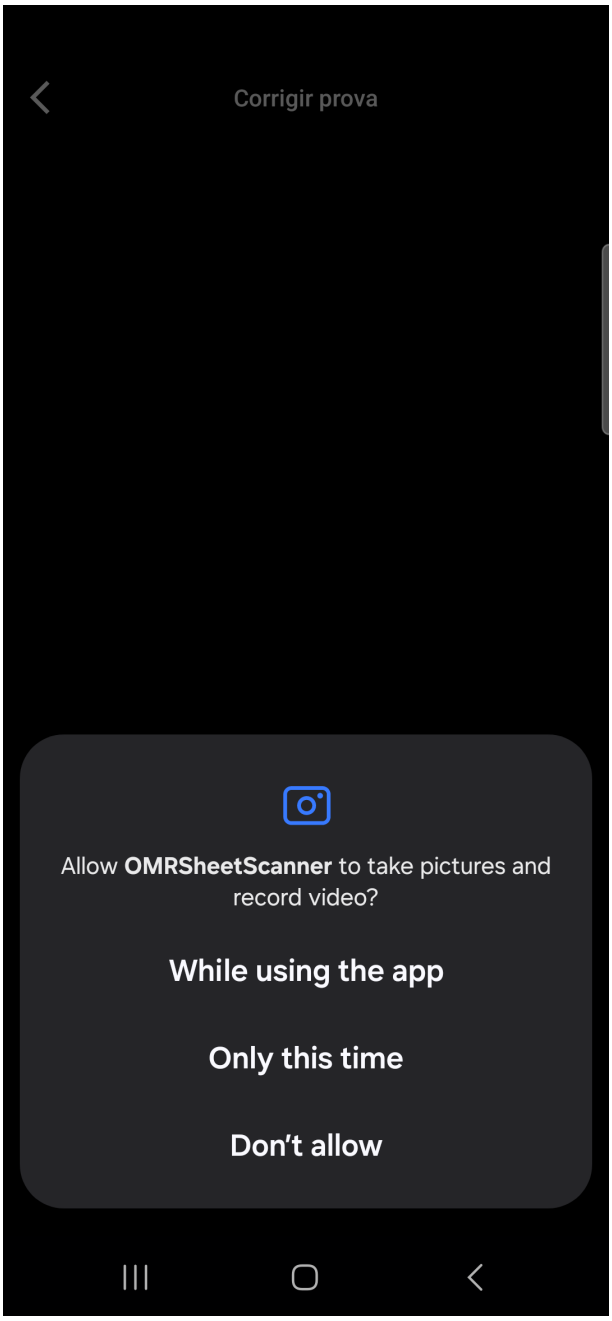
Figura 36 – Folha de gabarito



Fonte: Imagem elaborada pelo autor.

Ao clicar no botão **"Corrigir prova"** o usuário é redirecionado para a tela de captura onde, primeiramente, caso não haja a permissão para utilizar a câmera do dispositivo, a mesma é requisitada (**??**.

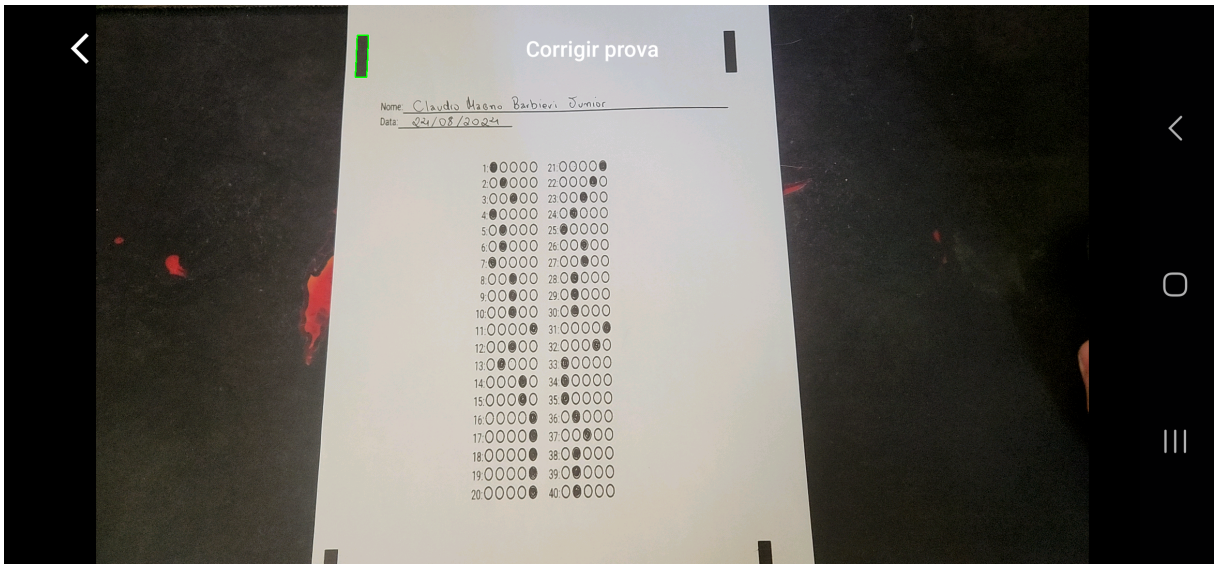
Figura 37 – Tela pedindo a permissão para uso da câmera



Fonte: Imagem elaborada pelo autor.

Com a permissão concedida, a câmera é iniciada e o usuário pode localizar a folha de gabarito que deseja realizar a correção (Figura 38).

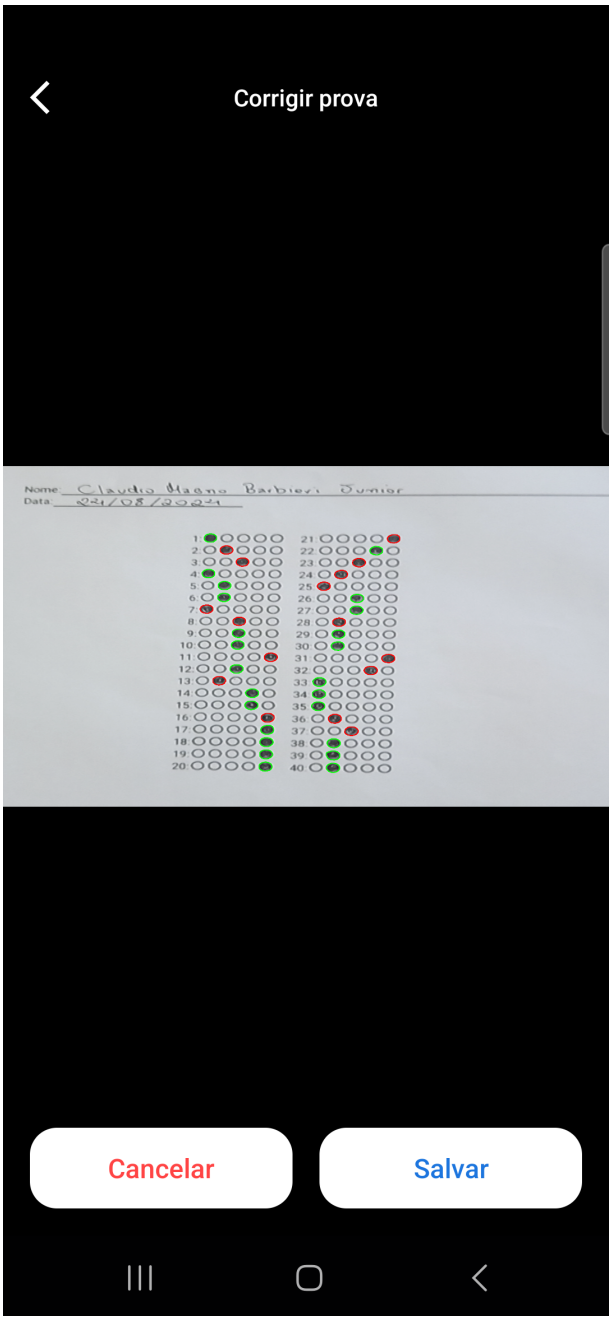
Figura 38 – Tela com a câmera habilitada



Fonte: Imagem elaborada pelo autor.

Após localizar o exame, caso a imagem seja válida o usuário é redirecionado para uma tela onde é mostrada a imagem com as questões corrigidas pela aplicação (Figura 39). Nessa tela as questões corrigidas são circuladas em verde caso estejam corretas e em vermelho caso estejam erradas. O objetivo dessa tela é mostrar ao usuário uma prévia de como a imagem foi processada e as questões corrigidas, sendo possível selecionar a opção de **"Cancelar"** para invalidar a imagem apresentada e retornar para a tela de câmera, com o objetivo de capturar uma imagem melhor.

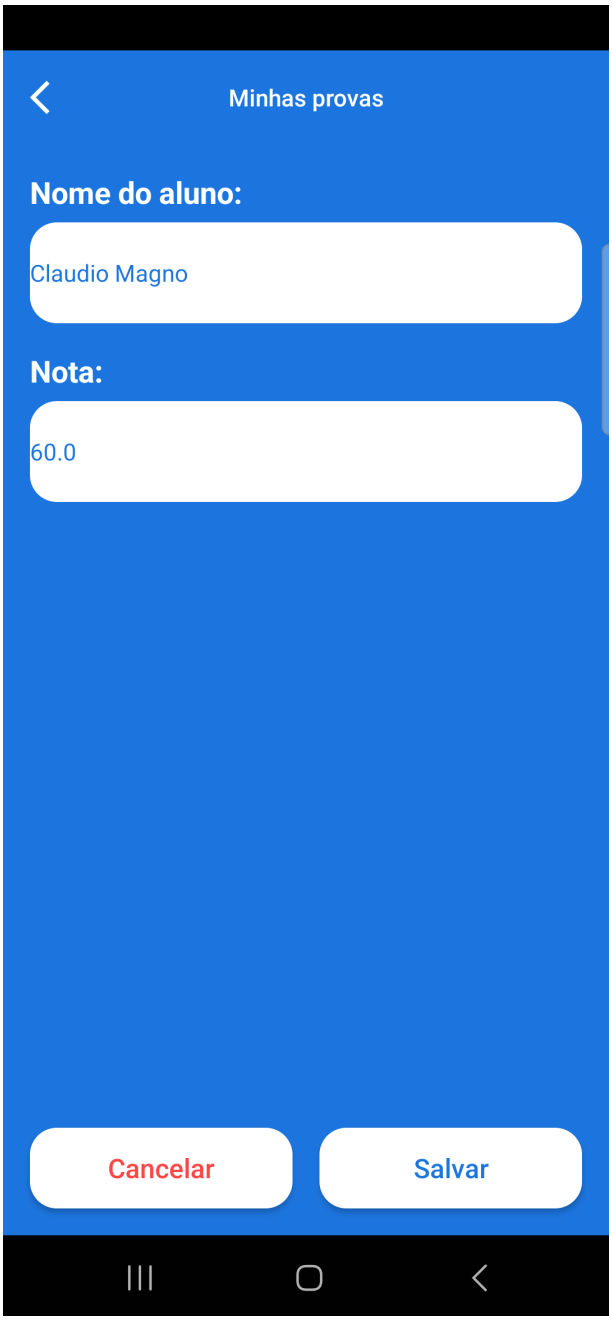
Figura 39 – Tela com a imagem capturada



Fonte: Imagem elaborada pelo autor.

Caso a imagem seja válida o usuário deve selecionar a opção **"Salvar"**. Ao clicar no botão o aplicativo mostra uma tela com campos de nome e nota onde o usuário pode preencher o nome do participante (Figura 40). O campo de nota já é preenchido automaticamente utilizando a correção validada pelo usuário na tela anterior. Nessa tela o usuário pode selecionar a opção **"Cancelar"** para retornar para a tela de câmera e invalidar a correção feita, ou pode selecionar a opção **"Salvar"** para salvar o resultado.

Figura 40 – Tela de salvar exame



Fonte: Imagem elaborada pelo autor.

Caso o usuário clique na opção **"Salvar"** o aplicativo mostra novamente a tela de informações do exame, porém dessa vez com a lista de alunos com exames corrigidos atualizada (Figura 41).

Figura 41 – Tela de informações do exame com aluno

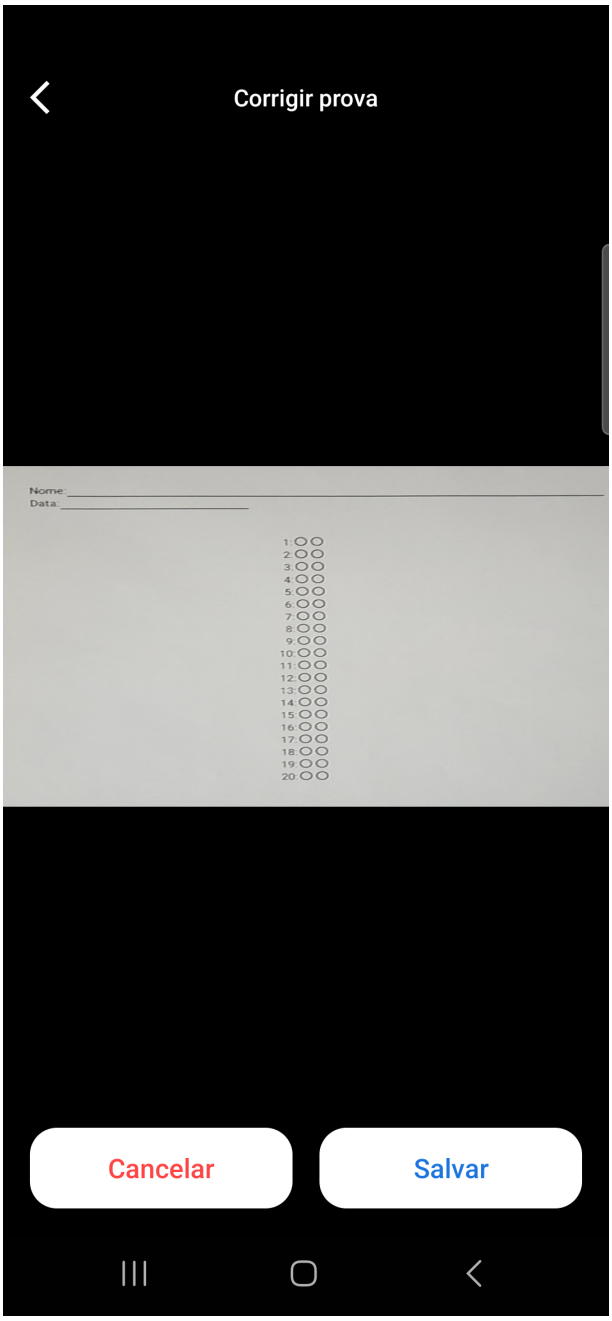
Fonte: Imagem elaborada pelo autor.

4.2 Testes

Nesta seção será mostrado o desempenho da aplicação ao realizar a correção de folhas de cartão-resposta em alguns cenários específicos.

No primeiro cenário foi realizado o teste da correção de uma folha de cartão-resposta sem nenhuma questão preenchida, neste cenário o aplicativo apresentou corretamente um total de 0 questões avaliadas conforme apresentado na Figura 42.

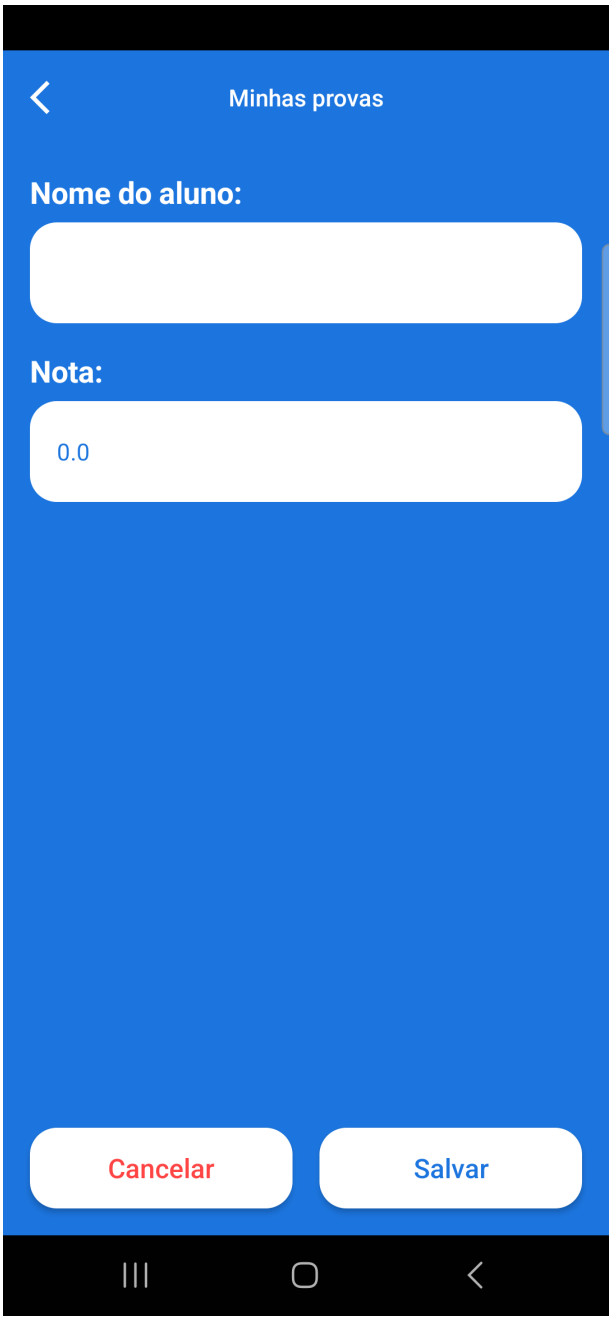
Figura 42 – Tela com imagem capturada sem questões preenchidas



Fonte: Imagem elaborada pelo autor.

Ao avançar com essa correção a aplicação exibe a nota obtida ao avaliar a folha. Nesse caso o resultado da avaliação é 0 pois não há nenhuma questão correta (Figura 43).

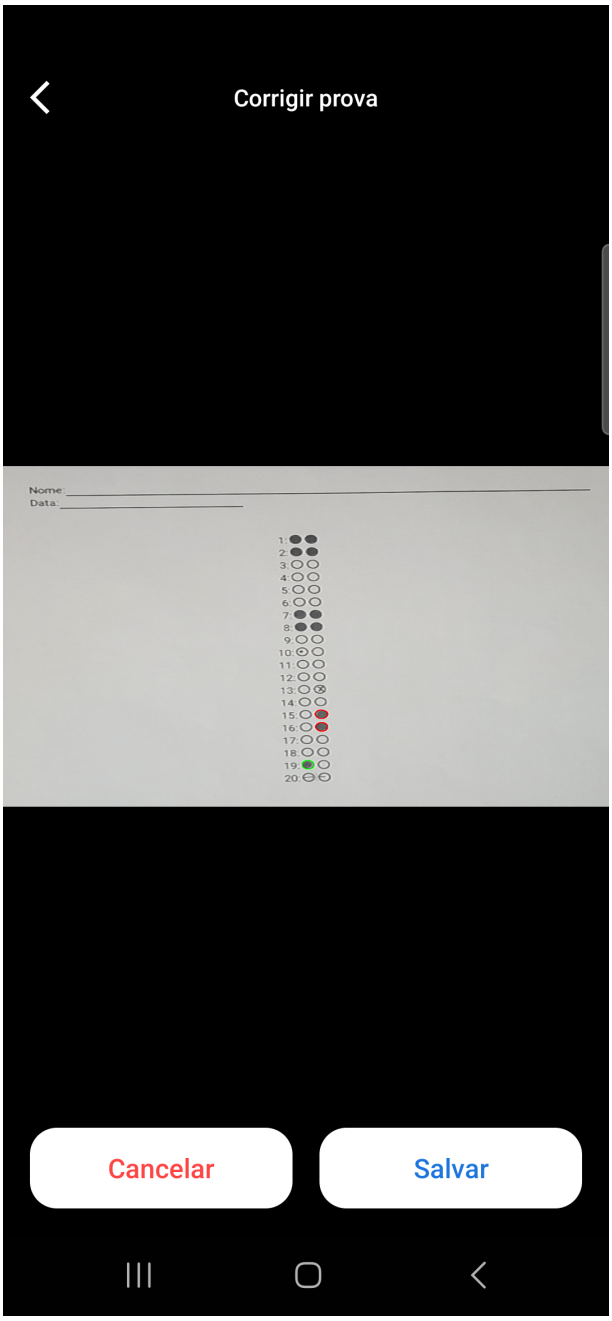
Figura 43 – Tela com a nota obtida para o exame sem questões preenchidas



Fonte: Imagem elaborada pelo autor.

No segundo cenário foi realizado o teste da correção de uma folha de cartão-resposta com questões preenchidas de forma errada, o objetivo desse cenário é analisar a inteligência da aplicação ao corrigir questões duplicadas, pouco ou não preenchidas. Para o melhor entendimento desse cenário a folha corrigida foi criada de modo em que todas as questões da primeira fileira estão corretas. Na Figura 44 podemos perceber que a única questão correta é a número 19, além de ser exibido o erro nas questões 15 e 16. As demais questões não foram analisadas pelo aplicativo devido aos erros no preenchimento.

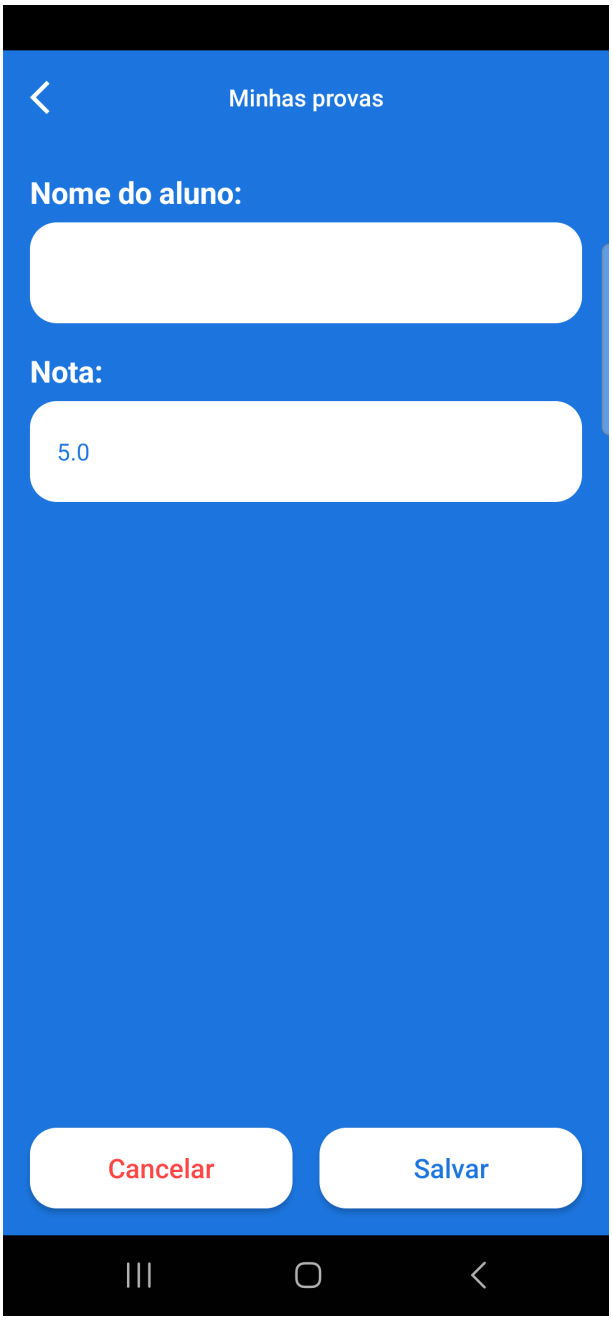
Figura 44 – Tela com imagem capturada questões erradas



Fonte: Imagem elaborada pelo autor.

Ao avançar com essa correção a aplicação exibe a nota obtida ao avaliar a folha. Nesse caso o resultado da avaliação é 5 pois só temos uma questão correta e a nota máxima do exame é 100 Figura 45.

Figura 45 – Tela com a nota obtida para o exame com questões erradas



Fonte: Imagem elaborada pelo autor.

5 CONSIDERAÇÕES FINAIS

Neste trabalho foi desenvolvido uma aplicativo para Android para corrigir folhas de cartão-resposta. O aplicativo possui duas principais funcionalidades sendo elas a criação do exame e a correção da folha de cartão-resposta.

Para o desenvolvimento do projeto foram utilizados principalmente fundamentos de desenvolvimento *mobile* para Android, como por exemplo: integração de banco local ROOM, manipulação de *view's* e de componentes nativos como câmera e armazenamento do dispositivo.

Para que fosse possível realizar a correção de uma folha de cartão-resposta a partir da câmera foi necessário a aplicação de conceitos de processamento de imagem aplicados em conjunto com a biblioteca OpenCV. A linguagem utilizada para o desenvolvimento da aplicação foi o Kotlin.

Um grande desafio para o desenvolvimento da aplicação foi o alinhamento da imagem obtido pela câmera para que houvesse uma maior nitidez ao realizar o processamento. Para isso foram adicionados elementos na folha com altura e largura definidos, possibilitando o algorítimo de reconhecimento ter como primeiro passo a identificação desses elementos. Com isso adicionado, definimos o tamanho que esses elementos deveriam ter na imagem mostrada pela câmera possibilitando a calibragem da distância entre o dispositivo e a folha.

Outro grande desafio, foi a ordenação das questões extraídas da folha de modo que fossem sequenciais, da primeira para a última e da esquerda para a direita. Nesse processo, foram utilizados conceitos de ordenação de matriz para termos as questões de forma sequencias, independente da quantidade de linhas ou colunas em que as questões estivessem dispostas.

Os resultados obtidos durante o desenvolvimento e a produção dos testes foram satisfatórios desde que a correção fosse feita em um ambiente bem iluminado e sem sombras na folha. Nesses casos as questões analisadas e nota resultante da correção foi calculada corretamente em todos os testes.

Por fim, é possível concluir que a aplicação desenvolvida pode ser utilizada na correção de folhas de cartão-resposta de maneira eficiente e segura.

5.1 Trabalhos futuros

Apesar da proposta do trabalho ter sido alcançada, algumas melhorias podem ser definidas em futuros trabalhos:

- Desenvolver solução para a troca a orientação da câmera para vertical.
- Desenvolver solução para a utilização do *flash* da câmera durante a captura da imagem.
- Desenvolver aplicação *mobile* para o sistema operacional iOS.
- Criar solução para salvamentos dos resultados em um servidor online.
- Criar página web para monitoramento dos exames corrigidos.

REFERÊNCIAS

DEVELOPER, A. *Android SDK*. 2023. Disponível em: <https://developer.android.com/tools>. Acesso em: 30 mai. 2024.

DEVELOPER, A. *Data Layer*. 2023. Disponível em: <https://developer.android.com/topic/architecture/data-layer>. Acesso em: 30 mai. 2024.

DEVELOPER, A. *Guide to app architecture*. 2023. Disponível em: <https://developer.android.com/topic/architecture>. Acesso em: 30 mai. 2024.

DEVELOPER, A. *Room*. 2023. Disponível em: <https://developer.android.com/training/data-storage/room>. Acesso em: 30 mai. 2024.

DEVELOPER, A. *Dagger Hilt*. 2024. Disponível em: <https://developer.android.com/training/dependency-injection/hilt-android>. Acesso em: 30 mai. 2024.

DEVELOPER, A. *Data Binding*. 2024. Disponível em: <https://developer.android.com/topic/libraries/data-binding>. Acesso em: 30 mai. 2024.

DOCUMENTATION, O. *Image Thresholding*. 2023. Disponível em: <https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>. Acesso em: 05 jun. 2023.

DOCUMENTATION, O. *Morphological Transformations*. 2023. Disponível em: <https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html>. Acesso em: 05 jun. 2023.

FIRESMITH, D. G. Security use cases. *Journal of object technology*, v. 2, n. 3, 2003.

GATE, R. *EXPLORANDO AS TÉCNICAS DE RECONHECIMENTO DE IMPRESSÕES DIGITAIS - Scientific Figure on ResearchGate*. 2023. Disponível em: <https://www.researchgate.net/figure/Passos-para-o-processamento-digital-de-imagens-b-Aquisicao-de-imagem-forma-com-que-a_fig2_349181208>. Acesso em: 05 jun. 2023.

GONZALEZ, R. C.; WOODS, R. E. *Processamento de imagens digitais*. [S.l.]: Editora Blucher, 2000.

HTTPS://EVALBEE.COM/. *EvalBee*. 2023. Disponível em: <https://evalbee.com/>. Acesso em: 06 jun. 2023.

HTTPS://WWW.MINHAPROVA.COM.BR/. *Sistema Minha Prova*. 2023. Disponível em: <https://www.minhaprova.com.br/>. Acesso em: 06 jun. 2023.

HUSSMANN, S.; DENG, P. W. A high-speed optical mark reader hardware implementation at low cost using programmable logic. *Real-Time Imaging*, Elsevier, v. 11, n. 1, p. 19–30, 2005.

IBM. *Automated Test Scoring*. 2012. Disponível em: <https://www.ibm.com/history/805-scoring-test?mhsrc=ibmsearch_a&mhq=IBM%20805>. Acesso em: 3 de maio de 2023.

LINDEVS. *Convert RGB Image to Grayscale Image using OpenCV*. 2023. Disponível em: <https://lindevs.com/convert-rgb-image-to-grayscale-image-using-opencv>. Acesso em: 05 jun. 2023.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125–160, 2009.

OPENCV. *Changing Colorspaces*. 2023. Disponível em: <https://docs.opencv.org/>. Acesso em: 06 nov. 2023.

OPENCV. *Contours in OpenCV*. 2023. Disponível em: <https://docs.opencv.org/>. Acesso em: 06 nov. 2023.

OPENCV. *Geometric Transformations of Images*. 2023. Disponível em: <https://docs.opencv.org/>. Acesso em: 06 nov. 2023.

OPENCV. *Image Thresholding*. 2023. Disponível em: <https://docs.opencv.org/>. Acesso em: 06 nov. 2023.

OPENCV. *Morphological Transformations*. 2023. Disponível em: <https://docs.opencv.org/>. Acesso em: 06 nov. 2023.

PRINT, G. N. &. *Serviço de leitura de gabaritos, cartões resposta e correção de provas para concursos, instituições de ensino e empresas*. 2020. Disponível em: <http://gbnet.com.br/blog/index.php/2020/captura-de-dados-2/servico-de-leitura-de-gabaritos-cartoes-resposta-e-correcao-de-provas-para-concursos-instituicoes-de-e>. Acesso em: 05 jun. 2023.

SHAIKH, R. *OpenCV (findContours) Detailed Guide*. 2020. Disponível em: <https://medium.com/analytics-vidhya/opencv-findcontours-detailed-guide-692ee19eeb18>. Acesso em: 05 jun. 2023.

SHAIKH, R. *OpenCv Perspective Transformation*. 2020. Disponível em: <https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143>. Acesso em: 05 jun. 2023.

SILVA, T. Minha prova-automatizando a correção de provas nas escolas. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2018. v. 7, n. 1, p. 150.

SOMMERVILLE, I. *Engenharia de Software*. [S.l.]: Pearson Education, 2011.

Wikipedia. *Diagrama de casos de uso*. 2020. Disponível em: <https://pt.wikipedia.org/wiki/Diagrama_de_caso_de_uso/>. Acesso em: 16 de novembro de 2023.