INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL MECÂNICA
CURSO DE GRADUAÇÂO EM ENGENHARIA MECATRÔNICA

Marcos Roberto Martini Monteiro

Busca Semântica de Imagens: Uma Aplicação Móvel Integrada a Serviços de IA na Nuvem

FLORIANÓPOLIS, 2025.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL MECÂNICA
CURSO DE GRADUAÇÂO EM ENGENHARIA MECATRÔNICA

Marcos Roberto Martini Monteiro

Busca Semântica de Imagens: Uma Aplicação Móvel Integrada a Serviços de IA na Nuvem

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro em Mecatrônica.

Orientador:
Prof. Dr. Delcino Júnior Picinin

FLORIANÓPOLIS, 2025.

Ficha de identificação da obra elaborada pelo autor.

Monteiro, Marcos
Busca Semântica de Imagens: Uma Aplicação Móvel Integrada **a Serviços de IA na Nuvem / Marcos** Monteiro; orientação **de Delcino Picinin. - Florianópolis, SC,** 2025.
63 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal **de Santa Catarina, Câmpus** Florianópolis. Bacharelado **em** Engenharia Mecatrônica. Departamento **Acadêmico de Metal** Mecânica.
Inclui Referências.

1. Busca Semântica. 2. Armazenamento em Nuvem. 3. **Criptografia. 4. Análise de Imagens. I.** Picinin, Delcino. **II. Instituto Federal de Santa Catarina. III.** Busca **Semântica de Imagens: Uma Aplicação Móvel** Integrada **a Serviços de IA** na Nuvem.

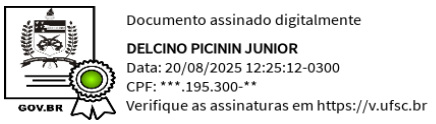
BUSCA SEMÂNTICA DE IMAGENS: UMA APLICAÇÃO MÓVEL INTEGRADA A SERVIÇOS DE IA NA NUVEM

MARCOS ROBERTO MARTINI MONTEIRO

Este Trabalho foi julgado adequado para obtenção do Título de Engenheira Mecatrônica em Julho de 2025 e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Mecatrônica do Instituto Federal de Educação Ciência e Tecnologia de Santa Catarina.

Florianópolis, 23 de Julho, 2025.

Banca Examinadora:



Delcino Picinin Júnior, Doutor
IFSC

Maurício Edgar Stivanello, Doutor
IFSC

Raimundo Ricardo Matos da Cunha, Doutor
IFSC

RESUMO

Este trabalho propõe o desenvolvimento de um protótipo de sistema de gerenciamento de fotos com busca semântica baseada em inteligência artificial, visando oferecer uma alternativa simplificada a soluções como o Google Fotos. O foco principal reside no controle e domínio de dados pessoais pelos usuários. Para tal, foi construído um servidor, onde ocorre a autenticação, armazenagem de dados em um banco SQL, criptografia e autorização de acesso, junto a um aplicativo para Android, o qual realiza o envio de imagens ao servidor, permite a visualização, pesquisa semântica (baseada na análise de IA realizada no servidor) e deleção de imagens. A análise busca, também, comparar o custo de diferentes provedores de armazenamento em nuvem e o modelo de criptografia a ser empregado, como criptografia em repouso ou de ponta a ponta, sendo essa última não recomendada devido ao fato de que dificultaria a análise de imagens no servidor. Por fim, são expostos os resultados, concluindo-se que é possível construir uma solução comparável ao Google Fotos e, ao mesmo tempo, reter controle sobre os dados pessoais.

Palavras-chave: Busca Semântica. Armazenamento em Nuvem. Criptografia. Análise de Imagens. Google Fotos (Alternativa).

ABSTRACT

This paper proposes the development of a prototype photo management system with AI-based semantic search, aiming to provide a simplified alternative to solutions like Google Photos, with a primary focus on user control and ownership of personal data. To accomplish this, a server was built to handle authentication, data storage in an SQL database, encryption, and access authorization, along with an Android application that enables image uploading, viewing, semantic search (based on AI analysis performed on the server), and image deletion. The analysis also compares the cost of different cloud storage providers and the encryption model to be used, such as encryption at rest or end-to-end encryption, the latter being discouraged as it would hinder image analysis on the server. Finally, the results show that it is possible to build a solution comparable to Google Photos while maintaining control over personal data.

Keywords:Semantic Search. Cloud storage. Encryption. Image Analysis. Google Photos (Alternative).

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

GCS	Google Cloud Storage
GCV	Google Cloud Vision
IFSC	Instituto Federal de Santa Catarina
PLN	Processamento de Linguagem Natural

SUMÁRIO

1 INTRODUÇÃO

1.1 Definição do Problema

O gerenciamento de mídia é um importante aspecto de smartphones. Com a crescente qualidade das câmeras, fabricantes de dispositivos móveis investem em funcionalidades avançadas para organização e backup de fotos e vídeos, como criação de álbuns, busca semântica e armazenamento em nuvem, frequentemente impulsionadas por modelos de Inteligência Artificial que analisam o conteúdo das mídias. No entanto, uma preocupação crescente entre os usuários é a privacidade e o controle de seus dados pessoais. Ao utilizar plataformas centralizadas, os usuários depositam sua confiança em terceiros, que detêm controle sobre as chaves de criptografia e, por consequência, ao conteúdo armazenado em nuvem. Essa falta de controle sobre as chaves de criptografia e a potencial exposição dos dados leventam questões sobre a segurança e privacidade dos dados amarzenados na nuvem.

1.2 Justificativa

Diante da preocupação da privacidade, segurança e o controle dos dados em plataformas de gerenciamento de mídia existentes, torna-se relevante a investigação e desenvolvimento de uma solução alternativa. A construção de uma biblioteca de imagens que permita aos usuários manter controle sobre as chaves de criptografia é uma resposta a essa demanda por maior privacidade. Ao mesmo tempo, essa plataforma pode se beneficiar da confiabilidade e escalabilidade dos provedores de serviços em nuvem, oferencendo o melhor de dois mundos: segurança, controle e privacidade para o usuário, aliado ao poder da infraestrutura em nuvem. Além disso, um sistema como o proposto pode abrir caminho para futuras explorações em funcionalidades de gerenciamento de mídia com foco na privacidade desde o design.

1.3 Objetivo Geral

Construir uma plataforma de gerenciamento de imagens com pesquisa semântica e armazenamento criptografado em nuvem, em que o usuário detém o controle sobre as chaves de criptografia.

1.4 Objetivos Específicos

Para este trabalho, os seguintes objetivos específicos foram levantados:

a) Possibilitar a pesquisa semântica de imagens no aplicativo Android, baseada nos metadados gerados pela análise de IA no servidor;

b) Implementar armazenamento criptografado em um serviço de nuvem, onde o usuário detém o controle das chaves de criptografia;

c) Analisar comparativamente o custo de diferentes serviços de armazenamento em nuvem relevantes para o projeto;

d) Possibilitar a deleção de imagens;

e) Sincronizar a biblioteca entre dispostivos.

1.5 Estrutura do Trabalho

Além da introdução, este trabalho está estruturado em cinco capítulos. O Capítulo 2 apresenta a fundamentação teórica, oferecendo embasamento sobre pesquisa semântica com inteligência artificial, armazenamento em nuvem e estratégias de sincronização. No Capítulo 3, é detalhada a metodologia adotada, bem como a arquitetura do sistema proposto, destacando a estratégia de sincronização e deleção utilizada, as bibliotecas e tecnologias selecionadas, além dos serviços em nuvem empregados. O Capítulo 4 traz a apresentação, análise e comparação dos resultados obtidos em pesquisas semânticas com plataformas existentes, além de uma avaliação comparativa entre diferentes serviços de armazenamento em nuvem e a solução de criptografia desenvolvida, discutindo também as consequências e limitações das soluções implementadas. Por fim, o Capítulo 5 sintetiza as conclusões extraídas dos resultados e comparações realizadas, oferecendo sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Existem diversas ferramentas de gerenciamento de imagens, tais como: Google Fotos, iCloud fotos, Ente fotos e OneDrive fotos. Essas ferramentas compartilham conceitos e funcionalidades comuns como: armazenamento em nuvem, pesquisa semântica (nem todos os serviços citados oferecem essa posibilidade), sincronização entre dispostivos, algum nível de criptografia. Entretanto, apesar dessas funcionalidades, uma preocupação crescente reside na privacidade e no controle dos dados pelos usuários. Em muitas dessas plataformas, o controle sobre as chaves de criptografia e a forma como os dados são armazenados e analisados permanece nas mãos das empresas provedoras.

Para entender a proposta e o desenvolvimento da plataforma de gerenciamento de imagens apresentada neste trabalho, os próximos tópicos abordarão os fundamentos de: pesquisa semântica baseada em inteligência artificial, explorando como a análise de conteúdo visual permite buscas mais intuitivas; o armazenamento em nuvem, discutindo suas vantagens e modelos de serviço e mencionando possíveis soluções como Google Cloud Storage, S3 e Backblaze; as estratégias de sincronização de dados entre dispositivos (com foco na abordagem pull-based adotada); e a importância da criptografia, com foco em soluções que conferem maior controle ao usuário, como a biblioteca Tink do Google. Também serão brevemente introduzidos conceitos relacionados às tecnologias utilizadas no desenvolvimento, como o framework de UI Jetpack Compose para o aplicativo Android e o framework Spring Boot para o backend.

2.1 Google Fotos

Google fotos é um aplicativo de gerenciamento e backup de mídia que está disponível em múltiplas plataformas. O principal fator competitivo do app é a pesquisa com linguagem natural de imagens. É possível pesquisar por lugares, pessoas, datas, objetos e momentos (GOOGLE, 2025b).

2.2 Pesquisa Semântica de Imagens

Pesquisa semântica é uma pesquisa por texto que foca em entender a intenção e contexto do termo pesquisado, em contraste a motores de busca mais convencionais que buscam por uma correspondência exata ou parcial com o termo de busca. Por exemplo, uma busca por "fruta vermelha" em um sistema de busca semântica poderia retornar resultados que incluem "morango", "cereja" e "framboesa", mesmo que essas palavras não estejam explicitamente presentes na consulta. A pesquisa semântica usa técnicas de processamento de linguagem natural (PLN) e aprendizado de máquina para entender o sentido do termo de busca e o conteúdo (CLOUD, 2025).

Em particular, o tipo de pesquisa semântica empregado neste trabalho é a pesquisa por vetores, que transforma tanto o texto como a imagem em uma representação numérica usando vetores em múltiplas dimensões . Esses vetores, por vezes chamados de embeddings, capturam as características semânticas do conteúdo. Modelos de Inteligência Artificial, como redes neurais convolucionais para imagens e modelos de linguagem para texto, são frequentemente utilizados para gerar essas representações vetoriais (CLOUD, 2025).

Esses vetores podem ser armazenados em um banco de dados otimizado para busca vetorial, como o Typesense. Dessa forma, um modelo de IA pode gerar vetores que representam tanto a imagem como também o texto da consulta e, a partir disso, calcular a similaridade de coseno entre os vetores (TYPESENSE, 2025b), indicando o quão semanticamente relacionados eles estão. Além disso, é possível usar outro modelo de IA, como o Google Cloud Vision (GCV) ou Amazon Rekognition, para gerar rótulos sobre o conteúdo de uma imagem, atingindo uma abordagem híbrida que permite uma pesquisa mais abrangente e relevante.

2.3 Docker

Docker é uma ferramenta de desenvolvimento e implantação que facilita o processo de desenvolver e compartilhar *builds* reproduzíveis (DOCKER, 2025).

Docker foi utilizado neste projeto para facilitar o desenvolvimento e orquestramento dos diferentes serviços que compõem a infraestrutura do servidor.

2.4 Spring Boot

Spring Boot é uma ferramenta para desenvolver servidores em Java ou Kotlin. Spring Boot fornece uma série de ferramentas e bibliotecas, incluindo injeção de dependências, integração com banco de dados SQL e autenticação (Spring Security) (SPRING, 2025).

A arquitetura RESTful para a comunicação entre o aplicativo Android e o servidor foi implementada utilizando os recursos do Spring Boot para a criação e exposição de endpoints da API

2.5 Serviços de Armazenamento

O gerenciamento eficiente de grandes volumes de dados, como bibliotecas de imagens, demanda soluções de armazenamento robustas e escaláveis. O armazenamento em nuvem é uma alternativa eficaz, oferecendo benefícios como escalabilidade, alta disponibilidade e durabilidade dos dados.

Dentre os modelos de serviço de computação em nuvem, o Object Storage (armazenamento de objetos) é particularmente relevante para o armazenamento de arquivos binários como fotos e vídeos. Plataformas como Google Cloud Storage, Amazon S3 e Backblaze fornecem esse tipo de serviço, permitindo o armazenamento de grandes quantidades de dados de forma flexível e acessível via APIs. Esses serviços geralmente adotam um modelo de preços baseado no volume total de dados armazenados, no tráfego de dados (upload e download) e no número de operações realizadas (criação, leitura, atualização e exclusão de objetos).

As soluções de Object Storage oferecem características importantes para sistemas de gerenciamento de mídia, como a facilidade de integração com APIs, mecanismos de controle de acesso para garantir a segurança dos dados e diferentes níveis de performance e redundância para atender a diversas necessidades.

2.6 Jetpack Compose

Jetpack Compose é um framework moderno e declarativo do Google para construir interfaces de usuário nativas no Android. Ele adota uma abordagem reativa, onde a UI é reconstruída em resposta a mudanças no estado, simplificando o desenvolvimento de interfaces complexas e dinâmicas (ANDROID, 2025).

Jetpack Compose fornece uma série de ferramentas para construção rápida de interfaces, incluindo soluções para paginação de dados, animações, listas performáticas e um sistema de design pronto que facilita a construção de uma marca.

2.7 Criptografia e a Biblioteca Tink

A criptografia é a conversão de dados de um formato legível para um formato codificado. Os dados criptografados só podem ser lidos depois de serem descriptagrafados (KASPERSKY, 2025). A biblioteca Tink permite dois tipos de criptografia: simétrica e assimétrica.

A principal diferença entre a criptografia simétrica e assimétrica é a quantidade de chaves: A simétrica fornece apenas uma; ao passo que a assimétrica usa dois chaves. Na criptografia assimétrica, um par de chaves é usado: uma chave privada (que deve ser mantida em sigilo) e uma chave pública (que pode ser compartilhada). Geralmente, a chave pública é usada para codificar o conteúdo, e apenas a chave privada correspondente pode descriptografá-lo. Alternativamente, a chave privada pode ser usada para assinar digitalmente um conteúdo, e a chave pública pode verificar a autenticidade dessa assinatura. (TOTVS, 2025)

A criptografia é essencial para proteção de dados na nuvem. Mesmo no evento de um vazamento de dados, as imagens do usuário não seriam comprometi-

das, uma vez que estariam codificadas. É de crucial importância manter a chave de criptografia segura e sob o controle do usuário.

2.7.1 Modelos de criptografia

Os três tipos mais comuns de modelo de criptografia usado em serviços são: Criptografia em repouso, trânsito e de ponta a ponta. Criptografia em repouso é aplicada nos dados armazenados, geralmente em um servidor. A criptografia é trânsito se refere à codificação que ocorre entre dois dispositivos em uma rede. É possível que os dados sejam armazenados descriptografados em ambos os dispositivos, porém a transfêrencia de dados em si é codificada. Por fim, a criptografia de ponta a ponta é realizada tanto no trânsito quanto em repouso. Normalmente, a fonte de dados codifica a informação antes mesmo de enviar ao servidor e, por isso, a mensagem está criptografada tanto em trânsito como em repouso. (RAZA, 2025)

2.7.2 Controle de Chaves pelo Usuário

Uma das desvantagens de serviços centralizados como Google Fotos e Google Drive é que as empresas detêm controle sobre as chaves de criptografia (SCHWENKE, 2024). Isso significa que, na prática, o conteúdo armazenado nesses serviços podem ser decodificados pelo Google.

Outras plataformas, como o iCloud Fotos, oferecem criptografia de ponta a ponta, mas isso não é garantia de segurança e privacidade, uma vez que governos podem passar leis que obrigam empresas a remover esse tipo de criptografia. Um caso que aconteceu recentemente envolve a Apple e o governo britânico, que forçou a Apple a remover a opção de Proteção Avançada de Dados do iCloud (DUFFY; EADICICCO, 2024).

Diante disso, é importante que o usuário tenha controle sobre as chaves de criptografia, o que protege os dados pessoais de qualquer agente externo, como agentes governamentais, empresas ou hackers, já que somente o proprietário das chaves pode decodificar os dados.

2.7.3 Biblioteca Tink

Tink é uma biblioteca de código aberto que fornece APIs de criptografia seguras e fáceis de usar, mantida por engenheiros de segurança no Google (GOOGLE, 2025d). Está disponível para múltiplos sistemas operacionais, incluindo Linux e Android, além de suportar Java, C++ e Go.

Tink também oferece uma ferramenta chamada Tinkey, que permite gerar chaves de criptografia. Dessa forma, o usuário retém controle sobre seus dados, desde que a chave seja armazenada de forma segura. Para isso, Tink oferece integrações

com serviços de gerenciamento de chaves populares, como Google Cloud KMS e AWS KMS. No entanto, se o usuário não quiser enviar suas chaves para terceiros, também é possível configurar uma chave localmente, de forma que nunca saia do controle do usuário.

Outra característica importante do Tink é a capacidade de escolher primitivos, que podem ser entendidos como algoritmos ou construções criptográficas eficientes para diferentes casos de uso, oferecendo diferentes níveis de segurança e desempenho. A página do Tink na internet oferece uma simples tabela para escolha do primitivo. Um dos primitivos disponível é o *Streaming AEAD*, voltado para criptografar dados grandes, como vídeos longos e imagens em alta resolução. Esse primitivo permite a construção de uma plataforma de gerenciamento de mídia eficiente.

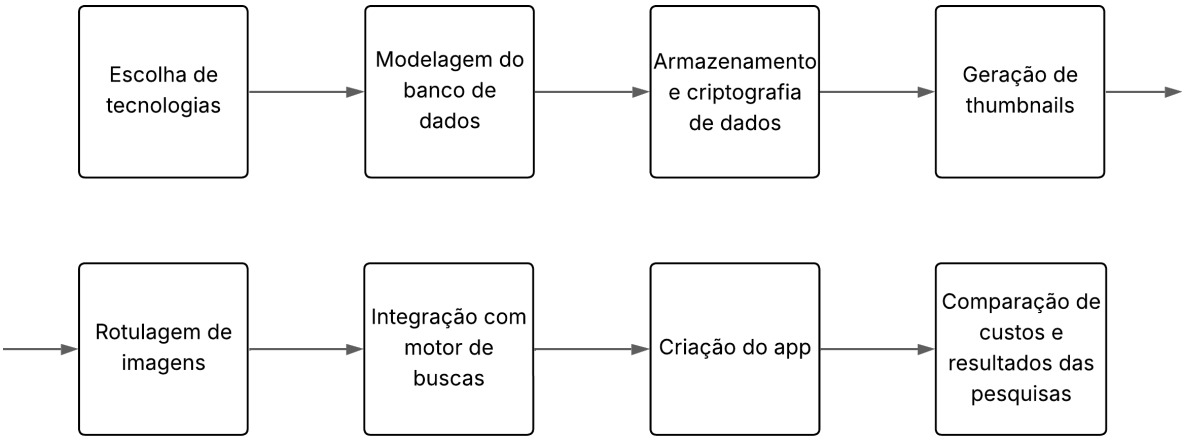
3 METODOLOGIA

A presente seção detalha a metodologia adotada para o desenvolvimento de um aplicativo protótipo de gerenciamento de imagens, concebido para simular funcionalidades essenciais de plataformas como o Google Fotos. Este protótipo visa demonstrar a viabilidade técnica e explorar desafios inerentes à organização, armazenamento e recuperação eficiente de grandes volumes de mídias digitais.

Para tanto, foram estabelecidas estratégias para abordar desafios como deduplicação de imagens, otimização da pesquisa e indexação automática da biblioteca do dispositivo, garantia de responsividade e velocidade da interface de usuário, e manutenção da sincronização e consistência dos dados entre os dispositivos e o servidor. Adicionalmente, este estudo contemplou a comparação de soluções para armazenamento de dados em nuvem, mecanismos de criptografia, serviços de rotulagem de imagens e motores de busca para pesquisa semântica.

Após a seleção das tecnologias, serviços e bibliotecas, o processo de desenvolvimento do protótipo foi iniciado de forma iterativa, seguindo uma sequência lógica de objetivos: modelagem do banco de dados, construção do serviço de autenticação, implementação do serviço de armazenamento de imagens, aplicação de criptografia de dados, extração de metadados, gerenciamento de autorização de recursos, criação e armazenamento de thumbnails, rotulagem de imagens, integração com o motor de busca e, por fim, o desenvolvimento do aplicativo cliente. Finalmente, procedeu-se à análise comparativa dos custos e dos resultados de desempenho do protótipo em relação a serviços de armazenamento de mídia consolidados no mercado. Na figura 1, é possível observar o fluxograma de como este trabalho será desenvolvido.

Figura 1 – Fluxograma da metodologia



Fonte: Elaborado pelo autor.

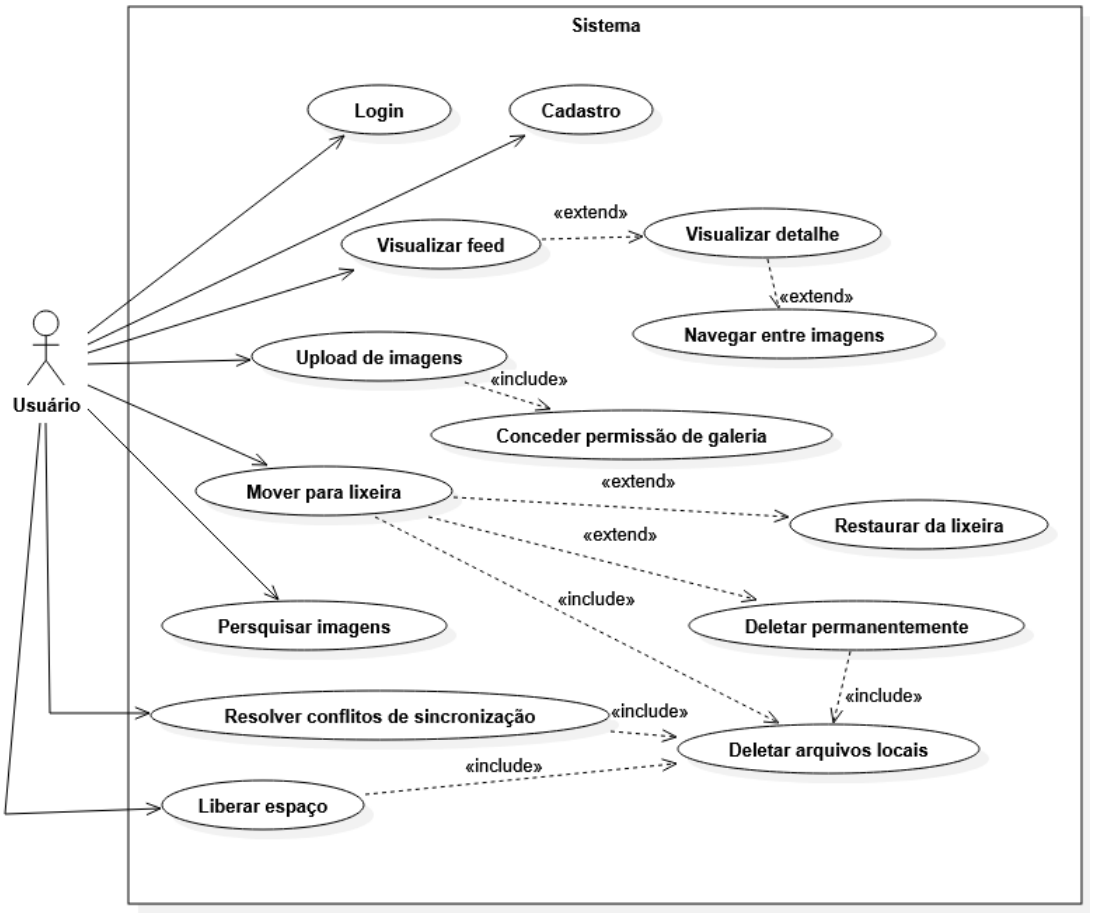
As próximas seções contêm alguns diagramas para facilitar o entendimento da arquitetura do sistema, começando com os diagramas de caso de uso, os quais

evidenciam as funcionalidades do sistema.

3.1 Diagrama de Caso de Uso

A figura 2 apresenta o diagrama de caso de uso do usuário, o qual pode criar registro na plataforma, fazer login, conceder permissão à galeria de imagens para realizar upload, visualizar o detalhe e navegar entre imagens arrastando para direita ou esquerda, mover para lixeira e deletar permanentemente ou restaurar um arquivo. Por fim, o usuário também pode pesquisar por imagens.

Figura 2 – Diagrama de caso de uso do usuário

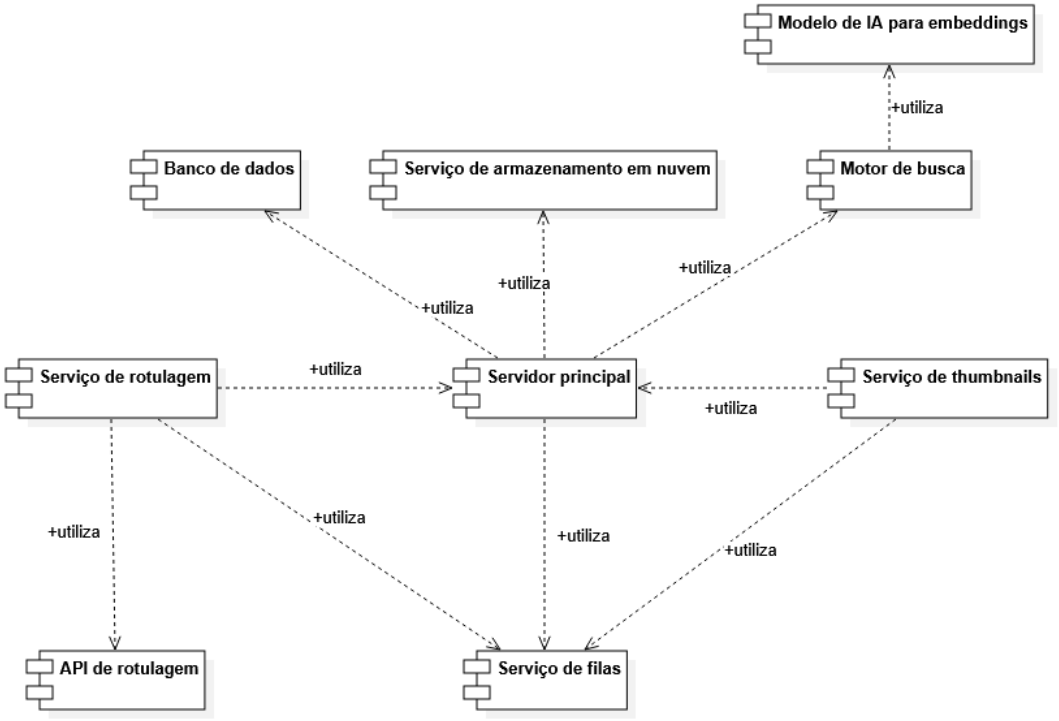


Fonte: Elaborado pelo autor.

3.2 Diagrama de Componentes

A figura 3 apresenta o diagrama de componentes do sistema. O servidor principal é responsável por orquestrar todo o sistema e também gerencia o banco de dados. Os serviços de rotulagem e thumbnail processam as requisições enviadas pelo serviço de filas, enquanto o servidor principal as dispara. O serviço de rotulagem utiliza uma API externa, o Google Cloud Vision.

Figura 3 – Diagrama de componentes

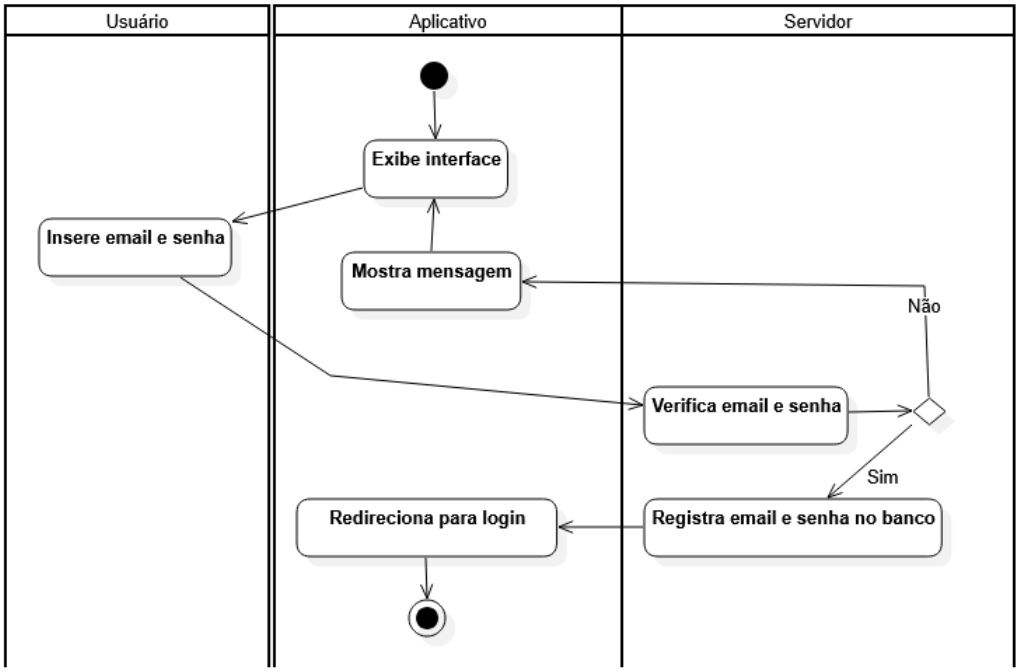


Fonte: Elaborado pelo autor.

3.3 Diagrama de Atividades

Nesta seção, serão apresentados os diagramas de atividades da plataforma. A figura 4 apresenta o diagrama de registro. O usuário pode fazer registro com email e senha, desde que o email não tenha sido usado por outro usuário. Se isso acontecer, o sistema mostra um erro. Caso o email ainda não exista na plataforma, o usuário é registrado no banco de dados e, em seguida, redirecionado à tela de login.

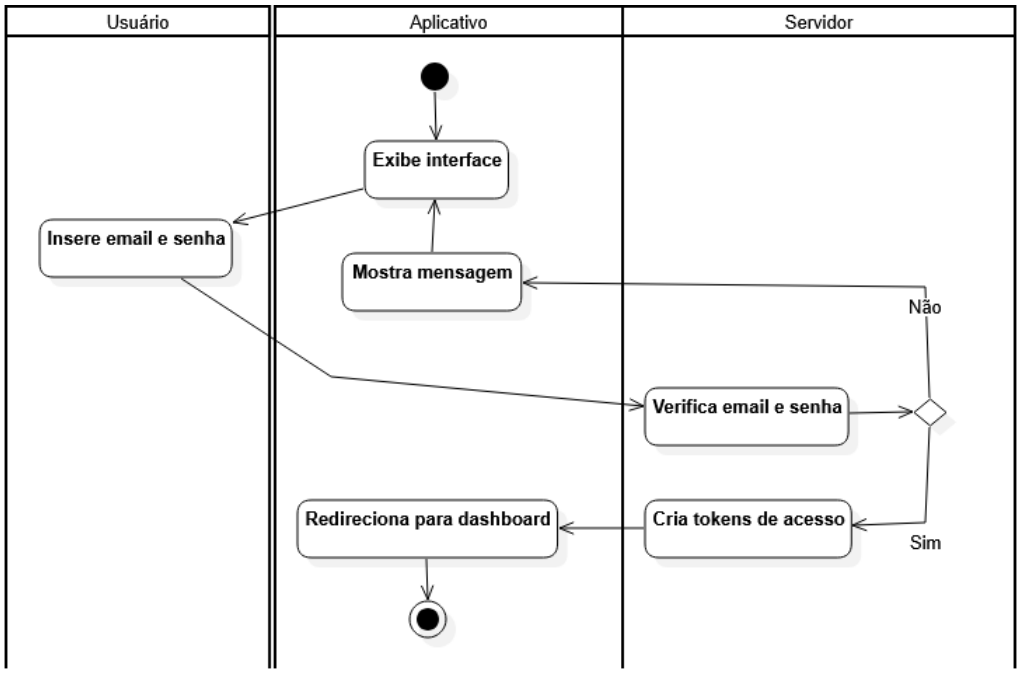
Figura 4 – Diagrama de atividades Registro



Fonte: Elaborado pelo autor.

A figura 5 apresenta o diagrama de atividades de login. O usuário insere o email e senha e, caso o email exista na plataforma e é senha for válida, é redirecionado à tela inicial. Caso contrário, o sistema mostra uma mensagem de erro.

Figura 5 – Diagrama de atividades Login

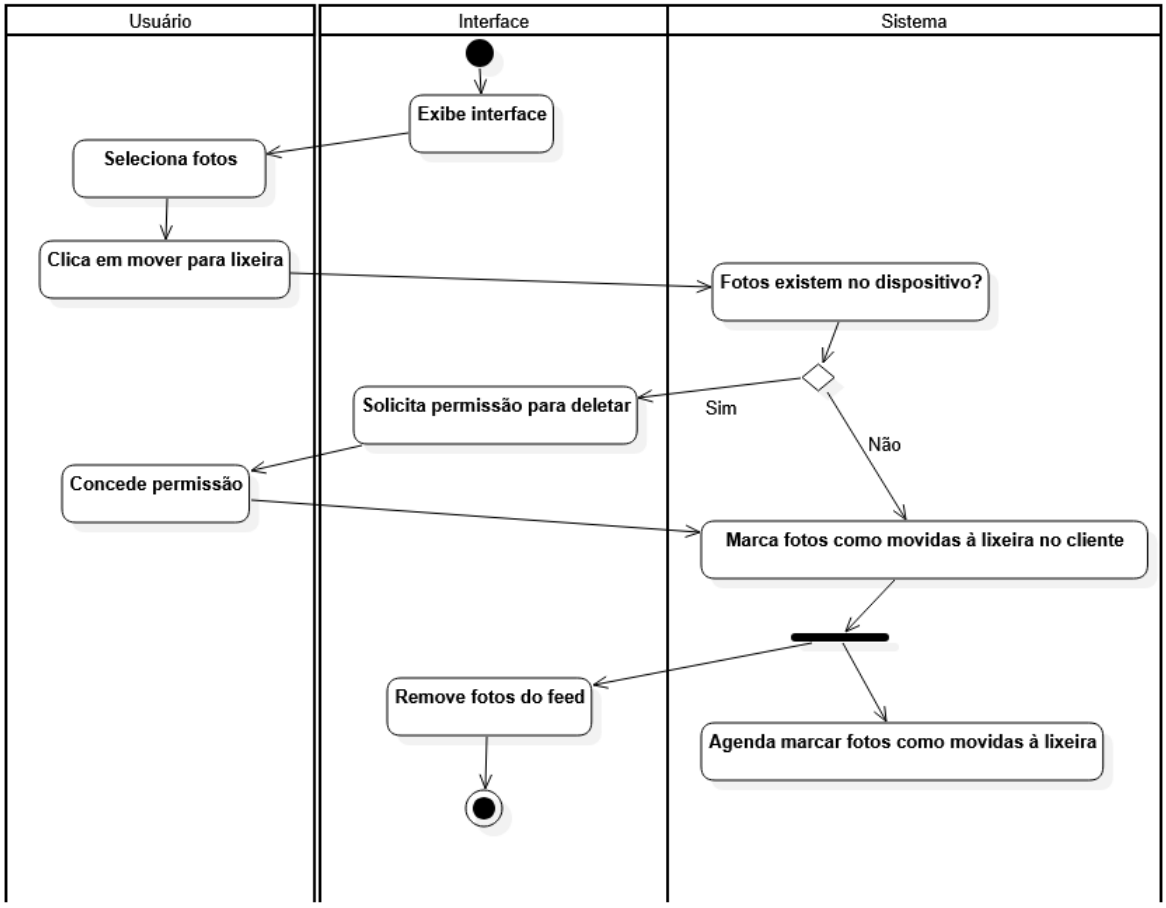


Fonte: Elaborado pelo autor.

A figura 6 apresenta o diagrama de atividades de mover para lixeira. O fluxo inicia quando o usuário seleciona uma ou mais fotos e clica no ícone de mover para

lixeira. Caso uma ou mais fotos ainda estejam armazenadas no dispositivo, o sistema pede permissão para deletar os arquivos locais, com intuito de economizar espaço no dispositivo, uma vez que as fotos já estão armazenadas no servidor. Em seguida, as fotos são marcadas como movidas para a lixeira no banco de dados local do dispotivo. Por fim, a interface é atualizada para remover as imagens do feed e, ao mesmo tempo, o cliente agenda um trabalho para sincronizar essa ação com o servidor.

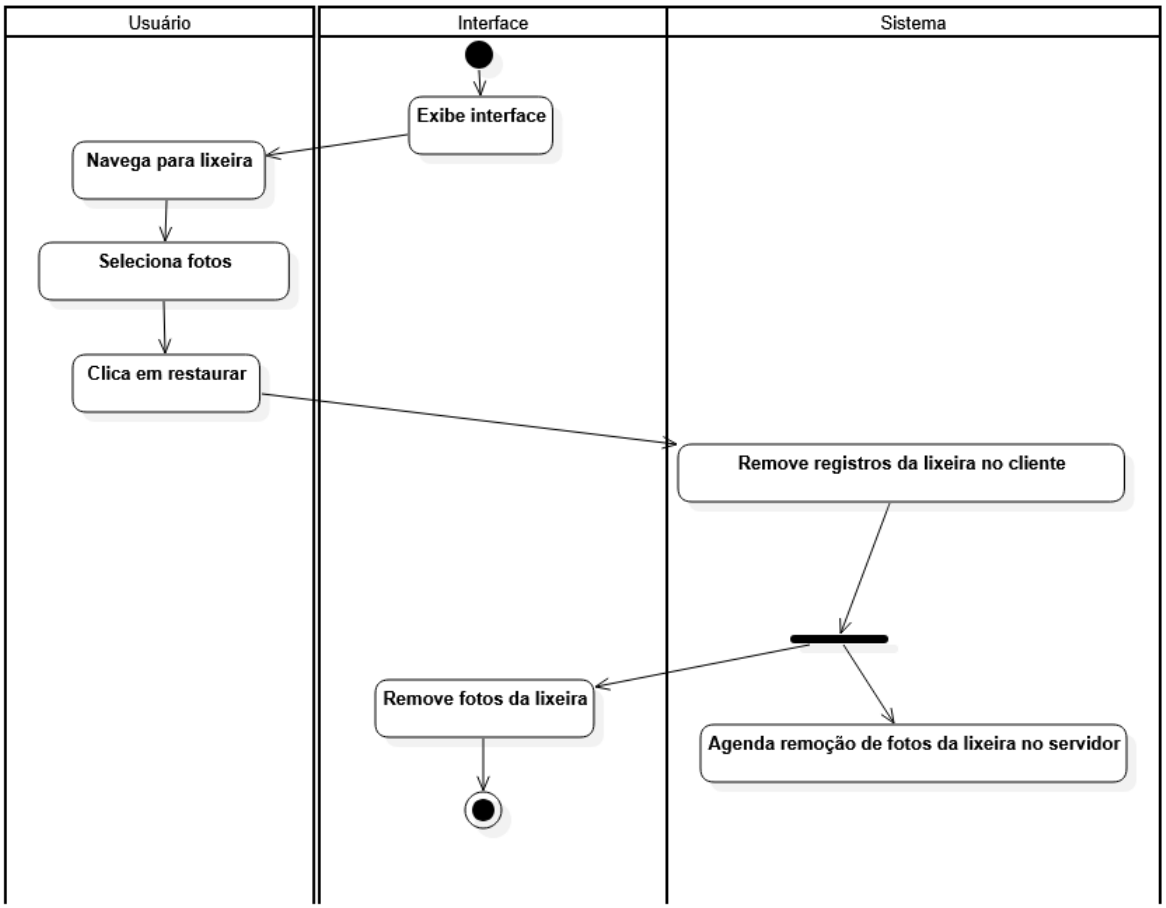
Figura 6 – Diagrama de atividades Mover para Lixeira



Fonte: Elaborado pelo autor.

A figura 7 apresenta o diagrama de atividades de restaurar da lixeira. Uma vez que uma imagem é movida para lixeira, usuários podem navegar até a pasta e selecionar fotos para restaurar.

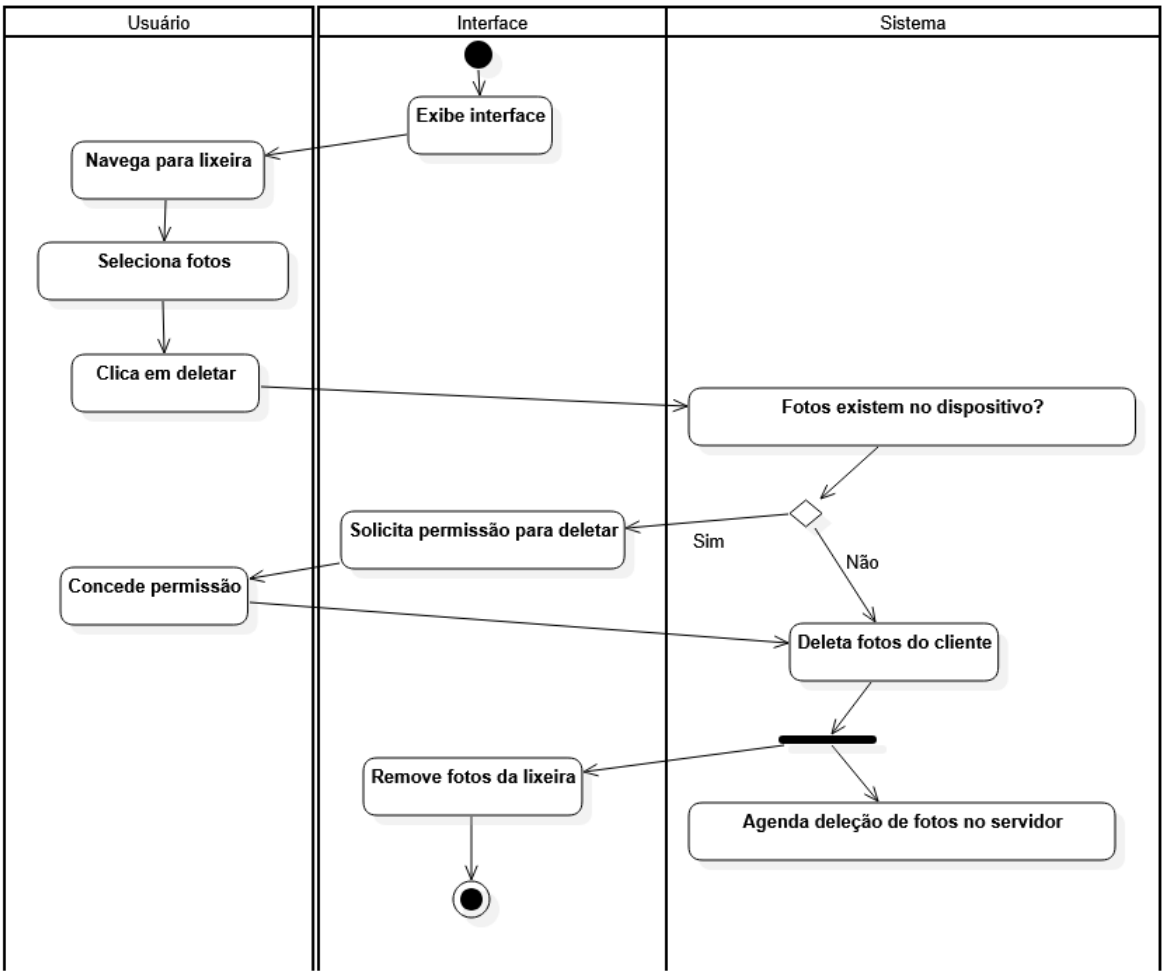
Figura 7 – Diagrama de atividades Restaurar da Lixeira



Fonte: Elaborado pelo autor.

A figura 8 apresenta o diagrama de atividades de deletar fotos permanentemente. Uma vez que uma imagem é movida para lixeira, usuários podem navegar até a pasta e selecionar fotos para deletar. Caso as imagens ainda estejam no dispositivo, o sistema vai pedir permissão para deletar.

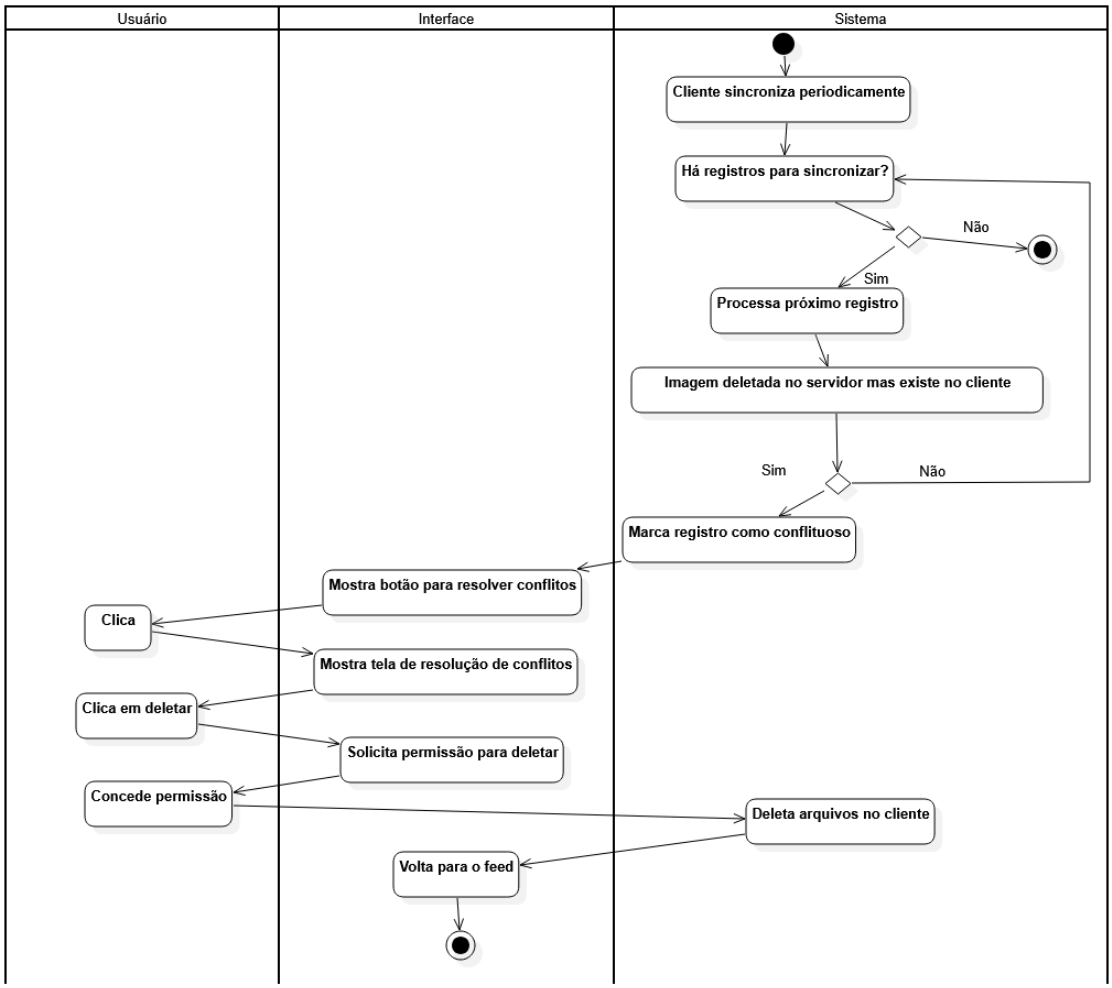
Figura 8 – Diagrama de atividades Deletar Permanentemente



Fonte: Elaborado pelo autor.

A figura 9 apresenta o diagrama de atividades de resolver problemas de sincronização. Se uma imagem for deletada em um dispositivo, e ainda estiver armazenada localmente em outro dispositivo, é necessário pedir permissão ao usuário para deletar a imagem no segundo cliente e, assim, resolver o problema de sincronização.

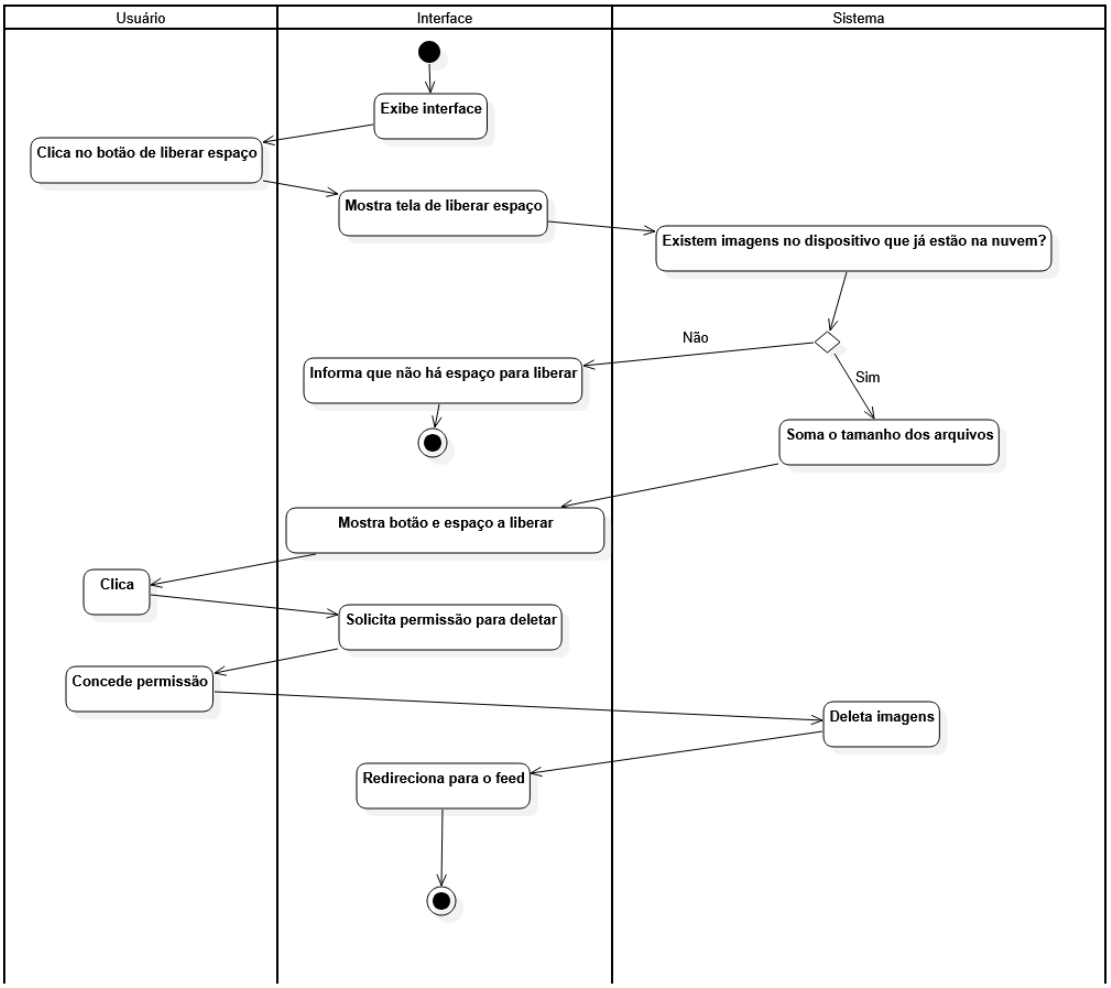
Figura 9 – Diagrama de atividades Resolver Conflitos de Sincronização



Fonte: Elaborado pelo autor.

A figura 10 apresenta o diagrama de atividades de liberar espaço. Essa funcionalidade permite ao usuário deletar imagens armazenadas no dispositivo que já estão na nuvem, dessa forma liberando espaço.

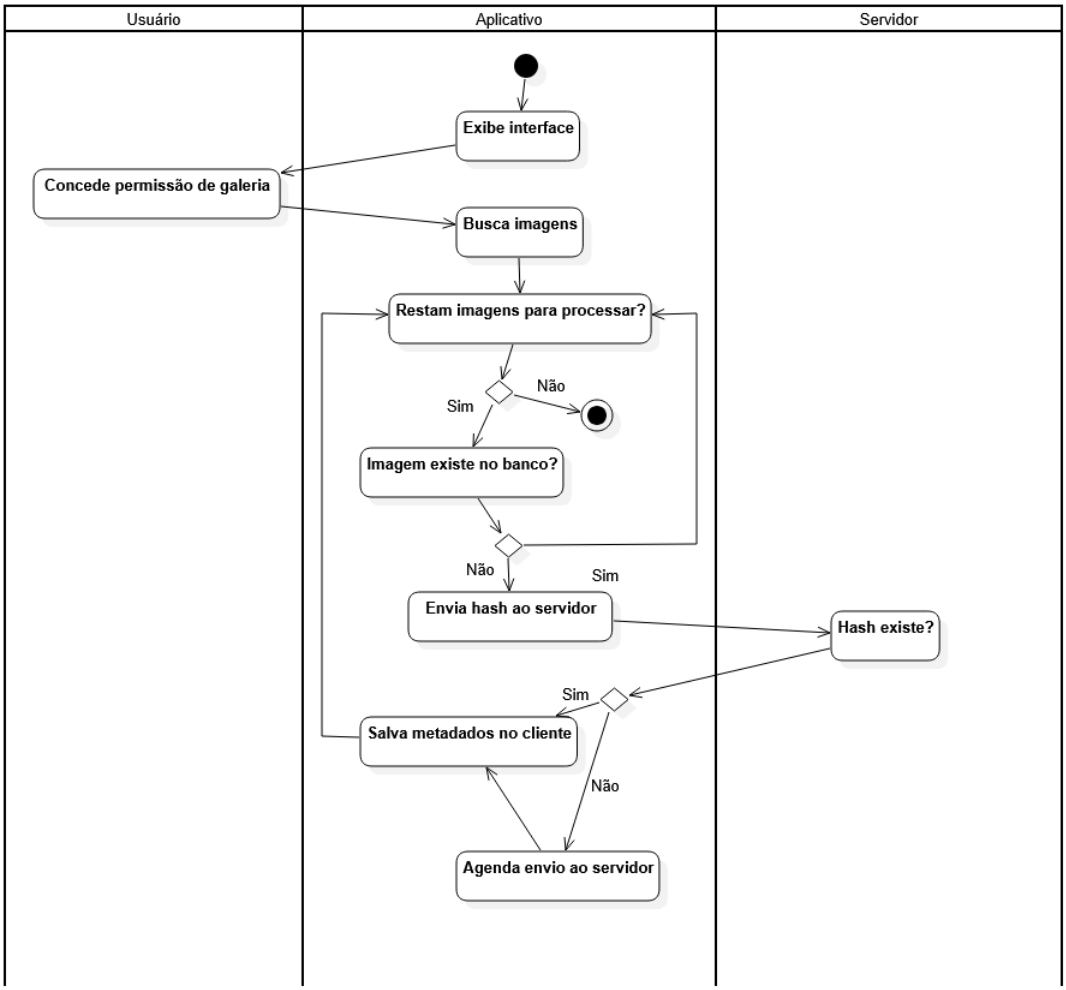
Figura 10 – Diagrama de atividades Liberar Espaço



Fonte: Elaborado pelo autor.

A figura 11 apresenta o diagrama de atividades de processar galeria. O fluxo começa quando o usuário concede permissão para explorar a galeria. Para cada imagem, é verificado se já está no banco de dados do aplicativo e também se já existe no servidor. Caso a imagem não exista no banco de dados do servidor, é agendado um trabalho para upload.

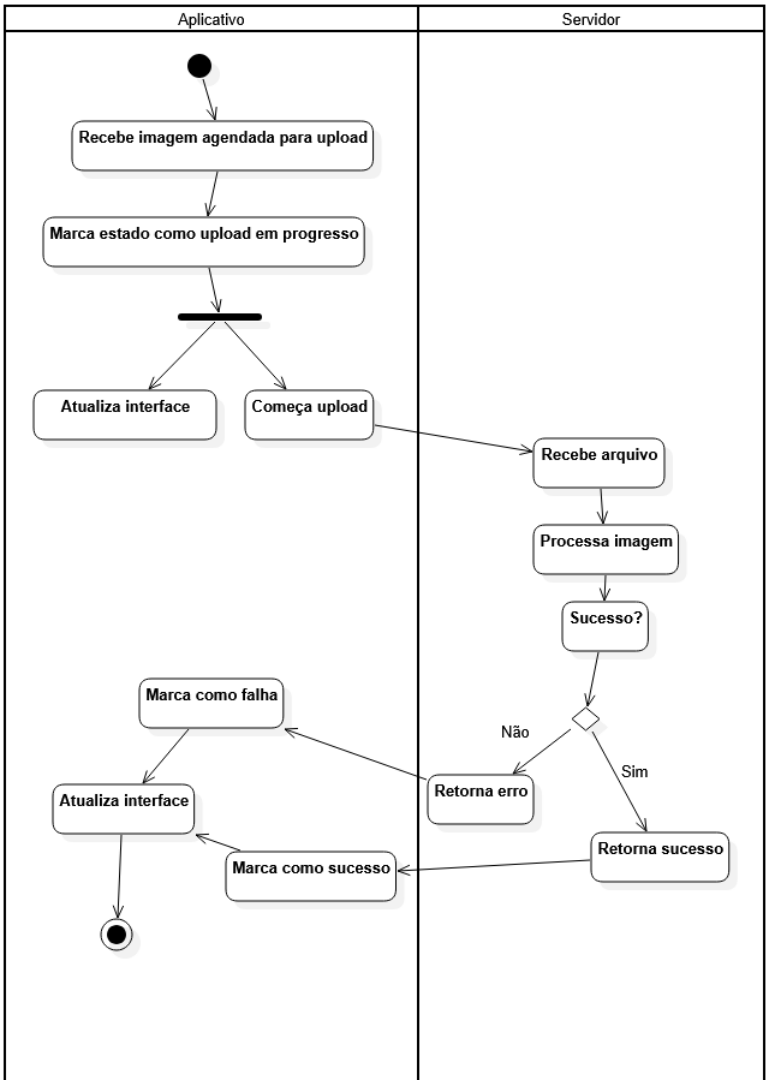
Figura 11 – Diagrama de atividades Processar Galeria



Fonte: Elaborado pelo autor.

A figura 12 apresenta o diagrama de atividades de upload de imagens, o qual demonstra a interação entre cliente e servidor.

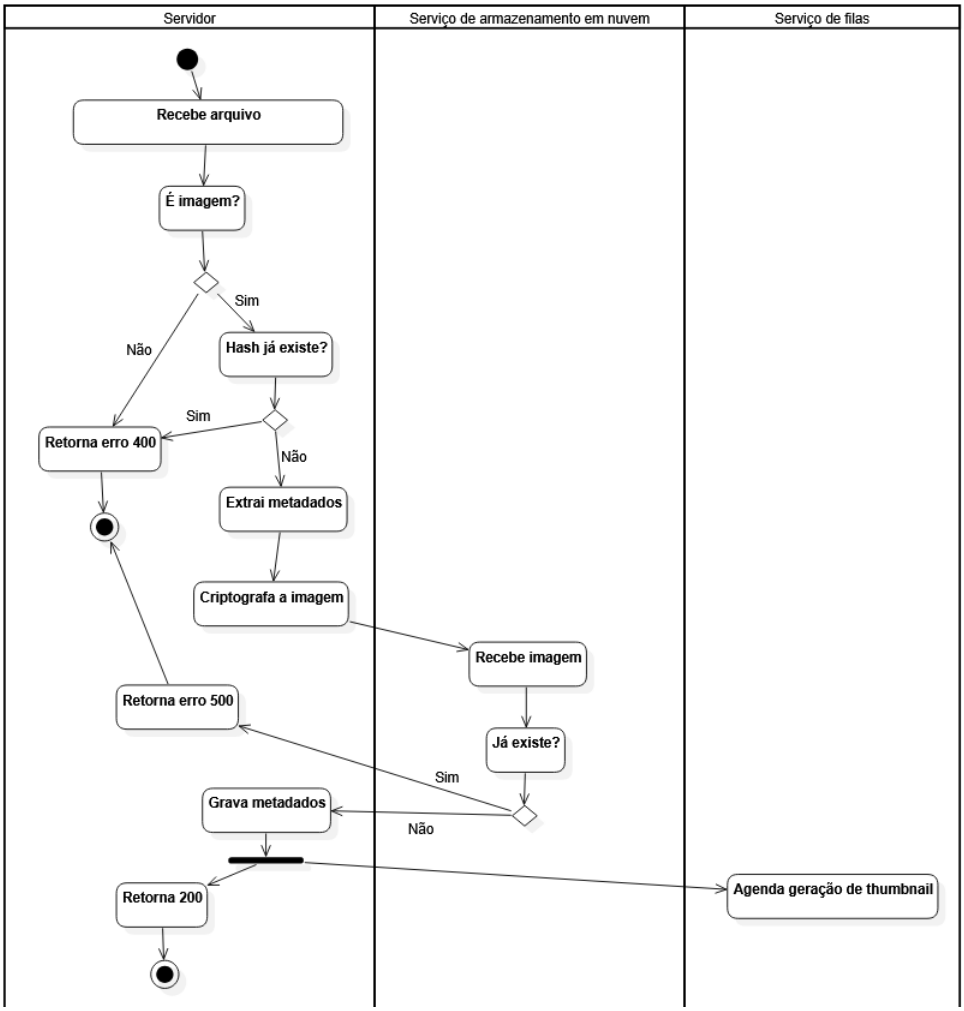
Figura 12 – Diagrama de atividades Upload de Imagens



Fonte: Elaborado pelo autor.

A figura 13 apresenta o diagrama de atividades de processamento de imagens no servidor. Vale observar que a geração de thumbnail é feita de forma assíncrona.

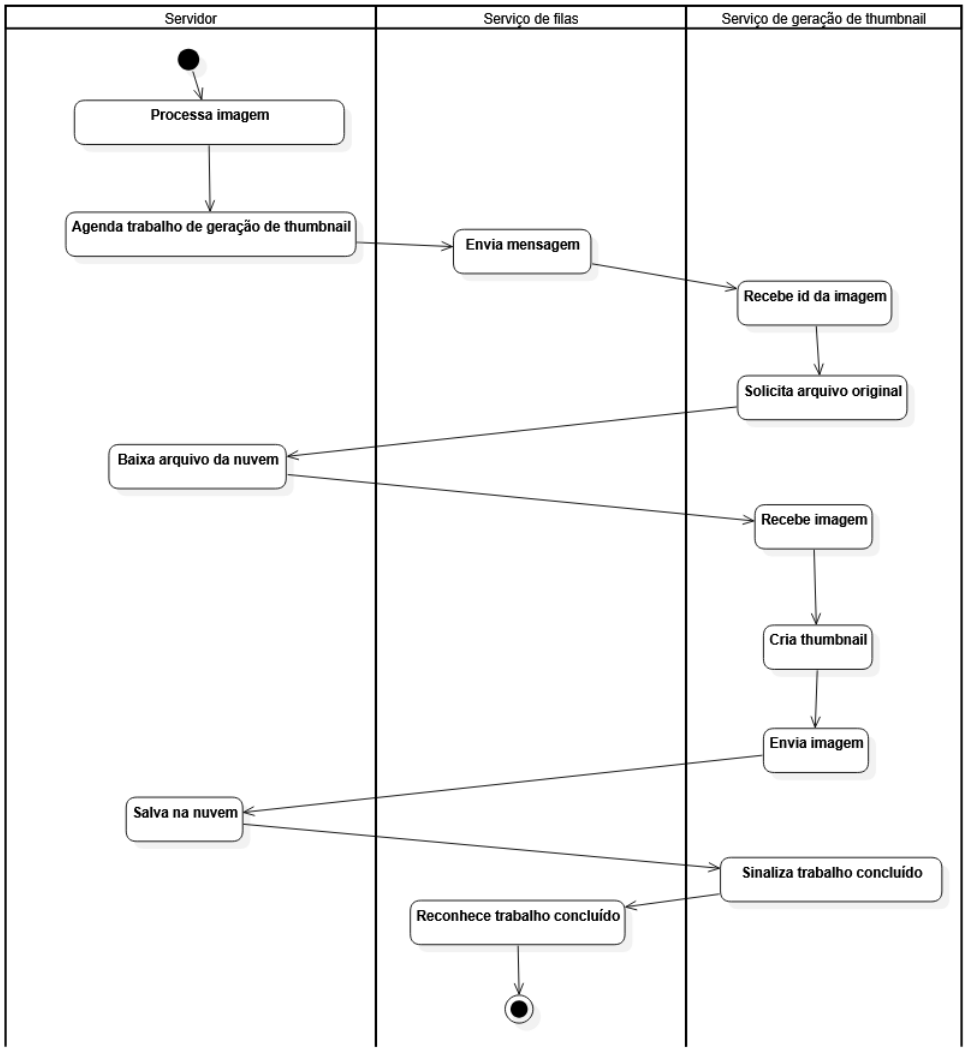
Figura 13 – Diagrama de atividades Processamento de Imagens



Fonte: Elaborado pelo autor.

A figura 13 apresenta o diagrama de atividades de geração de thumbnails, as quais serão utilizadas na tela inicial do aplicativo.

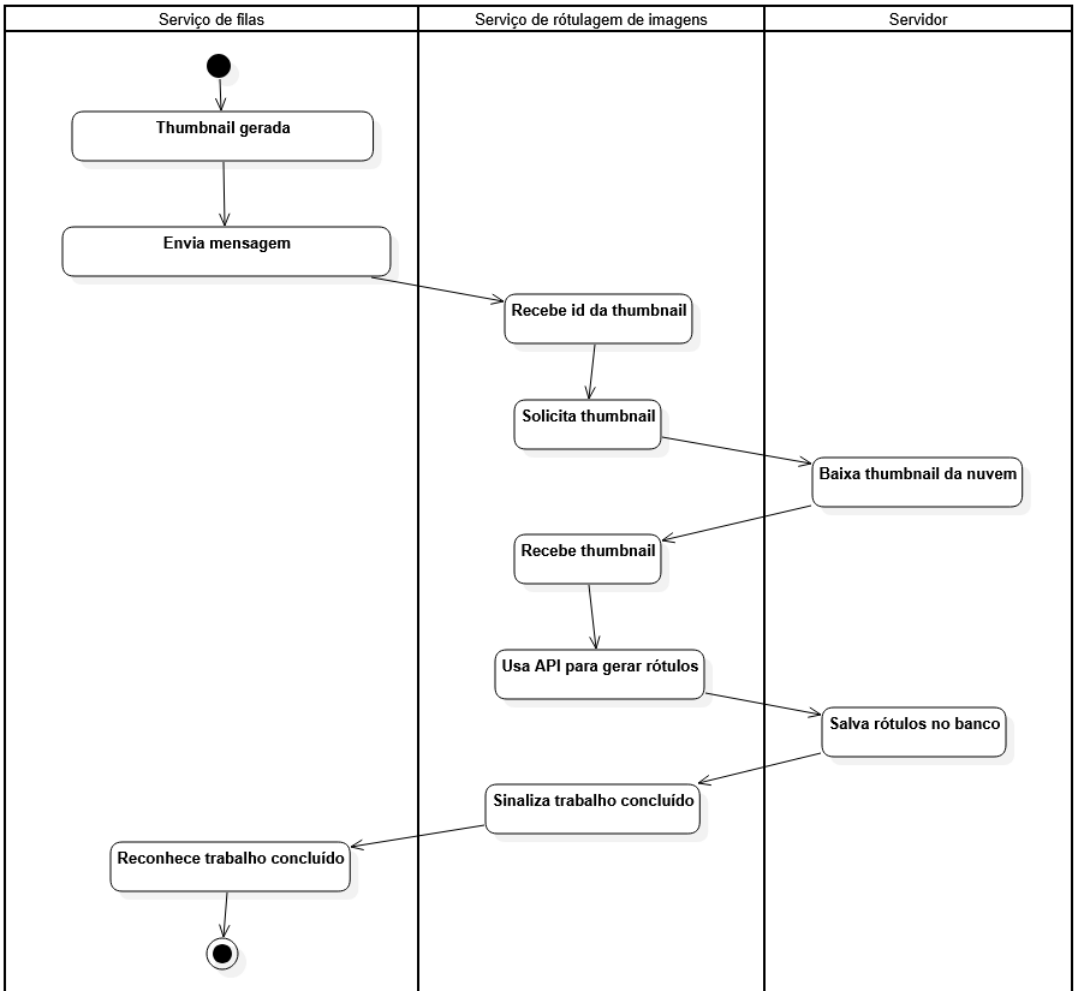
Figura 14 – Diagrama de atividades Geração de Thumbnail



Fonte: Elaborado pelo autor.

A figura 15 apresenta o diagrama de atividades de rotulagem de imagens. O serviço usa uma API externa para gerar os rótulos.

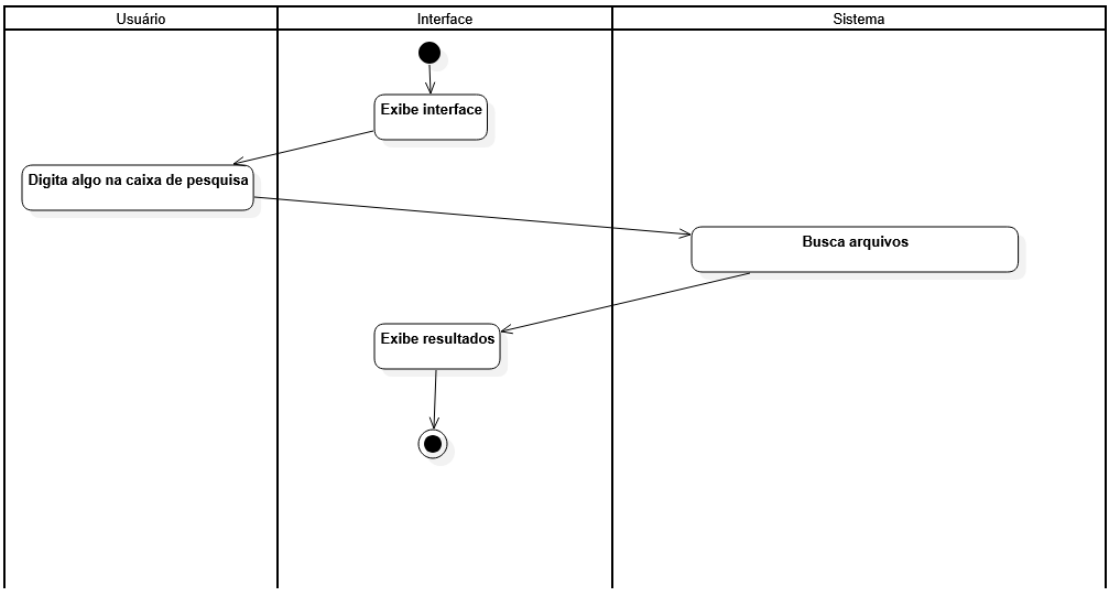
Figura 15 – Diagrama de atividades Geração de Rótulos



Fonte: Elaborado pelo autor.

A figura 16 apresenta o diagrama de caso de uso de pesquisar imagens. A busca leva em consideração tanto os rótulos gerados pelo modelo de IA como também os embeddings gerados a partir da imagem.

Figura 16 – Diagrama de atividades Pesquisar

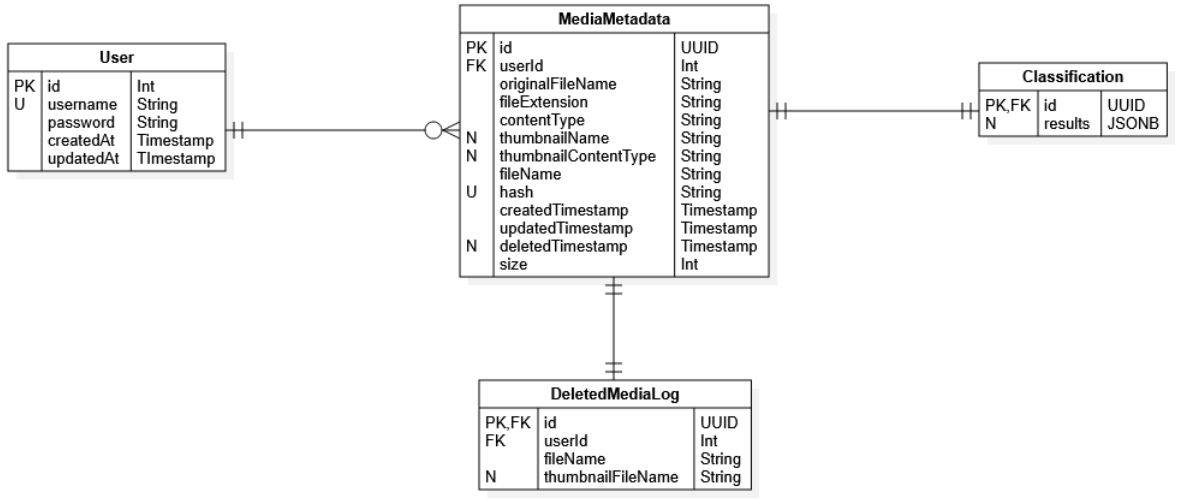


Fonte: Elaborado pelo autor.

3.4 Diagrama de Relação de Classes

Na Figura 17 é apresentado o diagrama de classes do servidor, evidenciando a estrutura de relacionamento entre seus componentes. Ao receber uma imagem, o servidor gera um registro do tipo MediaMetadata, que armazena todas as informações do arquivo. Ao mesmo tempo, para cada registro de MediaMetadata, é instanciado um objeto Classification, o qual utiliza uma chave estrangeira para referenciar o registro original. Cada objeto Classification contém um campo denominado results, que pode permanecer nulo ou ser preenchido com os rótulos produzidos pela inteligência artificial, armazenados no formato JSONB. Vale notar que o JSONB é um tipo de dado específico do PostgreSQL, capaz de armazenar e manipular documentos JSON de maneira eficiente (POSTGRESQL, 2025).

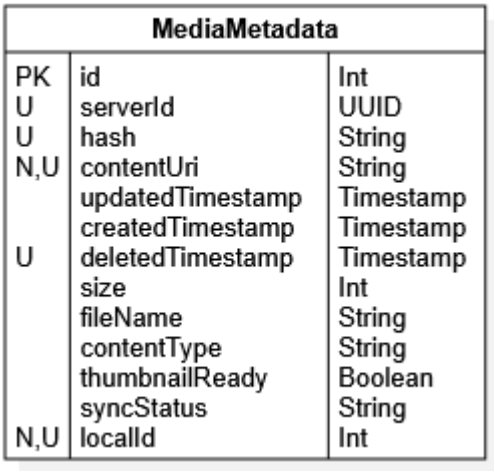
Figura 17 – Diagrama de relação de classes do servidor



Fonte: Elaborado pelo autor.

Na Figura 18 é apresentado o diagrama de classes do cliente Android, que segue a estrutura do diagrama do servidor, mas incorpora atributos específicos da plataforma. Em particular, os campos contentUri e localId indicam, respectivamente, a localização da imagem no sistema de arquivos do Android e a identificação única da imagem no ambiente local. Para gerenciar conflitos e monitorar o estado do upload, utiliza-se o campo syncStatus, que permite identificar se o envio está em progresso, se houve falhas ou se ocorreram conflitos com o servidor. Por fim, o campo thumbnailReady, originado no servidor, informa se o serviço assíncrono de geração de thumbnails já produziu a imagem em miniatura.

Figura 18 – Diagrama de relação de classes do cliente



Fonte: Elaborado pelo autor.

3.5 Etapas de Desenvolvimento

Nesta seção, serão apresentadas as etapas de desenvolvimento do projeto.

3.5.1 Escolha de Tecnologias

Para a construção do servidor, foi escolhido a framework Spring Boot, que atua como o componente central da infraestrutura, em conjunto com o banco de dados PostgreSQL. A escolha dessa combinação se deu principalmente pela facilidade de integração do Spring Boot com PostgreSQL e outros bancos de dados, permitindo a interação com o banco sem a necessidade de escrever código SQL diretamente, otimizando o desenvolvimento.

Para a geração de thumbnails, um simples script em Python foi integrado ao serviço de filas RabbitMQ, garantindo um processamento assíncrono e eficiente. De forma similar, um serviço em Python foi empregado para interagir com a Google Cloud Vision API, responsável pela geração de rótulos (labels) para as imagens. A documentação abrangente e os exemplos de uso fornecidos pela Google Cloud Vision API foram fatores determinantes para sua escolha neste projeto.

No que tange à criptografia, foi empregado a biblioteca Tink. Um de seus diferenciais, além da compatibilidade com Java, é a capacidade de rotacionar chaves e criptografar grandes volumes de arquivos de forma eficiente. Dada a necessidade de um sistema de gerenciamento de fotos lidar com múltiplas imagens simultaneamente, a performance do Tink para essa tarefa se mostrou ideal.

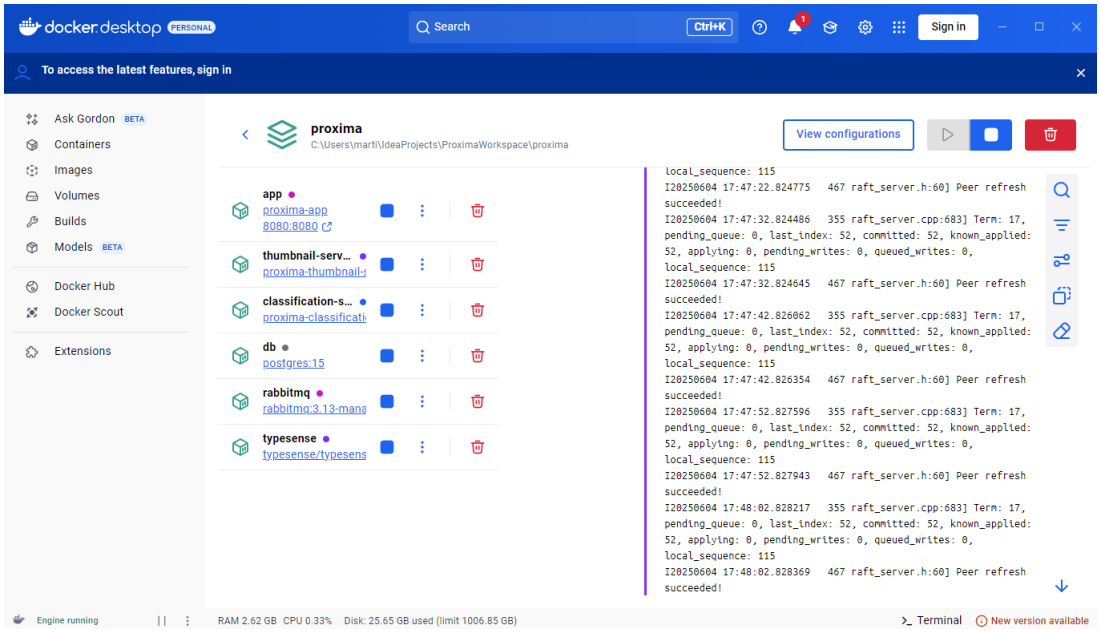
Para o armazenamento em nuvem, foram avaliados dois serviços: Google Cloud Storage (GCS) e Backblaze. Embora o Backblaze apresente um custo inicial atraente (U$6 por TB com transferência gratuita até 3TB), o GCS demonstrou maior potencial de otimização de custos a longo prazo. O GCS cobra U$0,026 por gigabyte (GB) para armazenamento com redundância em múltiplas regiões (GOOGLE, 2025a), totalizando aproximadamente U$26,63 por terabyte (TB), sem incluir custos de transferência e operações. No entanto, o GCS oferece a ferramenta de Gerenciamento do Ciclo de Vida de Objetos (GOOGLE, 2025c). Com ela, foi possível implementar uma regra para alterar a classe de objetos armazenados por mais de um ano para a classe Archive, que custa consideravelmente menos (U$0,0024 por GB, ou cerca de U$2,46 por TB/mês). Essa otimização de armazenamento permite atingir um custo similar ao do Backblaze. Além disso, o Google Cloud Storage fornece documentação robusta e bibliotecas prontas para desenvolvimento em Java e Spring Boot, facilitando a integração, o que o tornou a escolha ideal para o projeto.

O Typesense foi selecionado como o motor de buscas, destacando-se pela facilidade de uso e pela integração com modelos de Inteligência Artificial (IA) pré-construídos. O Typesense oferece diversos modelos que podem ser configurados com esforço mínimo (TYPESENSE, 2025a), o que se alinha perfeitamente com a necessidade de busca semântica do sistema. Elastic Search também foi outra solução considerada, entretanto, apesar de sua robustez, demandaria uma configuração mais

complexa e customização de modelos de IA, o que fugiria do escopo do TCC em comparação com a simplicidade do Typesense.

Por fim, para simplificar o desenvolvimento e a configuração de toda a arquitetura, a ferramenta Docker foi fundamental. A Figura 19 ilustra todos os serviços em funcionamento no ambiente Docker.

Figura 19 – Docker



Fonte: Elaborado pelo autor.

3.5.2 Modelagem do Banco de Dados

Nesta etapa, foram modelados os bancos de dados do servidor e do cliente Android. Para garantir boa performance no cliente, foram criados três indexes no banco de dados: Um index para o hash, outro composto que envolve as colunas created_timestamp e deleted_timestamp e, por fim, um no campo local_id. Os indexes em local_id e hash servem para consultas rápidas e eficientes de imagens. Quando o usuário tira uma foto, por exemplo, o aplicativo verificará se essa nova imagem existe no banco ao comparar esses dois campos. O index composto em created_timestamp e deleted_timestamp é crucial para a listagem eficiente das imagens na página inicial, uma vez que o aplicativo usa o deleted_timestamp para verificar se a imagem está na lixeira e created_timestamp para sortear as imagens por data de criação. A pasta da lixeira também se beneficia desse mesmo index. O código 3.1 mostra como foi configurado o banco de dados no cliente Android. É possível notar que há três ids na tabela: id, serverId e localId. Cada um deles se refere à algo diferente. O primeiro id é o identificador desse registro no banco do Android. O segundo, serverId, representa o id da imagem no servidor. Note que o tipo é opcional, marcado com interrogação, pois uma imagem que existe no dispositivo não necessariamente vai existir no servidor,

pois é preciso fazer o upload da imagem. Por fim, o localId se refere ao identificador no banco de dados do sistema operacional. Esse identificador também é opcional, já que uma imagem que existe no servidor não necessariamente vai existir no sistema de arquivos do cliente.

```
@Entity(
    tableName = "media_metadata",
    indices = [
        Index(value = ["hash"], unique = true),
        Index(value = ["created_timestamp", "deleted_timestamp"], unique = false),
        Index(value = ["local_id"], unique = true)
    ]
)
@TypeConverters(RoomConverters::class)
data class MediaMetadata(
    @PrimaryKey(autoGenerate = true)
    val id: Int? = null,
    @ColumnInfo(name = "server_id")
    val serverId: String?,
    @ColumnInfo(name = "hash")
    var hash: String?,
    @ColumnInfo(name = "content_uri")
    val contentUri: String?,
    @ColumnInfo(name = "updated_timestamp")
    val updatedTimestamp: Instant,
    @ColumnInfo(name = "created_timestamp")
    val createdTimestamp: Instant,
    @ColumnInfo(name = "deleted_timestamp")
    val deletedTimestamp: Instant?,
    @ColumnInfo(name = "size")
    val size: Long,
    @ColumnInfo(name = "file_name")
    val fileName: String,
    @ColumnInfo(name = "content_type")
    val contentType: String,
    @ColumnInfo(name = "thumbnail_ready")
    val thumbnailReady: Boolean,
    @ColumnInfo(name = "sync_status")
    val fileSyncStatus: FileSyncStatus,
    @ColumnInfo(name = "local_id")
    val localId: Int? = null
)
```

Código 3.1 – Banco de Dados no Android

FileSyncStatus representa os diferentes estados de sincronização do arquivo. O código 3.2 ilustra os possíveis valores. Em especial, os dois últimos estados são de grande importância para funcionalidade de resolver conflitos. Quando uma imagem é movida para lixeira em um dispositivo, todos os outros dispositivos que contêm a imagem receberão um aviso de conflito, pedindo ao usuário que dê permissão para deletar a imagem. De modo semelhante, se uma imagem for permanentemente deletada em um dispositivo, também será preciso pedir permissão para deletar em cada dispositivo que tiver a imagem armazenada localmente.

```
enum class FileSyncStatus {
    PENDING,
    UPLOADING,
    SYNCED,
    FAILED,
    TRASHED_PENDING_LOCAL_DELETE_PERMISSION,
    HARD_DELETED_PENDING_LOCAL_DELETE_PERMISSION
}
```

Código 3.2 – Estados de Sincronização

O código 3.3 apresenta a tabela de metadados no servidor. Nessa tabela, há dois índices compostos: idx_metadata_user_hash e idx_metadata_user_dlts_crts, que desempenham funções semelhantes às da tabela do cliente. O primeiro índice é utilizado para verificar se o usuário já fez upload de uma foto, comparando o hash, enquanto o segundo índice é empregado na construção da página inicial e da lixeira, onde as fotos são organizadas com base na data de criação.

```
create table media_metadata (
    created_timestamp timestamp(6) with time zone not null,
    deleted_timestamp timestamp(6) with time zone,
    file_size bigint not null,
    updated_timestamp timestamp(6) with time zone not null,
    user_id bigint not null,
    file_extension varchar(10) not null,
    id uuid not null,
    content_type varchar(255) not null,
    file_name varchar(255) not null,
    image_hash varchar(255) not null,
    original_file_name varchar(255) not null,
    thumbnail_content_type varchar(255),
    thumbnail_name varchar(255),
    primary key (id)
);

create index idx_metadata_user_dlts_crts
   on media_metadata (user_id, deleted_timestamp, created_timestamp);

create index idx_metadata_user_hash
   on media_metadata (user_id, image_hash);

alter table if exists media_metadata
   add constraint FKncmlnysijotv1omj9kqdinsgv
   foreign key (user_id)
   references users;
```

Código 3.3 – Tabela MediaMetadata no Servidor

O código 3.4 apresenta a tabela que armazena os rótulos de cada imagem. É possível observer que a tabela image_classification possui como identificador primário o id da tabela media_metadata. Além disso, no Spring Boot foi configurado a relação de cascata entre as tabelas: Quando um registro de media_metadata é deletado, a classification equivalente em image_classification também é deletada.

```
create table image_classification (
    media_metadata_id uuid not null,
    classification_results jsonb,
    primary key (media_metadata_id)
);

alter table if exists image_classification
   add constraint FK6mlrg0odm4ss216egf3m3p50u
   foreign key (media_metadata)
   references media_metadata;
```

Código 3.4 – Tabela de Rótulos

O código 3.5 apresenta a tabela de usuários.

```
create table users (
    created_at timestamp(6) with time zone not null,
    id bigint generated by default as identity,
    updated_at timestamp(6) with time zone not null,
    password varchar(255) not null,
    username varchar(255) not null,
    primary key (id),
    constraint idx_user_username unique (username)
);
```

Código 3.5 – Tabela de Usuários

O código 3.6 apresenta a tabela de metadados permanentemente deletados. Clientes podem consultar essa tabela para verificar quais arquivos foram deletados.

```
create table deleted_media_log (
    deleted_timestamp timestamp(6) with time zone not null,
    user_id bigint not null,
    id uuid not null,
    file_name varchar(255) not null,
    thumbnail_name varchar(255),
    primary key (id)
);

alter table if exists deleted_media_log
   add constraint FKnyw7po291hdl70qlvsvoleij
   foreign key (user_id)
   references users;
```

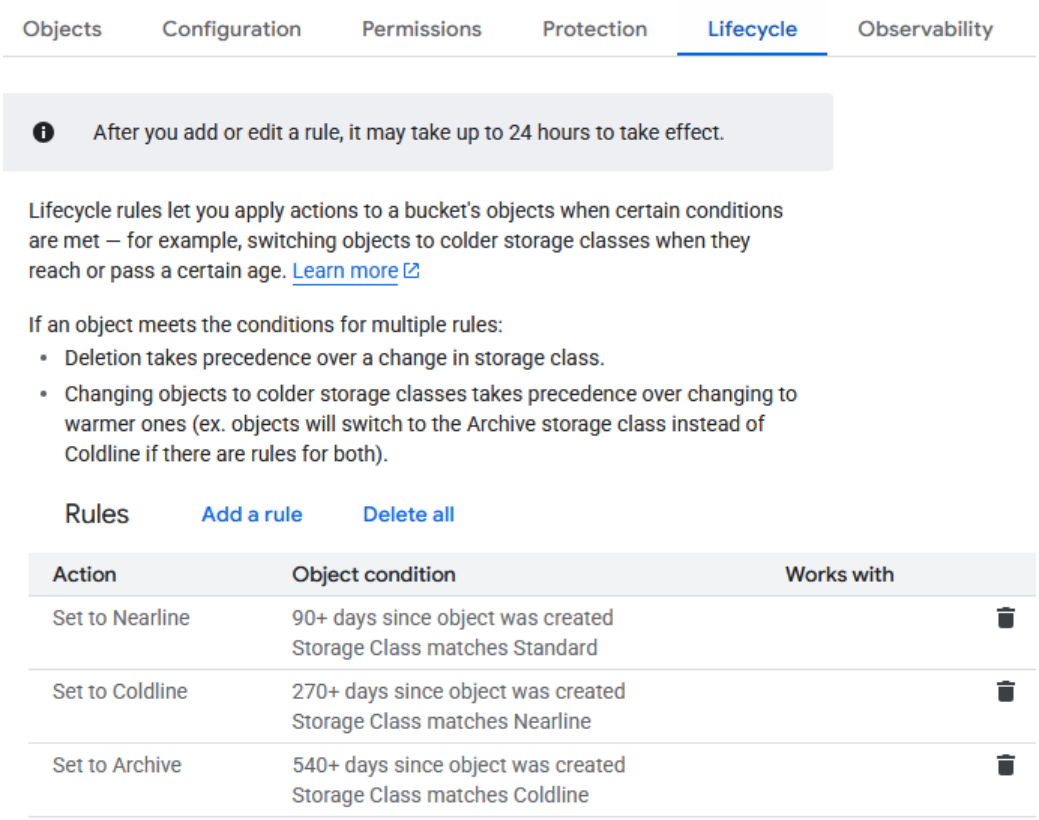
Código 3.6 – Tabela de Metadados Deletados

3.5.3 Armazenamento de Dados

Nesta etapa, foi criada toda a infraestrutura necessária para armazenagem de dados na nuvem. Para isso, foi criado um projeto no Google Cloud Console. Em seguida, foi ativado e configurado o serviço de armazenamento em nuvem Cloud Storage. Em particular, é de grande importância a configuração do ciclo de vida de objetos. A figura 20 mostra como foi configurado o ciclo de vida das imagens. Quando

uma imagem é armazenado, por padrão sua class é configurado como Standard, o tipo mais caro e de menor latência. Após 90 days, a classe muda para Nearline e, após 270 dias, para Coldline. Por fim, após 540 dias, a classe é configurada para Archive, a mais barata porém com maior latência.

Figura 20 – Configuração do Ciclo de Vida



Fonte: Elaborado pelo autor.

Após a configuração do Cloud Storage, foi criado um endpoint na aplicação Spring Boot para receber imagens. No código 3.7 é possível observar o endpoint, que aceita como parâmetros a data de criação, modificação e o próprio arquivo.

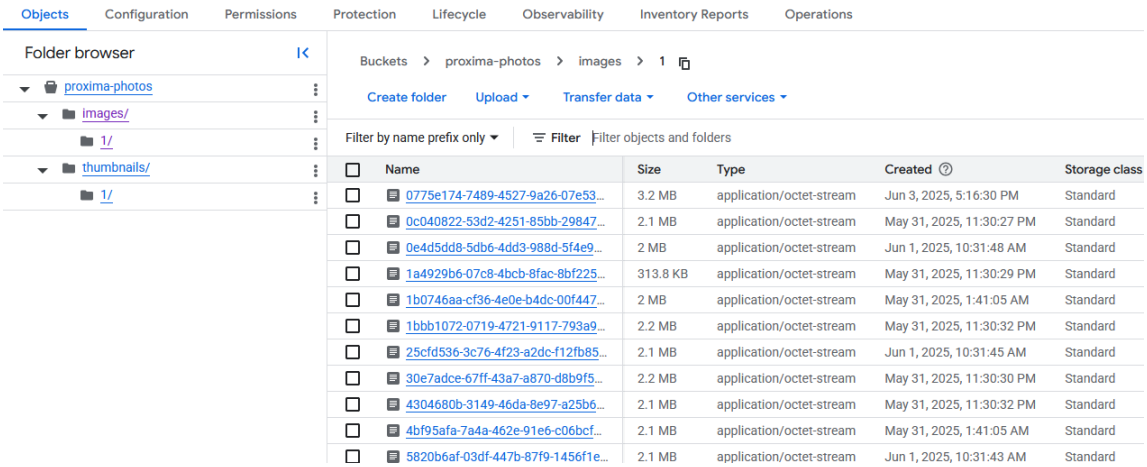
```
@RestController
@RequestMapping("/api/v1/images")
class ImageStorageController(
    private val imageStorageService: ImageStorageService,
) {
    @PostMapping
    fun uploadImage(
        @AuthenticationPrincipal jwt: Jwt,
        @RequestParam("file") file: MultipartFile,
        @RequestParam("updatedTimestamp") updatedTimestamp: Instant,
        @RequestParam("createdTimestamp") createdTimestamp: Instant,
    ) : ResponseEntity<UploadImageResponseDTO> {
        if (file.isEmpty) {
            throw InvalidFileException("File is empty")
        }
```

```
16
17          val savedMetadata = imageStorageService.storeImage(
18              userId = jwt.userId,
19              file = file,
20              updatedTimestamp = updatedTimestamp,
21              createdTimestamp = createdTimestamp
22          )
23
24          val response = UploadImageResponseDTO(
25              id = savedMetadata.id!!,
26              uniqueFileName = savedMetadata.fileName
27          )
28          return ResponseEntity.ok(response)
29      }
30  }
```

Código 3.7 – Endpoint de Upload de Imagens

O método storeImage do serviço ImageStorageService extrai o hash da imagem, valida se já existe no banco e, caso não exista, procede com o upload para o GCS, além de extrair e salvar metadados. A figura 21 apresenta como ficam guardados os dados no GCS. Na imagem, proxima-photos é o nome do bucket. Também é possível observar a classe de cada imagem na coluna "Storage class". Vale notar que existe uma pasta chamada thumbnails no canto esquerdo da imagem. Esta pasta está relacionada a etapa de geração de thumbnails.

Figura 21 – Imagens armazenadas no GCS



Fonte: Elaborado pelo autor.

3.5.4 Criptografia

Nesta etapa, foi implementado a criptografia de imagens. O primeiro passo foi configurar a biblioteca no ambiente do Spring Boot. O código 3.8 mostra a configuração da dependência tink no projeto. O tipo retornado é um objeto StreamingAead, capaz de codificar arquivos de modo eficiente, sem carregar a imagem interia na memória, algo essencial para reduzir consumo de memória.

```
@Bean
    fun aead(
        @Value("\${tink.streaming.aead.keyset.path}")
        keysetResource: Resource
    ): StreamingAead {
        try {
            TinkConfig.register()

            StreamingAeadConfig.register()

            val keysetJsonString = keysetResource.inputStream.use { inputStream ->
                inputStream.readBytes().toString(StandardCharsets.UTF_8)
            }

            val handle = TinkJsonProtoKeysetFormat.parseKeyset(
                keysetJsonString,
                InsecureSecretKeyAccess.get()
            )

            return handle.getPrimitive(RegistryConfiguration.get(), StreamingAead::
    class.java)

        } catch (e: GeneralSecurityException) {
            throw RuntimeException("Failed to initialize AEAD primitive", e)
        } catch (e: IOException) {
            throw RuntimeException("Failed to read keyset file", e)
        }
    }
```

Código 3.8 – Configuração da Biblioteca Tink

Vale notar a presença de um argumento chamado keysetResource. Essa é a chave que será utilizada na codificação das imagens. É possível gerar uma chave usando a ferramenta de linha de comando Tinkey. O Código 3.9 mostra o comando necessário para gerar a chave.

```
tinkey create-keyset \
  --key-template AES256_GCM_HKDF_1MB \
  --out-format JSON \
  --out keyset.json
```

Código 3.9 – Comando para Gerar a Chave

Com a biblioteca Tink configurada, foi criado uma interface para criptografar e descriptografar arquivos, como mostra o código 3.10.

```
interface CryptoManager {

    fun encrypt(inputStream: InputStream, associatedData: ByteArray):
    PipedInputStream

    fun decrypt(inputStream: InputStream, associatedData: ByteArray):
    PipedInputStream
}
```

Código 3.10 – Interface para criptografia

A partir disso, foi possível usar essa interface para criptografar a imagem antes de enviar ao serviço em nuvem, como mostra o código

```
private fun saveFile(
    file: MultipartFile,
    path: String
) {
    val encryptedStream = cryptoManager.encrypt(
        inputStream = file.inputStream,
        associatedData = path.toByteArray()
    )

    objectStorageService.writeObject(
        bucket = properties.bucket,
        inputStream = encryptedStream,
        path = path
    )
}
```

Código 3.11 – Código para salvar imagem na nuvem

3.5.5 Geração de thumbnails

Nesta etapa, foi desenvolvida a infraestrutura para geração de thumbnails, incluindo: Configuração do serviço de filas RabitMQT, criação do script de Python, criação de um endpoint para baixar imagens e thumbnails e criação de um endpoint para salvar thumbnails. Inicialmente, foi configurado o serviço de fila no ambiente Docker. Em seguida, foi definido duas filas: Uma para geração de thumbnail e outra para rotulagem de imagens. Com as filas definidas, foi criado um script em Python que recebe o identificador de uma imagem. Uma vez que o serviço em Python recebe esse id, é preciso fazer o download original do arquivo para gerar a thumbnail. Para isso, foi criado um endpoint no servidor principal que baixa arquivos do serviço em nuvem. Com o arquivo em mãos, o serviço Python é capaz de gerar a thumbnail, como mostra o código 3.12.

```
with Image.open(io.BytesIO(params.bytes)) as img:
    if img.mode not in ("RGB", "L"):
        img = img.convert("RGB")

    img.thumbnail(THUMBNAIL_SIZE)

    output_buffer = io.BytesIO()
    output_format = "webp"

    img.save(output_buffer, format=output_format, quality=80, optimize=True)

    return output_buffer.getvalue()
```

Código 3.12 – Código para Gerar Thumbnail

Com a thumbnail gerada, foi preciso criar um endpoint para salvar a thumbnail no servidor principal, como mostra o código 3.13.

```
@PostMapping("/thumbnail")
fun uploadThumbnail(
    @RequestParam("file") file: MultipartFile,
    @RequestParam("userId") userId: Long,
    @RequestParam("metadataId") metadataId: UUID,
    @RequestParam("contentType") contentType: String
): ResponseEntity<Void> {
    if (file.isEmpty) {
        throw InvalidFileException("File is empty")
    }

    imageStorageService.storeThumbnail(
        userId = userId,
        file = file,
        metadataId = metadataId,
        contentType = contentType
    )

    return ResponseEntity.noContent().build()
    }
```

Código 3.13 – Código para Salvar Thumbnail

3.5.6 Rotulagem

Nesta etapa, foi ativado o serviço Google Cloud Vision API no console do Google Cloud, configurado o serviço Python junto ao serviço de filas e criado os endpoints necessários no serviço principal. O código 3.14 mostra a configuração do cliente do Cloud Vision. Em especial, é possível notar 3 tipos de detecção: Rótulos (LABEL_DETECTION), objetos (OBJECT_LOCALIZATION) e propriedades de imagens (IMAGE_PROPERTIES), como as cores donminantes, por exemplo.

```
image = vision.Image(content=params.image_bytes)

features = [
    vision.Feature(type_=vision.Feature.Type.LABEL_DETECTION, max_results=10),
    vision.Feature(type_=vision.Feature.Type.OBJECT_LOCALIZATION, max_results=10),
    vision.Feature(type_=vision.Feature.Type.IMAGE_PROPERTIES, max_results=2)
]

response = vision_client.batch_annotate_images(requests=[request])

image_response = response.responses[0]
```

Código 3.14 – Configuração da Rotulagem de Imagens

Em seguida, são extraídos da resposta os três tipos de detecção solicitados, a começar pelos rótulos, que são armazenados como tipo LABEL. Depois, são extraídos os objetos, que são armazenados como tipo OBJECT. Por fim, das propriedades da imagem, são extráidos as cores dominantes, que são armazenadas como tipo COLOR, como mostra o código

```python
all_annotations = []

labels = image_response.label_annotations
for label in labels:
    all_annotations.append(
        ClassificationPrediction(
            label=label.description,
            confidence=round(float(label.score), 4),
            type="LABEL"
        )
    )

objects = image_response.localized_object_annotations
for obj in objects:
    all_annotations.append(
        ClassificationPrediction(
            label=obj.name,
            confidence=round(float(obj.score), 4),
            type="OBJECT"
        )
    )

colors = image_response.image_properties_annotation.dominant_colors.colors
for color in colors:
    label = rgb_to_color_label(color)
    if label:
        all_annotations.append(
            ClassificationPrediction(
                label=label,
                confidence=round(float(color.score), 4),
                type="COLOR"
            )
        )

return ClassificationResponse(annotations=all_annotations)
```

Código 3.15 – Extração dos Dados Produzidos pelo Modelo de IA

Com os resultados extraídos, foi criado um endpoint no servidor principal para receber e associar os dados com entre a tabela MediaMetadata e a Classification, como mostra o código 3.16

```kotlin
@PostMapping("/{metadataId}/classify")
fun classify(
    @PathVariable metadataId: UUID,
    @RequestBody request: UpdateClassificationInternalDTO
): ResponseEntity<Map<String, UUID>> {
    val saveMetadataId = imageClassificationService.classify(
        metadataId = metadataId,
        request = request
    )

    return ResponseEntity.ok(
        mapOf("id" to saveMetadataId)
    )
}
```

Código 3.16 – Endpoint para Gravar o Resultado da Análise do Modelo de IA

3.5.7 Motor de Busca

Nesta etapa, as seguintes atividades foram desenvolvidas: Configuração do serviço Typesense, sincronização dos rótulos entre o banco de dados e o Typesense e criação de um endpoint para pesquisa de imagens. Foi utilizado o modelo ts/clip-vit-b-p32 para criar os embeddings das imagens, modelo pronto e pré-configurado disponibilizado pelo Typesense. Para armazenar os rótulos gerados pelo serviço em Python, foi alterado o endpoint classify para incluir a sincronização com o Typesense, como mostra o código 3.17. Por fim, foi criado um novo endpoint para pesquisar imagens.

```
@PostMapping("/{metadataId}/classify")
fun classify(
    @PathVariable metadataId: UUID,
    @RequestBody request: UpdateClassificationInternalDTO
): ResponseEntity<Map<String, UUID>> {
    val saveMetadataId = imageClassificationService.classify(
        metadataId = metadataId,
        request = request
    )

    typesenseSyncService.syncClassificationForMetadata(
        metadataId = metadataId,
        dto = request
    )

    return ResponseEntity.ok(
        mapOf("id" to saveMetadataId)
    )
}
```

Código 3.17 – Endpoint classsify Alterado para Syncronizar com Typesense

3.5.8 Endpoints para Manipular Imagens

Nesta etapa, foram criados diversos endpoints para manipular imagens, incluindo: Mover para lixeira, restaurar da lixeira, deletar permanentemente e listar metadados de forma paginada. O Código 3.18 mostra um exemplo de como foi implementado a lógica de restaurar da lixeira.

```
@Transactional
fun restoreTrashed(
    ids: List<UUID>,
    userId: Long
) : List<UUID> {
    val media = repository.findAllByIdInAndUserIdAndDeletedTimestampIsNotNull(
        ids = ids,
```

```
8           userId = userId
9       )
10
11      val foundIds = media.mapNotNull { it.id }
12
13      val updatedMedia = media.map {
14          it.updatedTimestamp = Instant.now(clock)
15          it.deletedTimestamp = null
16          it
17      }
18
19      repository.saveAll(updatedMedia)
20
21      return foundIds
22  }
```

Código 3.18 – Lógica de Restaurar Imagens da Lixeira

3.5.9 Criação do Aplicativo Android

Nesta etapa, foi desenvolvido o aplicativo Android com as seguintes funcionalidades: login, cadastro, feed de imagens, visualização de imagens em tela cheia, pasta da lixeira, tela de pesquisa, tela de resolução de conflitos e funções para mover para a lixeira, restaurar e excluir permanentemente.

As telas de login e cadastro possuem campos para email e senha, além de mensagens de erro específicas. Caso o usuário insira uma senha incorreta ou um email não cadastrado, uma notificação será exibida informando que as credenciais estão incorretas.

O feed de imagens foi estruturado em um sistema de grades organizadas por data de criação de cada foto. Além disso, ele se adapta ao tamanho da tela do dispositivo. Em um tablet, por exemplo, as imagens aparecem em um formato maior e o número de colunas na grade é ajustado. Cada imagem no feed exibe um indicador de status de sincronização, informando se a sincronização falhou ou está em andamento. Nessa tela, é possível selecionar e mover imagens para a lixeira.

A tela de detalhes foi projetada para exibir a imagem em alta resolução. Ao selecionar uma miniatura, o aplicativo anima a imagem até que fique em tela cheia. Em seguida, o sistema inicia o download de uma versão em melhor resolução e, assim que o processo for concluído, atualiza automaticamente a tela de detalhes com a nova imagem.

A pasta de lixeira foi construída reutilizando boa parte da tela de feed, com a diferença de que as imagens não são separadas por data. Além disso, a lixeira possui as funções de restaurar, que levarão a imagem de volta ao feed, e excluir permanentemente.

A tela de pesquisa foi desenvolvida reaproveitando parte da estrutura do

feed, com a adição de uma barra de pesquisa e algumas animações extras. Assim como a lixeira, ela não organiza as imagens por data. Os resultados da busca excluem conteúdos da pasta de lixeira. Além da função de pesquisa, a tela também permite selecionar e mover imagens para a lixeira.

A tela de resolução de conflitos foi criada para reagir em tempo real a problemas de conflitos na biblioteca. Quando um conflito é detectado, uma caixa de diálogo é exibida na parte inferior da tela. Ao clicar nela, o usuário é levado a uma nova tela que exibe as imagens conflituosas, sem organização por data. Um botão foi adicionado para resolver os conflitos.

Por fim, após a construção da interface, foram criados diversos casos de uso para gerenciamento da biblioteca, incluindo mover para a lixeira, restaurar, excluir permanentemente, resolver e detectar conflitos, sincronizar periodicamente etc. O código 3.19 apresenta um exemplo do caso de uso de mover imagens para a lixeira.

```
class TrashFilesById(
    private val fileRepository: FileRepository,
    private val trashFilesScheduler: TrashFilesScheduler,
    private val dispatcherProvider: DispatcherProvider
) {
    suspend fun execute(ids: List<FileId>) = withContext(dispatcherProvider.io) {
        val files = fileRepository.findAllById(ids)

        launch {
            fileRepository.trashAll(files)
        }

        val serverIds = files.mapNotNull { it.identity.serverId }

        if (serverIds.isNotEmpty()) {
            trashFilesScheduler.scheduleRemoteTrash(serverIds)
        }
    }
}
```

Código 3.19 – Caso de Uso de Mover para Lixeira

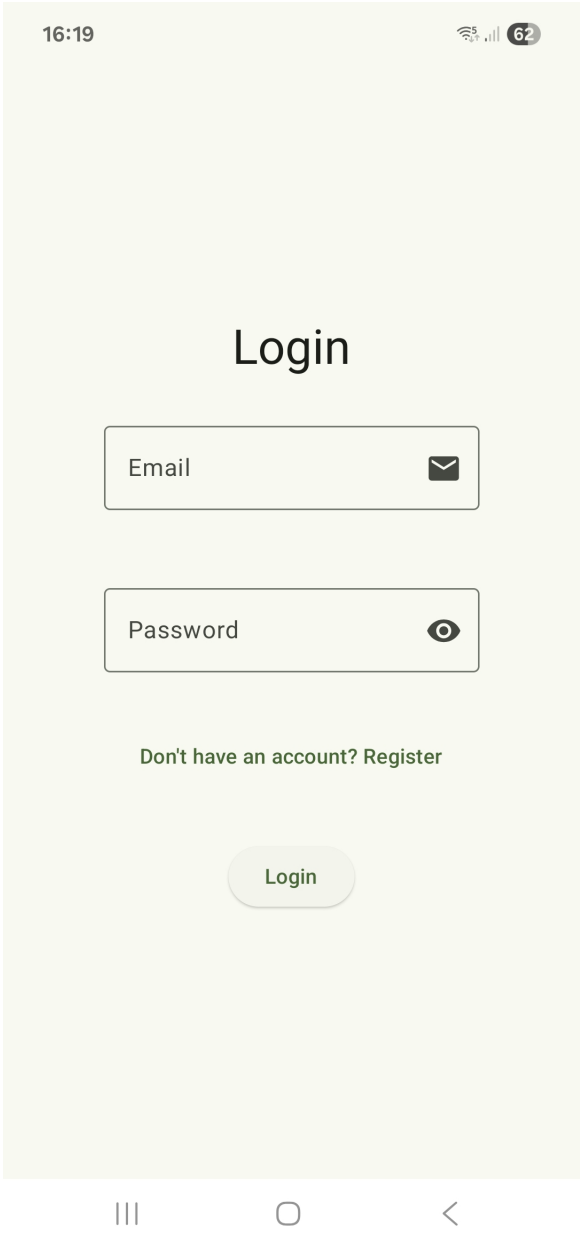
4 APRESENTAÇÃO DOS RESULTADOS

Este capítulo apresenta os resultados obtidos ao término do desenvolvimento da aplicação. As seções 4.1 a 4.6 exibem as diferentes telas do aplicativo, enquanto a seção 4.7 compara os resultados da pesquisa semântica do aplicativo com os do Google Fotos.

4.1 Tela de Login

A figura 22 apresenta a tela de login, que é similar à tela de cadastro, com a diferença que na de cadastro o título aparece como Register e há um link para ir à tela de login.

Figura 22 – Tela de Login

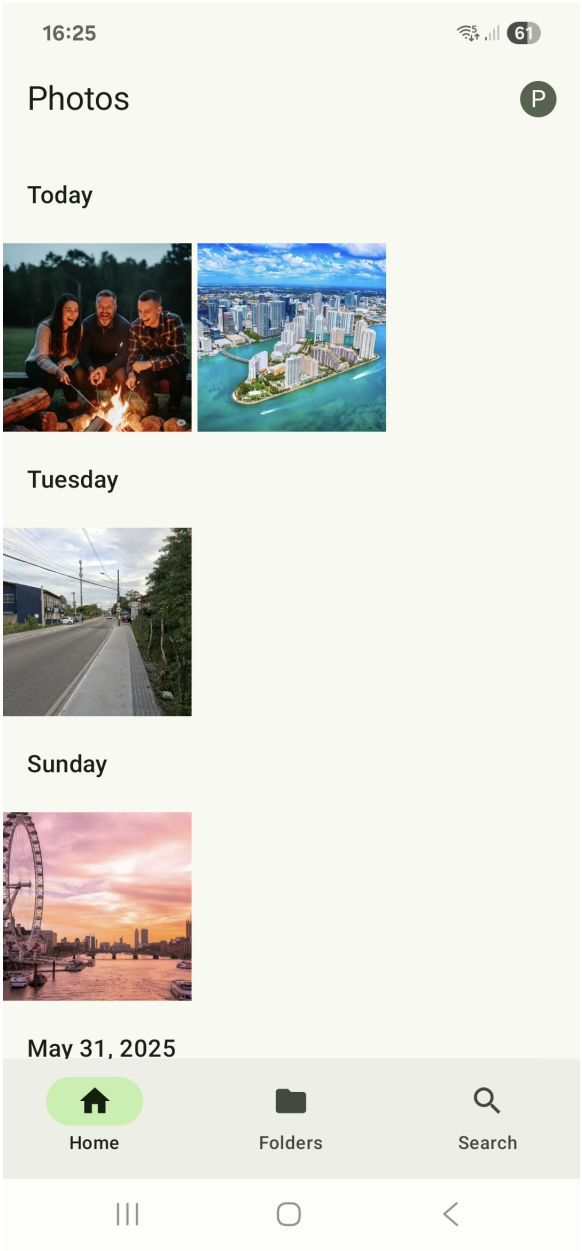


Fonte: Elaborado pelo autor.

4.2 Tela Inicial

A figura 23 apresenta a tela inicial do aplicativo, onde é possível ver o feed de imagens, separadas por data.

Figura 23 – Tela Inicial

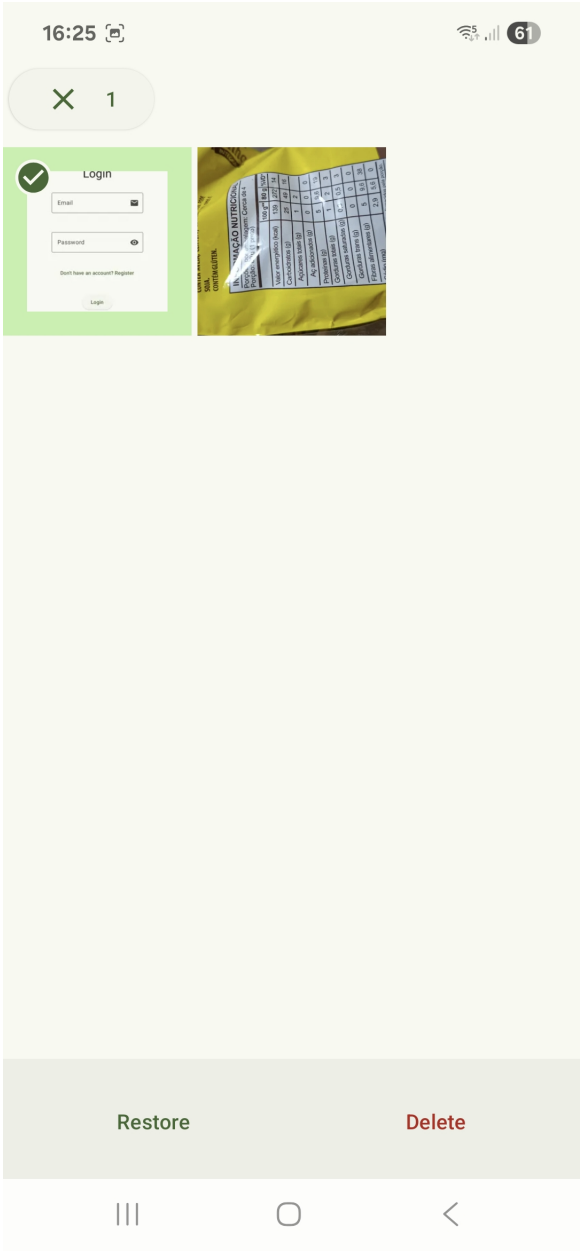


Fonte: Elaborado pelo autor.

4.3 Tela da Lixeira

A figura 24 apresenta a tela da lixeira do aplicativo. É possível observar que uma foto está selecionada e que os botões de deletar permanentemente (Delete) e restaurar (Restore) aparecem no canto inferior da tela.

Figura 24 – Lixeira

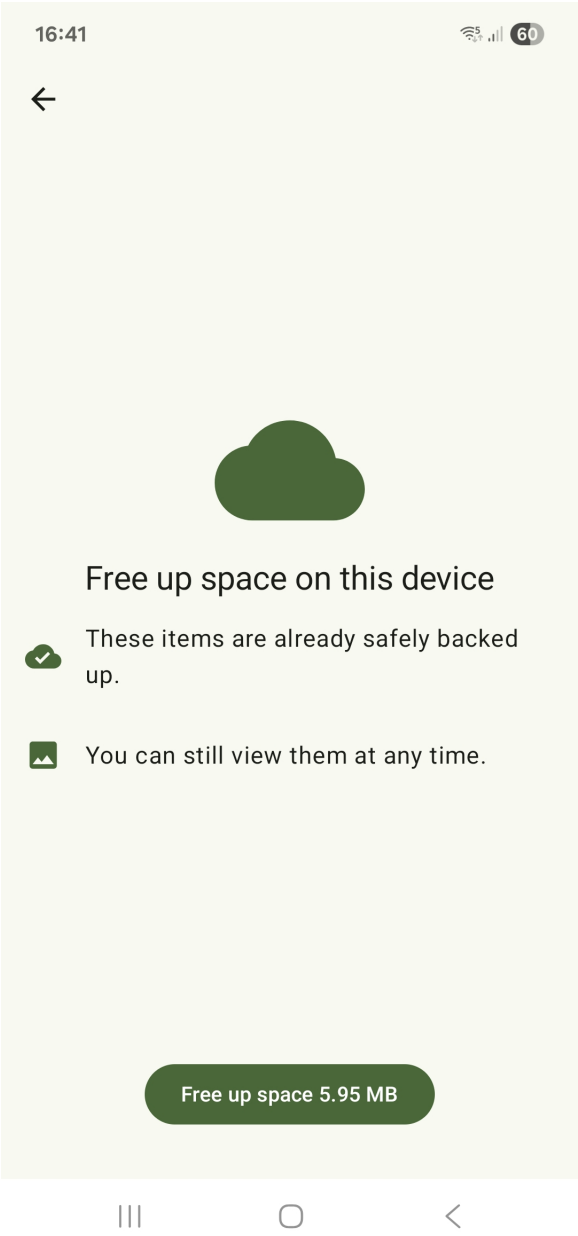


Fonte: Elaborado pelo autor.

4.4 Tela de Liberar Espaço

A figura 25 mostra a tela de liberar espaço. É possível observar a no botão quantidade de espaço a ser liberado.

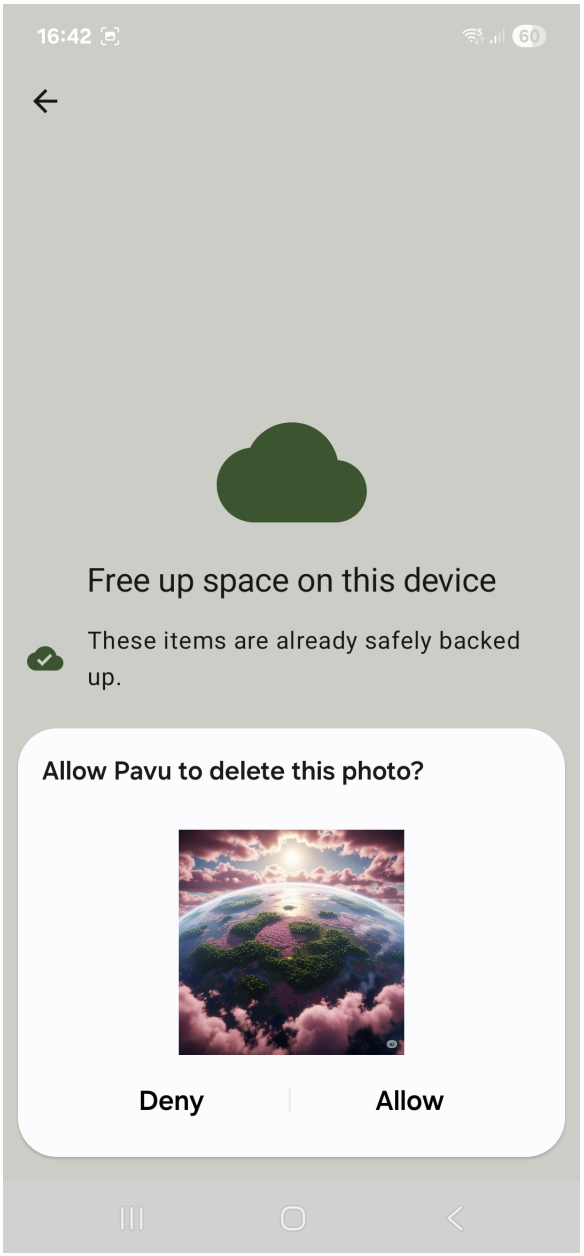
Figura 25 – Tela de Liberar Espaço



Fonte: Elaborado pelo autor.

A figura 26 mostra o aplicativo solicitando permissão para deletar um arquivo e liberar espaço.

Figura 26 – Caixa de Diálogo para Deletar Arquivo

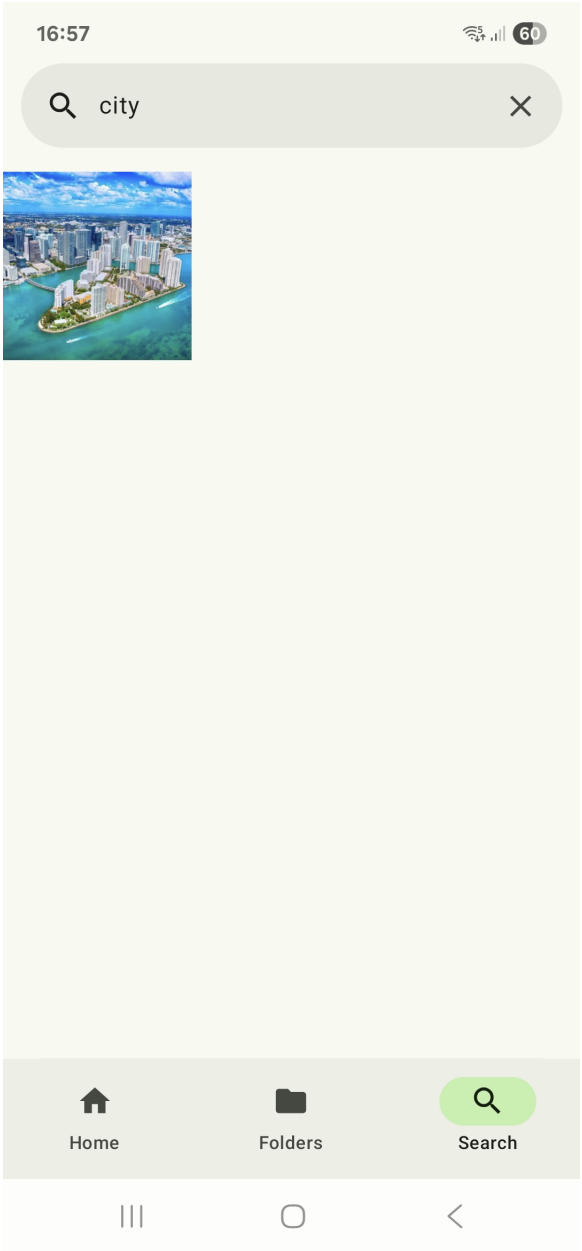


Fonte: Elaborado pelo autor.

4.5 Tela de Pesquisa

A figura 26 apresenta a tela de pesquisa do aplicativo.

Figura 27 – Tela de Pesquisa

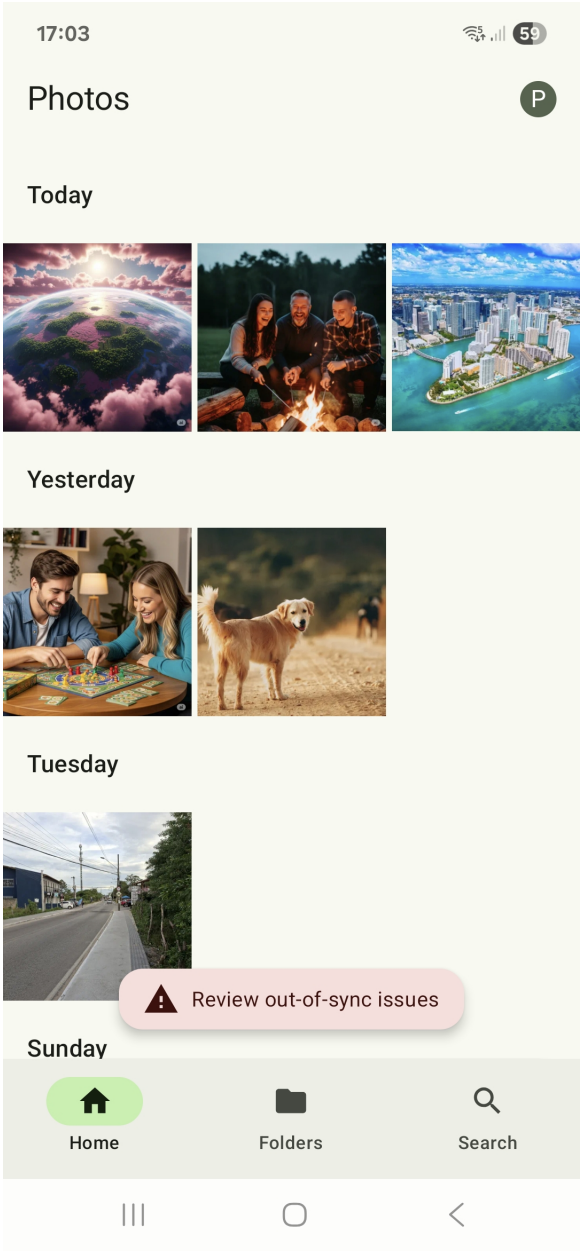


Fonte: Elaborado pelo autor.

4.6 Tela de Resolver Conflitos

A figura 28 apresenta o início do fluxo de resolver conflitos, na qual é possível ver o botão que redireciona à tela de conflitos.

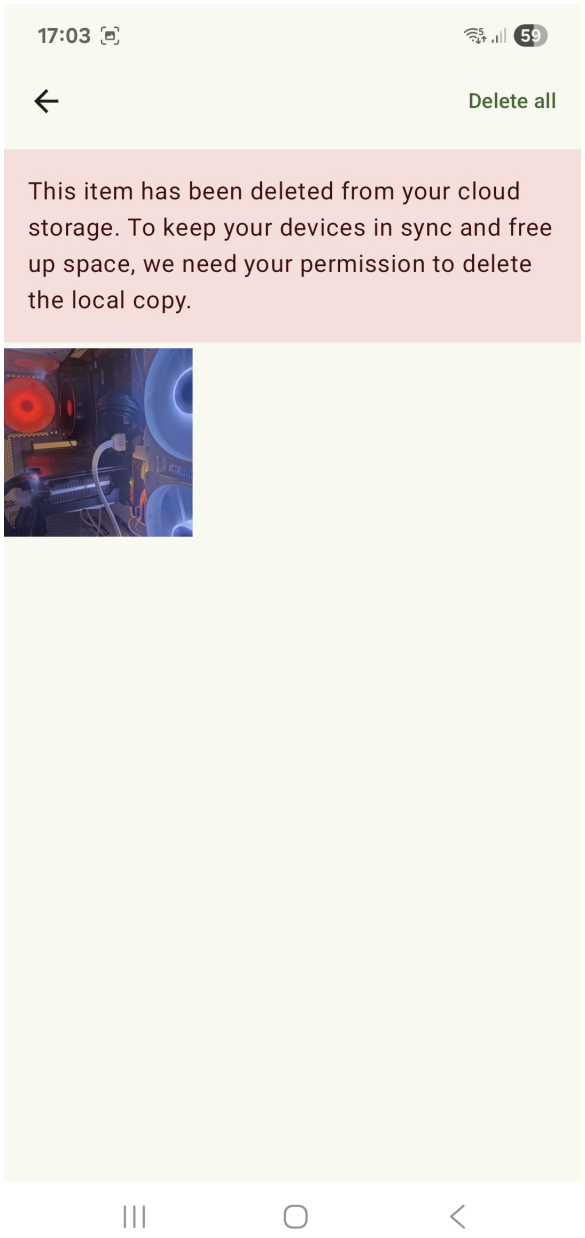
Figura 28 – Botão de Resolver Conflitos



Fonte: Elaborado pelo autor.

A figura 29 apresenta a tela de resolver conflitos, na qual aparece um aviso explicando o motivo do conflito e um botão para deletar arquivos locais.

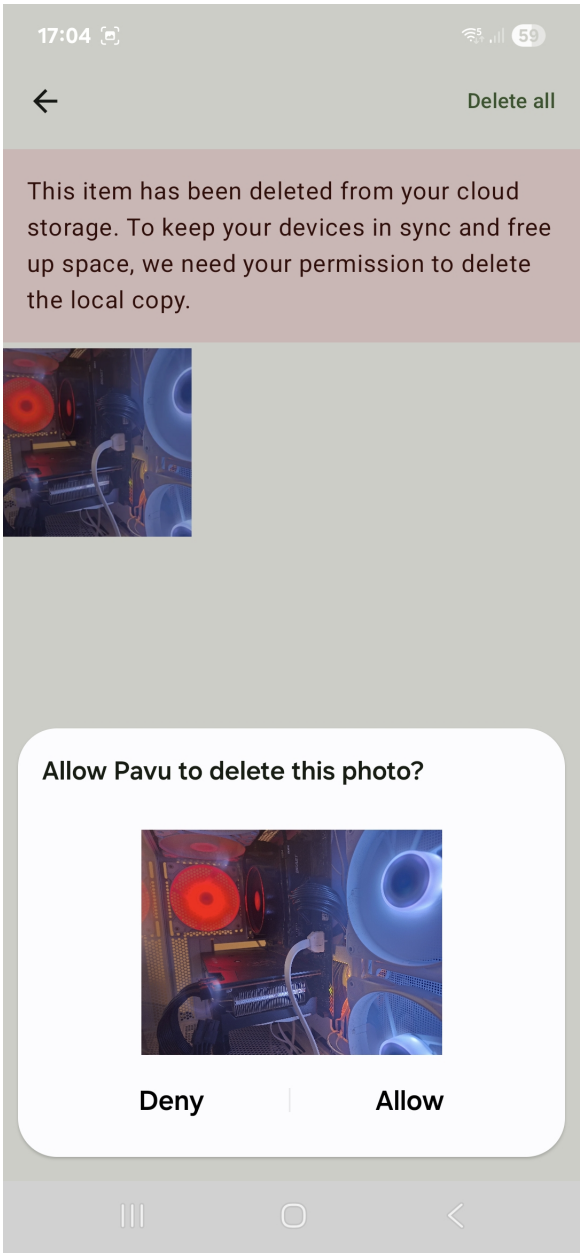
Figura 29 – Tela de Resolver Conflitos



Fonte: Elaborado pelo autor.

A figura 30 apresenta a caixa de diálogo para deletar arquivos locais e, assim, resolver os conflitos.

Figura 30 – Caixa de Diálogo para Deletar Arquivo e Resolver Conflitos

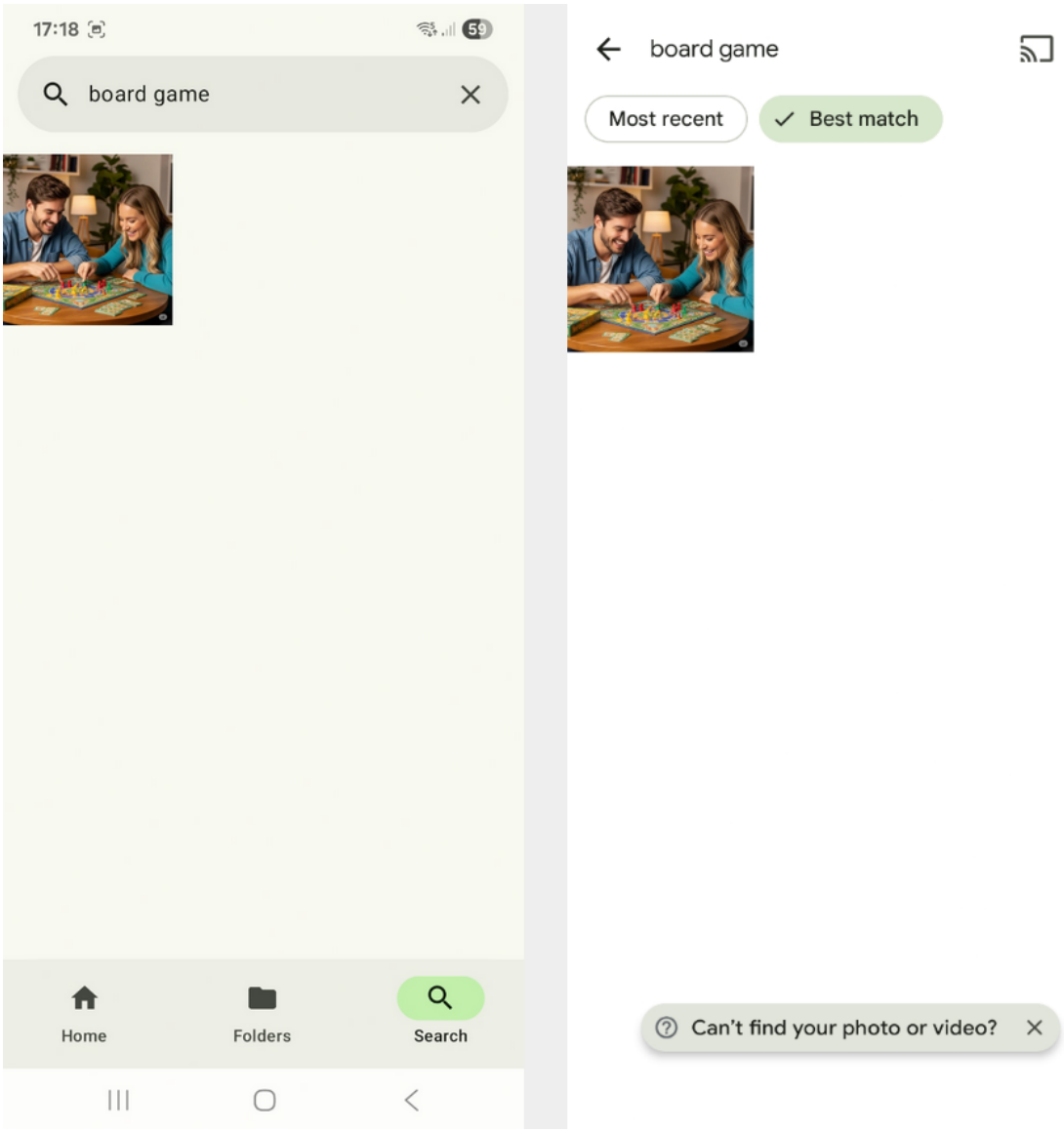


Fonte: Elaborado pelo autor.

4.7 Comparação dos Resultados de Pesquisa com o Google Fotos

A figura 31 mostra o resultado de pesquisa do termo "board game"para o aplicativo desenvolvido e o Google Fotos. É possível observar que ambos os apps retornam um único resultado, com excelente precisão.

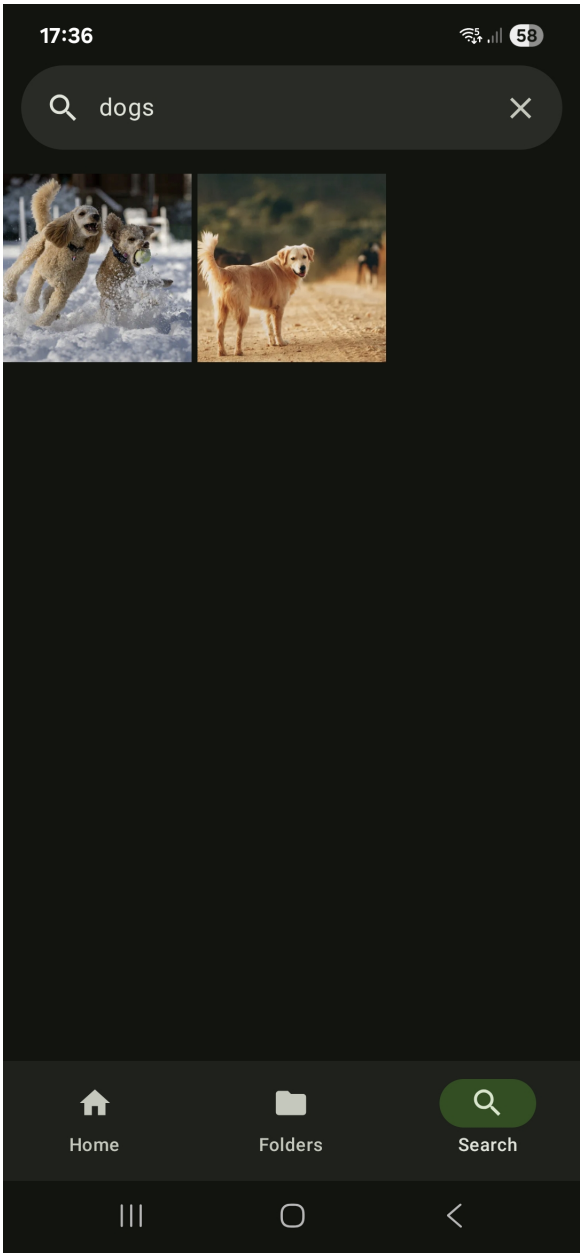
Figura 31 – Comparação dos Resultados de Busca para Jogo de Mesa



Fonte: Elaborado pelo autor.

Também é possível pesquisar por animais, como mostra a figura 32.

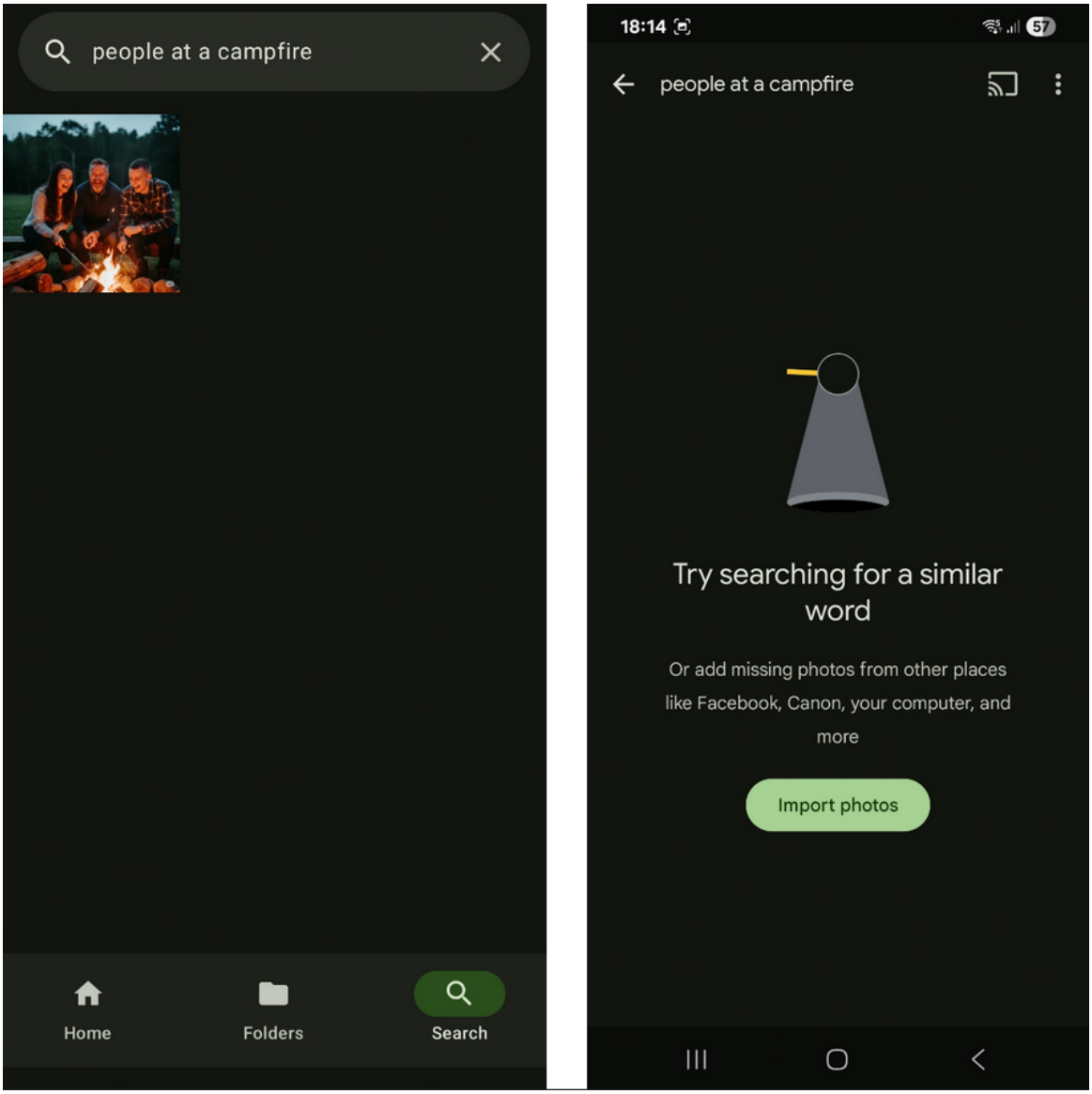
Figura 32 – Pesquisa por Animais



Fonte: Elaborado pelo autor.

Por vezes, o aplicativo desenvolvido consegue atingir resultados melhores que o Google Fotos, como mostra a figura 33.

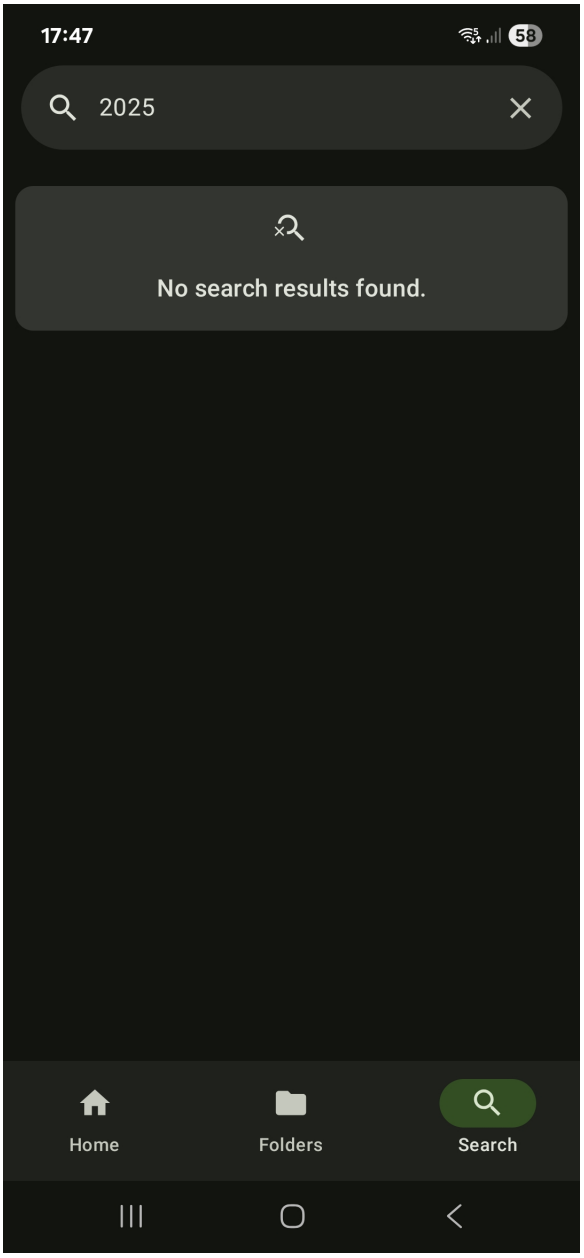
Figura 33 – Comparação dos Resultados de Busca para Fogueira



Fonte: Elaborado pelo autor.

Diferentemente do Google Fotos, entretanto, o aplicativo desenvolvido não permite buscas por data, pessoas ou localização. A figura 34 ilustra essa limitação, mostrando o resultado de uma pesquisa por fotos de 2025. Nota-se que o app não identifica imagens por data. Além disso, as pesquisas estão limitadas a termos em inglês, ao contrário do Google Fotos, que permite buscas em múltiplas linguagens.

Figura 34 – Pesquisa por Data



Fonte: Elaborado pelo autor.

5 Considerações Finais

Este trabalho teve como objetivo central propor e desenvolver um protótipo de sistema de gerenciamento de fotos com busca semântica baseada em inteligência artificial, buscando oferecer uma alternativa simplificada e focada no controle do usuário sobre seus dados pessoais, em contraponto a soluções amplamente utilizadas como o Google Fotos.

Para tanto, foi construído um servidor Spring Boot, responsável pela autenticação, armazenamento seguro de dados em banco SQL, criptografia e controle de acesso, em conjunto com um aplicativo para Android que permite o envio, visualização, pesquisa semântica e deleção de imagens.

Os resultados obtidos demonstram claramente a viabilidade de construir uma solução comparável ao Google Fotos, ao mesmo tempo em que se garante ao usuário o controle e domínio sobre seus dados pessoais, um diferencial fundamental neste cenário.

5.1 Limitações do Estudo

Apesar do sucesso na construção do protótipo e na validação dos princípios propostos, é importante reconhecer que este trabalho, por ser um TCC, possui algumas limitações inerentes a um projeto de escopo e tempo definidos. A dependência do Google Cloud Vision para a análise de IA, por exemplo, embora funcional, representa um ponto onde a privacidade poderia ser ainda mais aprimorada.

5.2 Sugestões para trabalhos futuros

Como todo projeto de pesquisa e desenvolvimento, este trabalho abre portas para futuras investigações e aprimoramentos. As sugestões apresentadas a seguir, como a implementação de modelos de IA locais, a indexação por localização e data, e o suporte a vídeos e outras plataformas, visam expandir as funcionalidades e aprimorar ainda mais o foco na privacidade e usabilidade da plataforma.

- Remover a dependência no GCV e utilizar um modelo de IA local, como o modelo DFN5B-CLIP-ViT-H-14-384 da Apple, de forma aumentar a privacidade da plataforma;
- Indexar a localização da imagem para tornar possível a pesquisa por localidade;
- Implementar a pesquisa por datas;
- Utilizar um modelo de IA para reconhecimento facial, possibilitando o cadastro e a pesquisa de pessoas específicas;

- Criar aplicativos para outras plataformas, como iOS e Windows;
- Melhorar a lógica de sincronização do aplicativo Android para permitir observar a fila de fotos a ser sincronizada, junto com o progresso individual de cada imagem;
- Adicionar suporte a vídeos;
- Adicionar suporte à pesquisa em outras línguas além do inglês;
- Implementar edições básicas, como aplicar filtros, cortar ou rotacionar;
- Implementar outras funcionalidades do Google Fotos, como a lista de memórias.

Em suma, o protótipo desenvolvido neste TCC não é apenas uma prova de conceito funcional, mas um indicativo de que a inovação tecnológica pode e deve caminhar de mãos dadas com a proteção da privacidade e o empoderamento do usuário no controle de suas informações.

REFERÊNCIAS

ANDROID. *Build better apps faster with Jetpack Compose*. 2025. Acesso em: 17 de mai. de 2025. Disponível em: https://developer.android.com/compose. 13

CLOUD, G. *What is semantic search, and how does it work? | Google Cloud*. 2025. Acesso em: 16 de mai. de 2025. Disponível em: https://cloud.google.com/discover/what-is-semantic-search. 11, 12

DOCKER. *Docker: Accelerated Container Application Development*. 2025. Acesso em: 4 de jun. de 2025. Disponível em: https://www.google.com/photos/about/. 12

DUFFY, C.; EADICICCO, L. *UK users are losing a key Apple security feature, raising questions about the future of privacy*. 2024. Acesso em: 17 de mai. de 2025. Disponível em: https://edition.cnn.com/2025/02/25/tech/apple-advanced-data-protection-uk-encryption. 14

GOOGLE. *Cloud Storage pricing*. 2025. Acesso em: 4 de jun. de 2025. Disponível em: https://cloud.google.com/storage/pricing#multi-regions. 32

GOOGLE. *Google Photos: Edit, Organize, Search, and Backup Your Photos*. 2025. Acesso em: 7 de jun. de 2025. Disponível em: https://www.google.com/photos/about/. 11

GOOGLE. *Object Lifecycle Management*. 2025. Acesso em: 4 de jun. de 2025. Disponível em: https://cloud.google.com/storage/docs/lifecycle. 32

GOOGLE. *What is Tink?* 2025. Acesso em: 17 de mai. de 2025. Disponível em: https://developers.google.com/tink/what-is. 14

KASPERSKY. *Criptografia de dados*. 2025. Acesso em: 16 de mai. de 2025. Disponível em: https://www.kaspersky.com.br/resource-center/definitions/encryption. 13

POSTGRESQL. *JSON Types*. 2025. Acesso em: 2 de jun. de 2025. Disponível em: https://www.postgresql.org/docs/current/datatype-json.html. 30

RAZA, M. *Encryption Explained: At Rest, In Transit End-To-End Encryption*. 2025. Acesso em: 17 de mai. de 2025. Disponível em: https://www.splunk.com/en_us/blog/learn/end-to-end-encryption.html. 14

SCHWENKE, E. *Is Google Drive Secure? Everything You Need to Know about Google Drive Security*. 2024. Acesso em: 17 de mai. de 2025. Disponível em: https://www.mimecast.com/blog/google-drive-security-guide/#:~:text=End%2Dto%2Dend%20encryption%20is,from%20the%20file%20storage%20location. 14

SPRING. *Spring Framework*. 2025. Acesso em: 16 de mai. de 2025. Disponível em: https://typesense.org/docs/27.1/api/vector-search.html#what-is-an-embedding. 12

TOTVS, E. *Criptografia simétrica e assimétrica: confira a diferença*. 2025. Acesso em: 16 de mai. de 2025. Disponível em: https://www.totvs.com/blog/gestao-para-assinatura-de-documentos/criptografia-simetrica-e-assimetrica/. 13

TYPESENSE. *Image Search | Typesense*. 2025. Acesso em: 4 de jun. de 2025. Disponível em: https://typesense.org/docs/27.1/api/image-search.html#create-a-collection. 32

TYPESENSE. *Vector Search | Typesense*. 2025. Acesso em: 16 de maio de 2025. Disponível em: https://typesense.org/docs/27.1/api/vector-search.html#what-is-an-embedding. 12