Mara Filipa Alexandre Felismino

Licenciada em Engenharia Informática

Conceção e Desenvolvimento de uma Aplicação Android para Eliminação Assistida de Fotografias Repetidas

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: João M. Lourenço, Prof. Auxiliar,
Faculdade de Ciências e Tecnologias
Universidade Nova de Lisboa

Co-orientador: Fernando P. Birra, Prof. Auxiliar,
Faculdade de Ciências e Tecnologias
Universidade Nova de Lisboa

Júri

Presidente: Prof. Ludwig Krippahl, FCT-NOVA
Arguente: Prof. Rui P. S. Nóbrega, FEUP
Vogal: Prof. João M. Lourenço, FCT-NOVA

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2019

Conceção e Desenvolvimento de uma Aplicação Android para Eliminação Assistida de Fotografias Repetidas

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Aos meus pais, irmã e namorado.

Agradecimentos

Esta dissertação é o resultado da cooperação de um grande grupo de pessoas. Estou profundamente grata a todas as pessoas que me apoiaram em todos os sentidos para tornar esta realização possível e, nas linhas seguintes, enumero algumas delas.

Em primeiro lugar, gostaria de expressar a minha gratidão aos meus conselheiros e orientadores, Professor João Lourenço e Professor Fernando Birra, pela oportunidade de integrar num projeto como este e por toda a orientação durante as fases de pesquisa e desenvolvimento. Foi uma ótima oportunidade de aprendizagem e profundamente importante para os meus desafios futuros. Desejo-lhes sucesso profissional e pessoal.

De seguida, gostaria de agradecer à WorldIT pela oportunidade, por todo o apoio e latitude dada ao longo de toda a dissertação, mas também de todos estes anos enquanto trabalhadora-estudante. Da grande equipa que encontrei na WorldIT, quero expressar a minha mais profunda gratidão ao Eng. Luís Saraiva e o Eng. Tiago Pereira, pelo apoio nos momentos de maior necessidade e por toda a gentileza trazida ao escritório.

Aos meus pais e irmã gostaria de agradecer por todos os esforços e apoio ao longo do tempo deste meu percurso académico. A distância e as saudades apertaram, mas sempre se mostraram disponíveis para me dar força. Foi uma longa jornada com muitos obstáculos e que nunca conseguiria superar sem vocês.

Finalmente, gostaria de agradecer ao meu namorado e aos meus amigos, por toda a ajuda e apoio nos bons e maus momentos. Obrigado por acreditarem em mim. Obrigado por tudo.

Obrigado a todos!

Resumo

A generalização da fotografia digital reduziu de forma significativa os custos da produção de fotos. Atualmente o único custo associado à produção de mais uma fotografia digital é o custo indireto do espaço que esta ocupa no suporte de armazenamento. Como a generalidade dos dispositivos móveis (e.g., telemóveis, *tablets*) da atualidade contam com câmaras fotográficas digitais de elevada qualidade, as pessoas tiram, com frequência, várias fotos repetidas do mesmo motivo, na expectativa de que pelo menos uma fique "boa" e com o plano de eliminar as restantes. Como o processo de escolha de qual das fotografias repetidas é realmente a melhor raramente é fácil, constata-se que a escolha da melhor fotografia e eliminação das restantes acaba por não ser realizada. O resultado são telemóveis e tablets com as suas galerias de fotos com imensas fotografias repetidas e com o espaço de armazenamento cheio.

Nesta dissertação propõe-se uma arquitetura de software de suporte a um *workflow* que visa a eliminação assistida de fotografias repetidas. Este *workflow* facilita o processo de comparação das características técnicas das fotografias repetidas, facilitando a identificação da melhor e, consequentemente, a eliminação das restantes. Nesta dissertação apresenta-se também o processo de conceção e implementação de uma aplicação móvel para dispositivos Android, que realiza parte da arquitetura proposta. Esta aplicação assenta em técnicas de análise e processamento de imagem para sugerir ao utilizador uma ordenação das fotografias repetidas de acordo com as suas qualidades técnicas, permitindo depois ao utilizador analisar e rever a ordenação proposta. Uma vez concluído o processo e a melhor fotografia escolhida e confirmada pelo utilizador, as restantes fotografias repetidas poderão ser eliminadas. Para ajudar o utilizador a decidir-se pela melhor fotografia, deu-se especial destaque à comparação direta de duas fotos, oferecendo na aplicação mecanismos de interação que agilizam essa comparação.

Palavras-chave: Android, Aplicações Móveis, Fotografia Digital, Gestão de Fotografias

Abstract

The generalization of digital photography has reduced significantly the costs of making photos. Currently, the only cost associated with the production of yet another digital photograph is the indirect cost of the space it occupies in the storage medium. Since nowadays most of the mobile devices (e.g., mobile phones, tablets) have high-quality digital cameras, people often take multiple repeated pictures of the same subject, expecting at least one to be "good enough" and with the plan to eliminate the remaining ones. As the process of choosing which of the repeated photographs is really the best is not easy, it turns out that this last step of choosing the best photograph and elimination of the remainder stays undone. The result is mobile phones and tablets with their photo galleries with a lot of repeated pictures and a full storage space.

This dissertation proposes a software architecture to support a workflow that aims at the assisted elimination of repeated photographs. This workflow facilitates the process of comparing the technical characteristics of the repeated photographs, facilitating the identification of the best one and, consequently, the elimination of the remaining ones. This dissertation also presents the process of designing and implementing a mobile application for Android devices, which is part of the proposed architecture. This application is based on image processing and analysis techniques to suggest an ordering of the repeated photographs according to their technical qualities, then allowing the user to analyze and review the proposed ordering. Once this process is completed and the best photograph is chosen and confirmed by the user, the remaining repeated pictures may be deleted. In order to help the user to decide on the best photograph, the application emphasizes the direct comparison of two photos, offering the interaction mechanisms that speed up this comparison.

Keywords: Android, Mobile Applications, Digital Photography, Photography Management

Índice

Lista de Figuras

Lista de Tabelas

Glossário

cloud	é o fornecimento de serviços informáticos (servidores, armazenamento, bases de dados, rede, *software*, análises, entre outros) através da Internet.
continuous shooting mode	é um modo de fotografar utilizado principalmente para captar o movimento sucessivo, como por exemplo em fotografias de desporto.
cross platform software	*software* implementado em múltiplas plataformas.
lambda	é uma função anónima que pode receber qualquer número de argumentos, mas com uma única expressão.
metadados	informações recolhidas das fotografias tais como a data de criação, a localização, ainda com a adição de outros dados mais internos da fotografias, como características (exposição, modelo de cores), palavras-chave, entre outros.
sdk	pacote de ferramentas de programação que permite que um programador desenvolva aplicações para uma plataforma específica.
sessão	é o processo de organização, análise e seleção de fotografias. O início e término da sessão será exatamente ao mesmo tempo que o do processo.
sub-exposição	característica de uma fotografia quando capturada com pouca luz.
timestamp	marca temporal que localiza um objeto ou evento no tempo.

Siglas

ANR	*Application Not Responding*.
API	*Application Programming Interface*.
APK	*Android Application Pack*.
HCI	*Human-Computer Interaction*.
IDE	*Ambiente Integrado de Desenvolvimento*.
OOM	*Out Of Memory*.
OpenCV	*Open Source Computer Vision Library*.
REST	*Representational State Transfer*.
UI Design	*User Interface Design*.

Capítulo 1

Introdução

1.1 Contexto

A evolução tecnológica permitiu a generalização do uso de telemóveis com câmara fotográfica, o que significa que uma grande parte da população tem sempre consigo um dispositivo capaz de fazer fotografias digitais com relativa boa qualidade e por um custo adicional nulo em relação a ter o telemóvel. O crescimento das redes sociais incentiva a partilha de momentos relevantes da nossa vida e as fotografias são um excelente testemunho desses momentos. Estas plataformas incentivam o utilizador a partilhar os acontecimentos relevantes do seu dia a dia, sendo para muitos importante a qualidade da fotografia capturada. Outro exemplo de utilização massiva das câmaras fotográficas dos telemóveis, é o caso em que um utilizador vai de férias e tira fotografias para mais tarde poder partilhar com os seus amigos ou familiares. Neste caso tira com frequência várias fotografias do mesmo tema/motivo com o intuito de, mais tarde, as rever e deixar apenas as melhores, apagando as que estão menos boas ou repetidas.

1.2 Descrição do Problema

Existem inúmeras razões que levam um utilizador a tirar fotografias, sejam estas profissionais, um simples passeio ao ar livre, *selfies* com amigos, ou até mesmo *real-time events*. As estatísticas do *Flickr* referentes a 2017 dizem que metade das fotografias expostas no *site* tem como origem *smartphones* [1].

Aproveitando um dos cenários descritos anteriormente, por exemplo, o caso em que um utilizador vai de férias e tira imensas fotografias, quando chega o momento da sua visualização o utilizador depara-se com um problema: demasiadas fotografias repetidas para cada um dos temas que capturou (ver Figura 1.1). Uma das possíveis causas para

esta repetição pode ser o próprio modo de fotografar continuamente (*continuous shooting mode*), mas também pode ser apenas o simples facto do utilizador querer tirar bastantes fotografias a cada um dos temas, na esperança de que pelo menos uma das fotografias tenha capturado o tema de forma que o satisfaça.

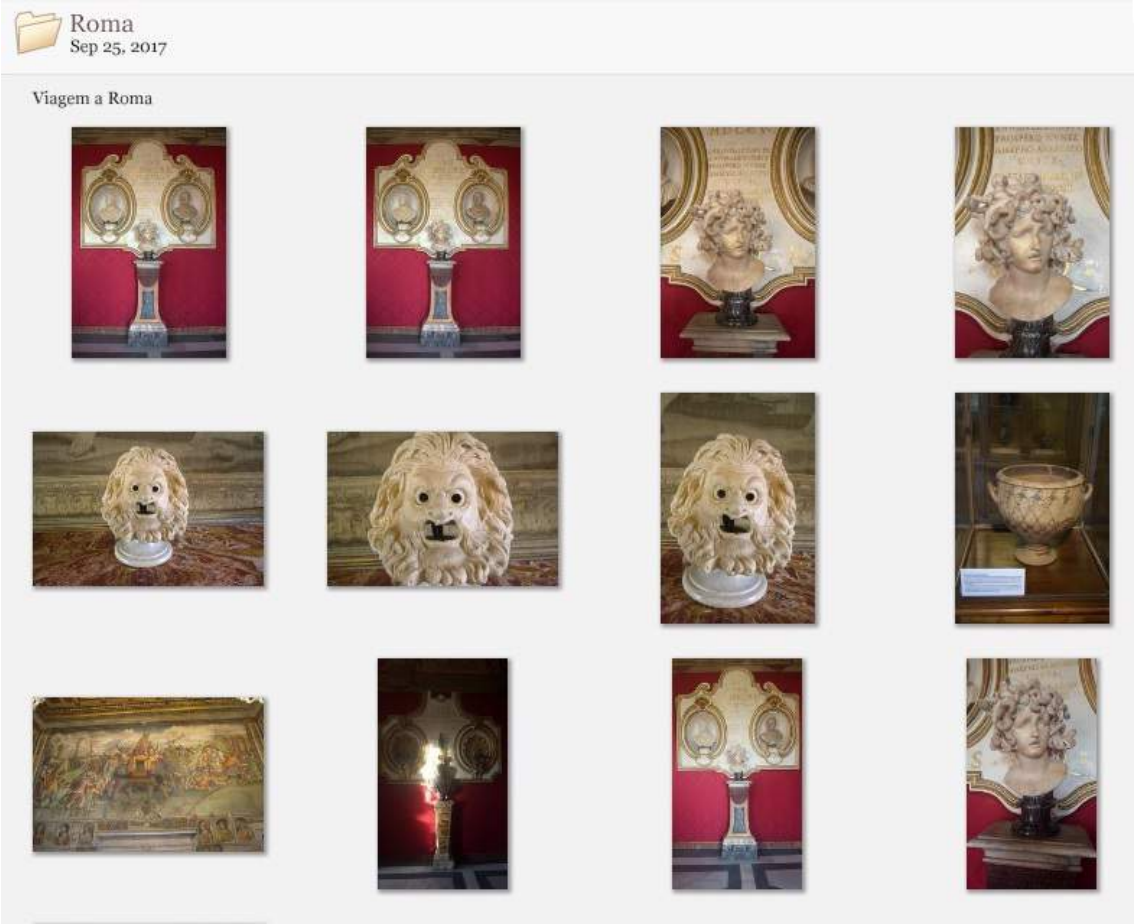


Figura 1.1: Conjunto de fotografias com duplicados para determinados temas.

Voltando à Figura 1.1, o utilizador, ao analisar mais cuidadosamente o conjunto de fotografias que tirou durante a sua visita ao Museu Capitolini [2], identifica seis temas diferentes com uma ou mais fotografias semelhantes (repetidas): a estátua completa da medusa, o busto da medusa, a estátua com a boca aberta, a cratera de *Aristonothos*, o quadro dos nossos antepassados e outra estátua (com sub-exposição), que embora esteja com o mesmo enquadramento, não é da medusa. O utilizador vê-se assim na necessidade de comparar as fotografias semelhantes, para cada um dos temas, na procura de qual a que melhor corresponde às suas expectativas. Quando realizado sem ajuda de software especializado, este processo consiste em abrir duas fotografias lado a lado ou em sequência, primeiro uma, depois a outra, depois novamente a primeira, depois novamente a segunda, sem o auxílio de qualquer ferramenta para além do visualizador, de forma a tentar validar e valorizar os pontos relevantes de cada fotografia em relação à sua outra semelhante (ver Figura 1.2). Porém, nem todas as características são passíveis de ser analisadas na versão

da imagem que foi reduzida para caber no ecrã do dispositivo. Este é o caso do foco, pois o processo de redução para caber no ecrã aumenta o *sharpening* das imagens. Neste caso, será necessário fazer zoom sobre os pontos de interesse, avaliar a nitidez, selecionar a fotografia seguinte, fazer zoom, avaliar a nitidez, e depois voltar à primeira e à segunda quantas vezes forem necessárias até se tomar uma decisão.

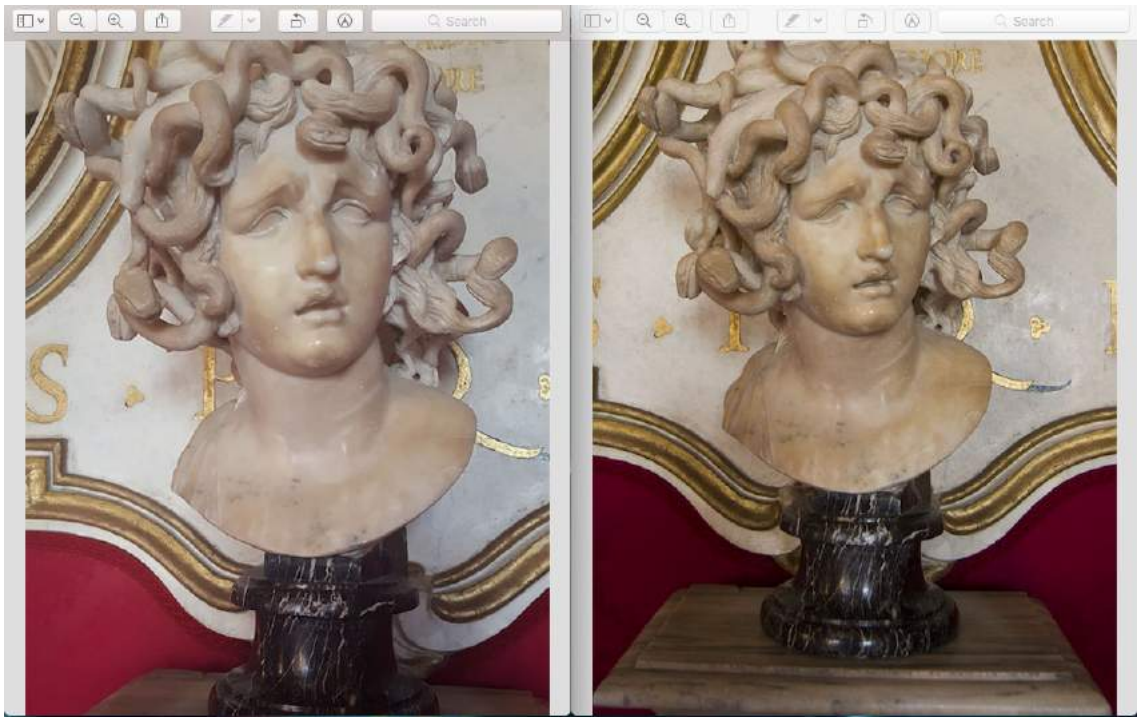


Figura 1.2: Comparação entre fotografias, na procura de qual a que melhores caraterísticas apresenta.

Terminado o processo de comparação para um dos subconjuntos, segue-se a repetição do processo para os restantes conjuntos de fotografias semelhantes, para que no final se fique apenas com um exemplar que será a melhor fotografia de cada conjunto[1]. Este procedimento de avaliação e seleção de fotografias, aplicado a cada um dos temas fotografados, pode gerar ansiedade no utilizador pois é moroso e baseado numa avaliação visual pouco informada.

Assim, o objetivo deste trabalho é a conceção e implementação de uma aplicação de suporte ao processo de organização, comparação e análise de fotografias semelhantes, com o objetivo de ajudar o utilizador a selecionar as melhores e eliminar as demais. Neste caso, a plataforma escolhida para o desenvolvimento da aplicação foi o Android.

[1] O termo "melhor fotografia" está a ser mencionado num contexto subjetivo, pois depende bastante da opinião do utilizador. No desenvolvimento deste trabalho, sempre que este termo seja referido será relativamente aos critérios do utilizador.

1.3 Solução Proposta

Tal como mencionado na Secção 1.2, após uma viagem de férias onde foram tiradas imensas fotografias, as pessoas com frequência deparam-se com o problema de terem demasiadas fotografias semelhantes para cada um dos temas capturados e da consequente necessidade da eliminação das repetidas.

A solução que propomos passa por desenhar e desenvolver uma aplicação de suporte ao utilizador aquando da organização, análise e seleção das melhores fotografias. Por exemplo, dado o conjunto de fotografias apresentado na Figura 1.1 (que inclui seis subconjuntos de fotografias semelhantes), a ferramenta irá propor a organização das fotografias por tema, ordenando-as da "melhor" para a "pior" fotografia (ver Figura 1.3).

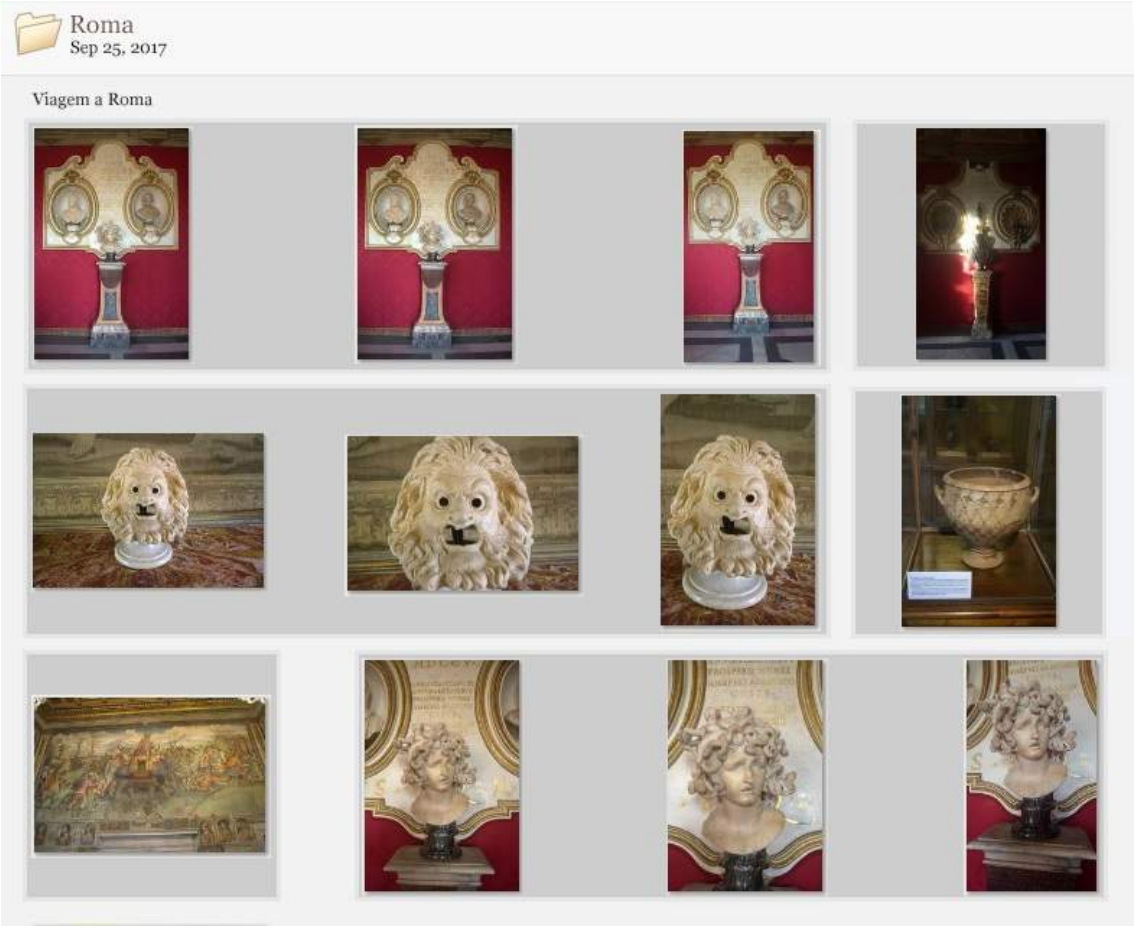


Figura 1.3: Organização de fotografias por tema, da melhor para a pior fotografia.

Uma vez organizadas as fotografias, o utilizador saberá qual a fotografia melhor qualificada em termos técnicos para cada um dos temas e poderá simplesmente aceitar a sugestão da aplicação, escolhendo imediatamente a fotografia melhor qualificada. Caso contrário, poderá fazer uma análise manual mais cuidada com o apoio da aplicação, confirmando a proposta desta ou selecionando outra fotografia como sendo a melhor. Se o utilizador optar por fazer a análise, poderá efetuar a comparação de duas fotografias

visualizando-as simultaneamente no ecrã, sincronizadas para um mesmo motivo. Além da comparação, o utilizador poderá visualizar os detalhes das fotografias através de gestos comuns de *zoom* e *panning*, de modo a decidir qual fotografia mais lhe agrada.

Ao terminar o processo de comparação e seleção de fotografias, o conjunto inicial ficará reduzido a apenas cinco fotografias: a melhor fotografia de cada um dos temas (ver Figura 1.4). Neste caso, o utilizador optou por eliminar completamente um dos temas, visto a sua única fotografia apresentar uma qualidade insatisfatória.

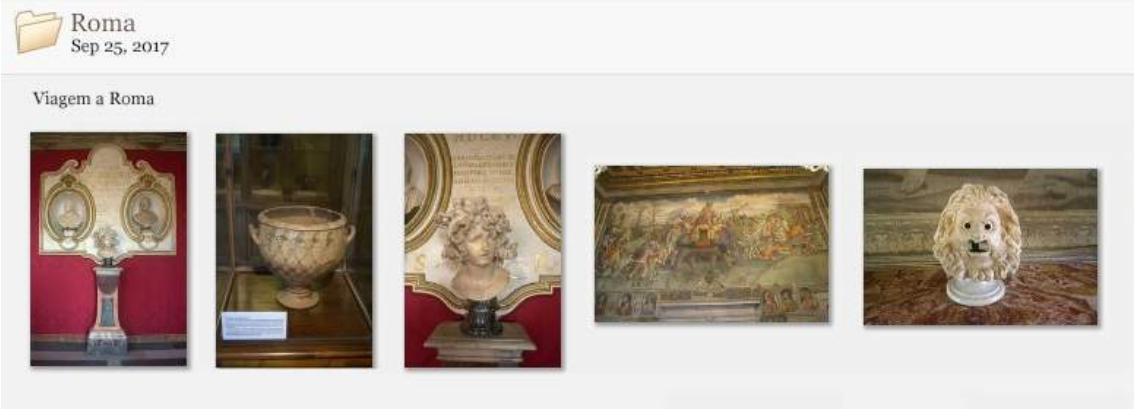


Figura 1.4: Resultado após a escolha do utilizador.

Ao fazer uso desta aplicação, o utilizador poderá não só poupar tempo e libertar muito do espaço ocupado em disco pelas fotografias repetidas, como ainda o conseguirá fazer mantendo baixo o nível de ansiedade causado pela incerteza de se a sua fotografia preferida é ou não tecnicamente aceitável. Outra vantagem desta aplicação será o facto de poder ser implementada em múltiplas plataformas, chegando a mais utilizadores e promovendo uma solução de continuidade[2] através de sessões.

Tal como mencionado anteriormente, a plataforma escolhida para implementar a solução foi o Android. Para tornar a aplicação independente da plataforma — *cross platform software* — foi necessário conceber uma arquitetura que possibilitasse a utilização da aplicação não apenas em *smartphones* e *tablets*, como também em computadores (na forma de aplicações nativas e/ou web). Outra vantagem do desenvolvimento *cross platform* em junção com sessões, é que passa a ser possível que o utilizador inicie o processo de eliminação de fotografias semelhantes num dispositivo e termine noutro. Para implementar o conceito de continuidade, decidimos incluir na arquitetura um servidor na *cloud*, para armazenar as sessões existentes para cada utilizador e realizar também alguns dos processamentos que se verificam computacionalmente mais pesados e, portanto, inadequados para serem realizados em equipamentos móveis com capacidade de processamento e recursos energéticos limitados.

A arquitetura proposta é apresentada na Figura 1.5, onde se propõe a sua divisão em três grandes módulos: o servidor; as plataformas onde a aplicação poderá correr; e os

[2] O termo continuidade é utilizado para descrever o início de um trabalho/ação num dispositivo e término noutro.

gestores de fotografias já previamente utilizados e configurados com as fotografias do utilizador. Na Figura existem três ligações entre os grupos, uma entre o servidor e as plataformas disponíveis, outra entre as plataformas e os gestores de fotografias e, por fim, uma entre o servidor e os gestores.

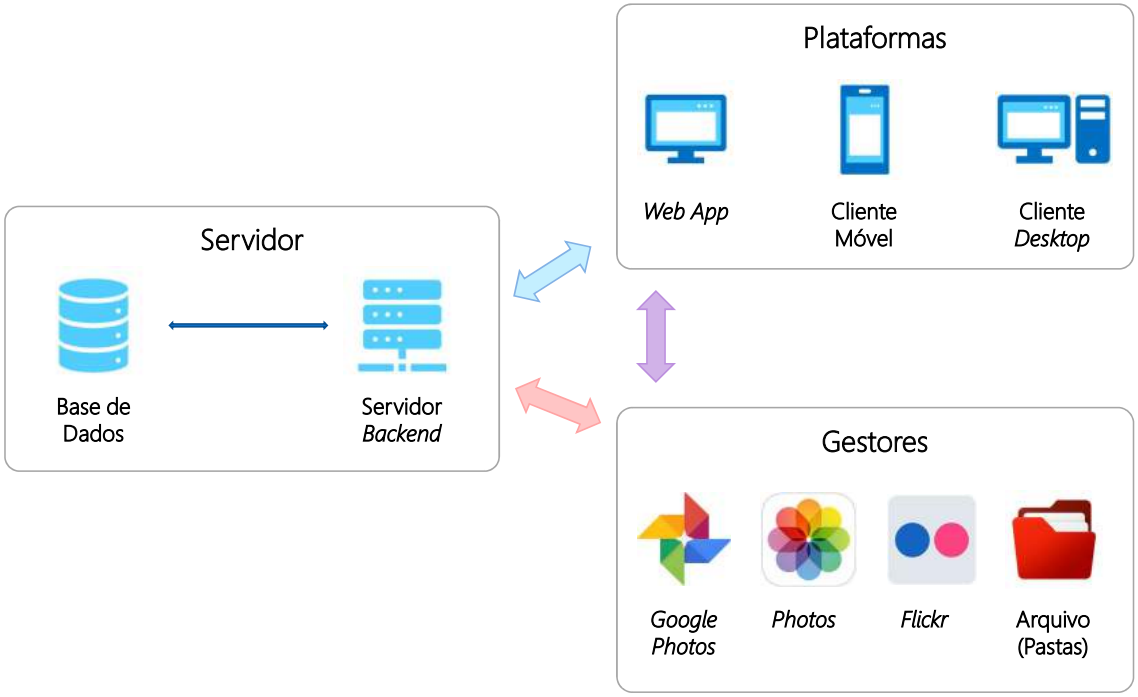


Figura 1.5: Arquitetura proposta.

A primeira ligação, entre o servidor e as plataformas, visa permitir a transparência de tarefas que exigem processamentos pesados para ambiente móvel — de modo a poupar os seus (limitados) recursos — bem como permitir a gestão das sessões.

A segunda ligação, entre as plataformas e os gestores de fotografias, ilustra a conexão entre os serviços de fotografias já utilizados (gestores de fotografias) e a aplicação.

Por fim, a última ligação, entre o servidor e os gestores de fotografias, refere-se à importação das fotografias dos gestores de fotografia para o servidor. Esta importação permitirá evitar que sejam realizados *uploads* a partir do dispositivo móvel quando a fotografia em causa já foi enviada e está disponível num gestor de fotografias na *cloud*. Por outro lado, uma vez importadas as fotografias, é possível aplicar-lhes um conjunto de processamento necessários ao bom funcionamento da aplicação.

Os componentes apresentados na Figura 1.5 são:

- **Base de Dados** — Módulo responsável pelo armazenamento de todos os dados (sessões ativas, *caching* de processamentos efetuados, fotografias, entre outros);
- **Servidor *Backend*** — Módulo responsável pelo tratamento dos pedidos de processamento das fotografias, gestão da comunicação/sincronização entre as plataformas e de acesso às informações na base de dados;

- **Cliente Móvel** — Módulo referente às aplicações móveis (Android, iOS e Windows);
- **Cliente *Desktop*** — Módulo referente às aplicações de desktop (macOS e Windows);
- ***Web App*** — Módulo referente à aplicação na Web;
- ***Google Photos*, *Apple Photos* e *Flickr*** — Exemplos de gestores de fotografias (na *cloud*) utilizados pelos utilizadores;
- **Arquivo (Pastas existentes)** — Módulo referente às pastas de fotografias dos utilizadores.

Com a arquitetura proposta fornece-se não só um vasto leque de opções de utilização da aplicação como também uma maior flexibilidade/comodidade na sua utilização, uma vez que se retira quer a limitação dos dispositivos a usar — *Mobile Client*, *Desktop Client*, *Web App* — quer mesmo a obrigatoriedade da instalação de qualquer *software* extra —*Web App*. Outro fator importante é o facto de permitir que os utilizadores façam uso das suas fotografias já existentes noutros gestores — *Google Photos*, *Apple Photos e Flickr* — tirando proveito de processamentos e alterações já por eles efetuadas.

Embora a disponibilização duma receita automática para a identificação de grupos de semelhança e classificação das fotografias dentro desses grupos possa ser extremamente útil, especialmente detetando situações de onde estas são muito más em termos técnicos, haverá sempre a necessidade de assegurar ao utilizador o controlo da situação, pelo que o fluxo idealizado é apenas semi-automático.

A identificação de grupos de semelhança, por si só, é extremamente exigente do ponto de vista computacional, em especial tendo em atenção a utilização de dispositivos móveis com capacidades de processamento a recursos energéticos limitados. Para que uma fotografia seja considerada semelhante a outra é necessário correlacionar pontos de uma com pontos de outra. Este processo envolve deteção de pontos característicos em cada uma das fotos — que se destacam dos seus vizinhos — e um posterior processo de "acasalamento" entre alguns pontos de uma das imagens com alguns pontos da outra. Mesmo os algoritmos que permitem aferir a qualidade técnica duma imagem (foco, ruído, exposição deficiente, etc.), são demasiado exigentes para serem executados num dispositivo móvel.

Assim, havendo necessidade de transferir uma parte do processamento para a *cloud*, identificam-se desde logo um conjunto de funcionalidades relacionadas com a interação entre a aplicação móvel e os servidores da plataforma. Essas interações deverão permitir:

- Enviar para a plataforma um conjunto de fotografias para análise e processamento;
- Saber em que estado esse processamento se encontra;
- Obter o resultado do processamento automático.

Estas operações são essenciais para uma visão atualizada e consistente do processo em qualquer momento. Apesar dos processos que deverão ser executados na *cloud* serem

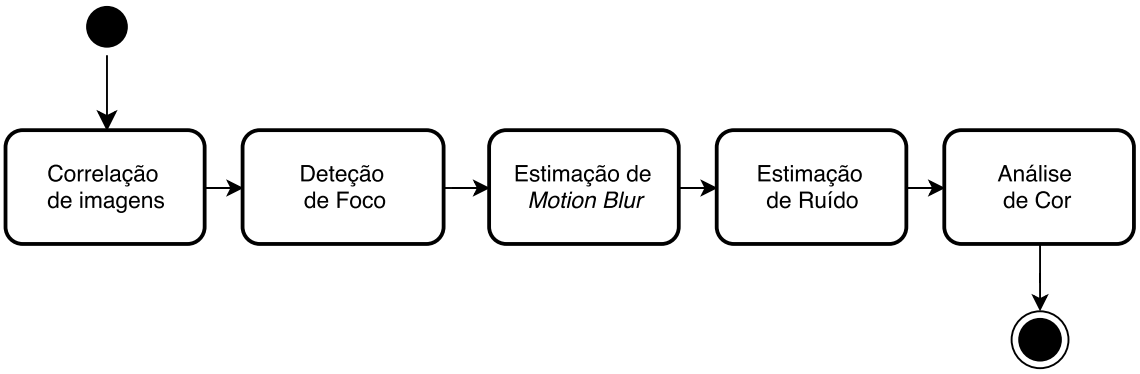


Figura 1.6: Proposta de *workflow* para processamento de coleções de fotografias.

automatizados, há grande vantagem em que sua configuração possa ser customizada e ajustada às necessidades do utilizador. A Figura 1.6 ilustra um *workflow* possível para a automatização do processo de classificação e ordenação dum conjunto de fotografias[3]. O estudo deste *workflow* e dos algoritmos a utilizar foram efetuados previamente por Alves [3] e Machado [4].

Após a identificação dos grupos e a sua classificação interna, o controlo fino do processo de identificação da melhor fotografia de um grupo, por parte do utilizador da aplicação, após o processamento automático ocorrido na *cloud*, deverá oferecer um conjunto de funcionalidades que passamos a identificar:

- Mover fotografias de um grupo de semelhança para outro;
- Subdividir um grupo de semelhança;
- Redefinir a ordenação dentro do grupo de semelhança;
- Eliminar fotografias dum grupo de semelhança.

É nossa convicção plena que, para que a análise da qualidade das fotografias dum mesmo grupo possa ser efetuada pelo utilizador, este terá que ser capaz de fazer comparações entre fotos, recorrendo a uma comparação entre pares de fotos. Qualquer análise de detalhe envolvendo mais do que duas fotografias em simultâneo não nos parece ser alcançável, nem desejável. Uma grande parte do trabalho desenvolvido nesta dissertação assenta precisamente neste ponto, onde pensamos ter uma proposta interessante.

1.4 Estrutura do Documento

Este documento está organizado em cinco capítulos. Este Capítulo 1, Introdução, teve como principal objetivo fazer a introdução do tema da dissertação, começando por apresentar o seu contexto, seguido de uma descrição mais detalhada do tema e do problema

[3] As funcionalidades de automatização do processo de classificação e ordenação dum conjunto de fotografias farão parte dos componentes a serem implementados pelo servidor

em si e terminando com uma descrição de todo o sistema por detrás da solução delineada e implementada.

No Capítulo 2, Tecnologias e Trabalho Relacionado, são descritos os conceitos principais de desenvolvimento móvel (tecnologias e conceitos a ter em conta), sendo que também são mencionados trabalhos e aplicações já existentes que apresentam funcionalidades dirigidas a alguns dos objetivos identificados, para além de serem também apresentados os conceitos básicos de comparação de imagens.

No Capítulo 3, Conceção da Interface do Utilizador e Desenvolvimento da Aplicação, é apresentado um protótipo da solução proposta, nomeadamente, todas as funcionalidades desenvolvidas, todas as maquetas pré-concebidas e todos os ecrãs efetivamente implementados.

No Capítulo 4, Desenvolvimento da Aplicação (*photo | uniq*), são introduzidos os fatores importantes para o desenvolvimento da aplicação, nomeadamente, a arquitetura, o modelo de dados, a interação aplicação/servidor, o sistema de autenticação, as transformações responsáveis pela comparação de fotografias lado a lado e, por fim, as bibliotecas externas utilizadas.

No Capítulo 5, Validação do Sistema, é apresentado um estudo do desempenho, da escalabilidade e do consumo de memória da aplicação em função de casos de uso relevantes.

No Capítulo 6, Conclusão e Trabalho Futuro, é feito um resumo do que foi implementado dos objetivos propostos, são apresentadas algumas lições aprendidas durante toda a conceção do sistema e desenvolvimento da aplicação e são ainda mencionadas possíveis melhorias e funcionalidades a serem acrescentadas no futuro.

Por fim, nos Appendices A e B são apresentados os dados recolhidos para a validação do sistema e a lista completa de *bugs* encontrados na versão atual da solução, respetivamente.

Capítulo 2

Tecnologias e Trabalho Relacionado

Neste Capítulo são descritos os conceitos principais de desenvolvimento para aplicações móveis (Secção 2.1), é feita uma análise das principais funcionalidades presentes nas aplicações que oferecem suporte na remoção de duplicados de fotografias (Secção 2.2), e é feito um resumo do sistema de suporte à comparação de imagens (Secção 2.3). Por fim, na Secção 2.4 é apresentado o Ambiente de Desenvolvimento, nomeadamente o sistema operativo Android, algumas das linguagens populares e *Ambiente Integrado de Desenvolvimento* (IDE) oficial para programar em Android.

2.1 Conceitos Fundamentais do Desenvolvimento de Aplicações Móveis

Com o surgimento do *smartphone*, sendo este um dispositivo portátil, sempre presente com o utilizador no dia a dia, surgiram também as aplicações móveis em resposta às necessidades dos utilizadores como, por exemplo, para a leitura de e-mails ou marcação de eventos na agenda.

Ao migrar do desenvolvimento de aplicações *desktop* para aplicações móveis, foi necessário ter em conta muitas outras áreas, nomeadamente, a capacidade de processamento e armazenamento, os tamanhos dos *displays*, o consumo energético, as necessidades de comunicação, entre outras. Estas áreas acabaram por fazer parte do processo de desenvolvimento de uma aplicação móvel, uma vez que para aplicações de *desktop*, não havia a preocupação do consumo de energia, dos tamanhos de ecrã ou, até mesmo, da disponibilidade de rede móvel. Para além destes novos encargos a ter em conta, este avanço tecnológico fez com que o número de aplicações móveis disparasse, tornando a componente móvel importante nos mercados pessoal, profissional e de entretenimento [5]. Assim, para que existisse alguma homogeneidade entre as interfaces disponibilizadas pelas aplicações

móveis, acabaram por surgir algumas regras e diretrizes a cumprir, por parte de cada uma das plataformas existentes — Android [6], iOS [7] e Windows Phone [8].

A presente Secção pretende introduzir os conhecimentos/conceitos básicos necessários para a criação de uma aplicação móvel, bem como algumas das regras e diretrizes existentes.

Quando surge a ideia de desenvolver um novo produto, projetado para dispositivos móveis, existem sete grandes fases [9, 10] no todo que é o processo de desenvolvimento de uma aplicação móvel: Ideia, Prototipagem, *Design* da Interface do Utilizador, Desenvolvimento, Validação, Publicação e Atualizações (ver Figura 2.1).

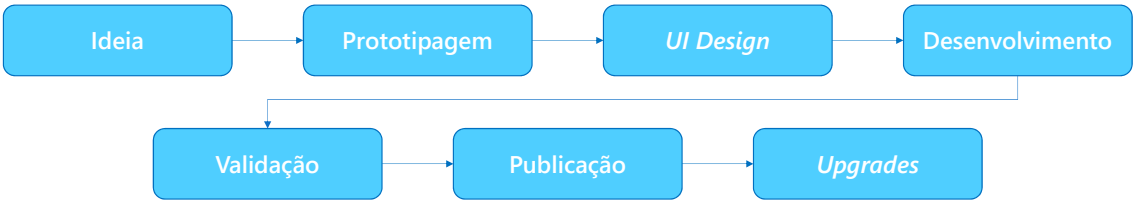


Figura 2.1: Processo de Desenvolvimento de Aplicações Móveis.

2.1.1 Ideia

Ideia, o objetivo/propósito para o qual se pretende desenvolver uma aplicação. Pode-se dizer que este é o passo mais importante de todos, uma vez que sem propósito não existe aplicação. Para além do objetivo é também necessário fazer alguma pesquisa de mercado para descobrir se esta é ou não uma ideia original e, caso não seja, qual o valor acrescentado que a aplicação a desenvolver introduzirá. Feito o estudo de mercado, segue-se a recolha de requisitos, ou seja, o levantamento de quais as principais funcionalidades a implementar, bem como a previsão e identificação das limitações impostas. Por fim, ainda nesta fase, deverá ser escolhido não só o público-alvo como também as plataformas a abranger (Android, iOS e Windows Phone).

2.1.2 Prototipagem

A prototipagem consiste, como o próprio nome indica, no desenvolvimento de um protótipo que cubra todas as funcionalidades, etapas e ecrãs disponíveis pela aplicação. Geralmente, os protótipos são ainda complementados com um documento contendo mais informações sobre a aplicação, respetivas funcionalidades e comportamentos para cada uma das etapas referidas no protótipo. A qualidade das maquetas desenvolvidas acaba por influenciar bastante o processo de desenvolvimento (ver Secção 2.1.4) uma vez que caso sejam ambíguas ou contenham pouca informação, poderão levantar dúvidas aos programadores e que mais tarde levarão a alterações dispendiosas na aplicação.

2.1.3 *Design* da Interface do Utilizador

Design da Interface do Utilizador é outro dos passos cruciais no desenvolver da aplicação, uma vez que é este que faz a ponte entre a aplicação e o utilizador. Esta fase pode facilmente condicionar o sucesso ou insucesso de uma aplicação móvel.

Quando se fala de *User Interface Design* (UI Design), existem dois conceitos relevantes e inter-relacionados: Usabilidade e Experiência de Utilização.

Usabilidade

Interação Pessoa-Máquina (ou, na nomenclatura inglesa *Human-Computer Interaction* (HCI)) é o estudo da interação — disponibilizada através de interfaces — entre pessoas e máquinas [11].

A Usabilidade, no contexto de HCI, estuda a forma como o utilizador comunica com a máquina e a respetiva tecnologia responde à interação, considerando os seguintes pontos [12]:

- **Facilidade na Aprendizagem** — O sistema deve ser fácil de assimilar pelo utilizador, para que este possa começar a trabalhar de imediato e progrida facilmente no uso da aplicação;
- **Eficiência** — O sistema deve ser eficiente para que o utilizador, depois de o saber usar, possa atingir um bom nível de produtividade;
- **Facilidade na Memorização** — O sistema deve ser facilmente memorável para que, mesmo depois de algum tempo sem o utilizar, o utilizador ainda se recorde de como o fazer;
- **Segurança** — O sistema deve ser tolerante a falhas, evitando que os utilizadores cometam erros e, se o fizerem, permitindo a fácil recuperação do estado anterior;
- **Satisfação** — O sistema deve ser desenhado de forma a proporcionar uma utilização agradável aos utilizadores, para que estes fiquem satisfeitos com a sua utilização.

Experiência de Utilização

Enquanto a Usabilidade dá importância a quão fácil e simples é a utilização de determinado sistema, a Experiência de Utilização estuda as sensações e sentimentos sentidos pelo utilizador durante a interação deste com esse sistema. Ou seja, para além da Usabilidade, a Experiência de Utilização está relacionada com aspetos adicionais como o *branding*, o *design*, a própria usabilidade e funcionalidades existentes que fazem parte da experiência do utilizador quando está a interagir com um dado sistema [13, 14].

Segundo *Peter Morville* [15], um pioneiro na área de Experiência de Utilização, existem 7 grandes características [16, 17] que influenciam a Experiência de Utilização de um dado sistema (ver Figura 2.2): Útil, Utilizável, Encontrável, Credível, Desejável, Acessível e Vantajoso.

Figura 2.2: Fatores que influenciam a Experiência de Utilização (adaptado de [18]).

- **Útil** — É importante que o sistema tenha alguma utilidade para o utilizador, pois, caso contrário não conseguirá manter a atenção deste por muito tempo. A Utilidade poderá ser demarcada através de uma nova função ainda não existente no mercado ou simplesmente por fornecer uma nova experiência de utilização.

- **Utilizável** — A Usabilidade pretende medir o quão rápido e fácil é o efetuar de uma tarefa num dado sistema. Num cenário em que o utilizador sinta bastante dificuldade na interação com o sistema, levando mais tempo que o expectável a efetuar uma tarefa, poderá levar a que utilizador desista de utilizar este sistema. Daí a importância deste ponto, Usabilidade, como um dos fatores que influenciam a Experiência de Utilização.

- **Encontrável** — Este conceito tem como principal objetivo garantir que tanto o sistema, como a informação nele contido, é fácil de encontrar. Caso um utilizador não tenha conhecimento da existência do sistema ou, pior, que conhecendo-o, não o consiga encontrar, o sistema acaba por se tornar inútil. Outro exemplo, desta vez para o conteúdo, é o caso em que não exista navegação ou menus no sistema que informem o utilizador de como realizar determinadas tarefas.

- **Credível** — A Credibilidade é um dos pontos a ter atenção no momento da gestão de conteúdos do sistema, uma vez que influenciam bastante o modo como o utilizador avalia o sistema. Se o sistema transparecer pouca credibilidade, o utilizador deixa de sentir segurança na sua utilização, fazendo com que procure alternativas.

- **Desejável** — Critério que pretende medir o grau de desejo/prazer do utilizador ao interagir com o sistema. A melhor experiência de utilização é aquela em que é criado um laço de conexão entre o utilizador e o sistema. Um exemplo deste conceito é o

branding, em que o utilizador, por vezes, acaba por pagar mais pela marca por sentir uma maior atração por esta.

- **Acessível** — A Acessibilidade é um dos fatores que pretende alavancar o sistema aos utilizadores que tenham alguma incapacidade, como por exemplo, o daltonismo. Os utilizadores daltónicos, tendo dificuldade na distinção das cores, necessitam de legendas associadas aos botões, uma vez que as cores não são suficientes para definir o seu comportamento.

- **Vantajoso** — Por fim, o sistema deve fornecer valor, não só para o seu proprietário como também para os seus utilizadores. Perante um sistema sem fins lucrativos, a experiência do utilizador acaba por ser a ponte para o sucesso.

Após análise, é possível concluir que apesar de ambos conceitos, Usabilidade e Experiência de Utilização, estarem relacionados, são considerados como independentes e bastante importantes para o sucesso de uma aplicação. A Usabilidade contribuindo com conceitos relacionados com a HCI e a Experiência de Utilização garantindo que toda a interação com a aplicação é previamente pensada e calculada, de forma a provocar um sentimento positivo de conexão entre o utilizador e a aplicação. Estes conceitos deverão ser seguidos e testados por cada vez que se publique uma nova aplicação.

2.1.4 Desenvolvimento

Terminados os protótipos, chega a fase da construção da aplicação, o desenvolvimento de todas as interfaces, funções e comportamentos previamente desenhados e estudados. Ao desenvolver uma aplicação é ainda muito importante seguir as diretrizes de design existentes para cada uma das plataformas, Android [6], iOS [7] e Windows Phone [8], uma vez que pode levar à recusa da publicação por parte das correspondentes lojas. Apesar das diretrizes ditarem um conjunto de regras a cumprir, estas não chegam para que seja possível o desenvolvimento de aplicações completamente funcionais e projetadas para dispositivos móveis. Por exemplo, para criar uma aplicação para um dispositivo móvel, em que seja necessário guardar dados localmente, é necessário saber como gerir estes mesmos dados, ou seja, como mapear e utilizar o armazenamento do dispositivo. Assim, outros fatores a ter em conta, nesta fase de desenvolvimento, são o Tamanho do Ecrã, a Gestão de Recursos do dispositivo e a *Cache* de Dados [19].

Tamanho do Ecrã

Os dispositivos móveis, de modo geral, podem ser divididos em duas categorias, os telemóveis e os *tablets*. Embora tenham muitas semelhanças em *hardware* e *software*, o tamanho do ecrã é um dos pontos em que mais diferem. Os telemóveis têm um espaço de ecrã muito limitado e os *tablets* um pouco maiores, mas, mesmo assim, ambos são bastante menores que o ecrã de um *desktop*. Assim, ao desenvolver uma aplicação, existe

uma maior responsabilidade na gestão dos componentes da interface, de forma a otimizar e maximizar a experiência do utilizador [20].

Gestão de Recursos

Alguns dos desafios no desenvolvimento de aplicações móveis passam pelo diminuir do tempo de resposta das aplicações, pelo uso eficiente da bateria e pela gestão de banda-larga disponível. A bateria, a velocidade de processador, o tamanho da memória e do disco, são alguns dos exemplos de componentes dos dispositivos móveis que acabam por ser mais reduzidos em prol das restrições de energia. Sendo estes recursos limitados, é preciso ter um cuidado extra no desenvolvimento das aplicações para que sejam apenas utilizados os recursos estritamente necessários e desligados assim que deixem de ser utilizados. Por exemplo, o desligar de pedidos ao GPS quando este já não seja mais necessário [21].

***Cache* de Dados**

O conceito de *cache* pode ser entendido como o persistir dos dados mais utilizados de uma aplicação, de forma a poupar tempo, banda-larga e uso desnecessário do *hardware* do dispositivo, melhorando assim o desempenho, disponibilidade e eficiência energética do dispositivo. Assim, havendo *cache* de dados, perante dispositivos com banda-larga limitada, condições de Internet variáveis, latência e grandes períodos de desconexão, as aplicações acabam por ter suporte para estes tipos de problemas, não condicionando o normal funcionamento da aplicação [22].

2.1.5 Validação (Testes)

A validação consiste no testar de toda a aplicação, nomeadamente, no teste de todos os ecrãs, botões e comportamentos. Devem ser gerados tantos testes quanto possíveis, de forma a encontrar e corrigir o máximo número de erros. Pode haver sempre detalhes que escapem aos programadores e até protótipos ambíguos, algo que não funciona da forma esperada visto que nem sempre os protótipos são suficientes para definir e ensaiar todas as funções, ações e comportamentos esperados.

2.1.6 Publicação

Desenvolvida a aplicação, após os testes e correções de erros, segue-se a publicação na loja correspondente, *Google Play* [23] para Android, *iTunes* [24] para iOS e *Windows Store* [25] para Windows Phone. Assim que a aplicação é submetida para a loja, existe um processo de aprovação da aplicação, sendo um dos passos a avaliação das respetivas diretrizes de *design* (mencionadas na Secção 2.1.4). Uma vez aprovada a aplicação, esta ficará disponível para todos os utilizadores da plataforma.

2.1.7 Atualizações

Por fim, este é o passo que pretende dar alguma continuidade à aplicação. Uma vez que já existe contacto direto com os utilizadores, fornecer atualizações às aplicações, sejam eles correções de erros ou novas funcionalidades, faz com que os utilizadores sintam que continua a existir alguma preocupação e suporte à aplicação. Embora este passo seja opcional e posterior ao lançamento da aplicação, é considerado como uma boa prática no desenvolvimento de aplicações.

2.2 Análise Funcional de Aplicações Móveis direcionadas ao Tratamento de Imagens Semelhantes

Um dos primeiros passos para qualquer aplicação móvel, relacionada com fotografias, é o visualizar do conjunto total de fotografias. Agora, no contexto do trabalho a desenvolver, quando introduzimos o requisito de remover as fotografias consideradas tecnicamente piores, são suportados dois passos intermédios: a organização e a ordenação das fotografias. Enquanto a organização de fotografias consiste no criar de subconjuntos por semelhanças entre fotografias, a ordenação tem como propósito o classificar das fotografias de acordo com determinado critério, como por exemplo a data.

Perante a visualização do conjunto total de fotografias, o próximo passo é a comparação de fotografias semelhantes (dentro de cada subconjunto), na procura de qual a fotografia que mais satisfaz o utilizador. Esta funcionalidade raramente é suportada pelas aplicações de gestão de fotografias. Mesmo quando disponibilizada, esta funcionalidade é bastante limitada, pois raramente permite a visualização de duas fotografias lado a lado e, quando permite, mostram normalmente a totalidade das fotografias num ecrã relativamente reduzido, tornando-se bastante difícil apreciá-las e encontrar as diferenças. Nos casos em que não existe a comparação lado a lado, o utilizador ou confia nas sugestões feitas pela aplicação sobre qual a melhor fotografia, ou tem de abrir as fotografias uma a uma (alternadamente) e tentar identificar de forma empírica a que mais lhe agrada.

Por fim, o último passo é a escolha de qual a fotografia com melhores características, para posterior remoção das outras de pior qualidade. Um mecanismo a ter em conta neste passo é o da segurança das fotografias. Por exemplo, no caso do utilizador por lapso eliminar uma fotografia com boas caraterísticas técnicas, deverá existir alguma opção que permita reverter a eliminação (não desejada) da fotografia.

2.2.1 Agrupamento de Fotografias

Nesta secção apresenta-se o processo de organização, ordenação e comparação de fotografias.

2.2.1.1 Métodos de Organização e Ordenação

Organização por *timestamp* e organização por semelhança entre fotografias estão entre os métodos mais interessantes para a organização de fotografias em subconjuntos.

A organização por *timestamp* consiste na divisão em subconjuntos baseados no instante em que as fotografias foram tiradas, fazendo agrupamentos de fotografias que foram tiradas em instantes próximos. A organização por semelhança considera o conteúdo/imagem representado na fotografia. A Figura 1.1, na página 2, representa um exemplo de um conjunto de fotografias com duplicados, organizados por data de captura, em que as fotografias semelhantes não se encontram todas juntas, precisamente por terem sido capturadas em momentos diferentes. Neste caso, a organização por *timestamp*, tendo um tempo de intervalo reduzido, poderia criar dois grupos de fotografias para o busto da medusa: um grupo para as duas primeiras fotografias do busto e outro para a última fotografia do busto, capturada após todos os outros temas. No caso da organização por semelhança tal separação não acontecia, uma vez que seria tido em conta o conteúdo das fotografias (o busto da medusa) e, assim, apenas seria formado um grupo para as três fotografias da medusa.

Sendo a organização por *timestamp* bastante mais fácil de aplicar, este é o método mais encontrado nas aplicações móveis, tais como *Remo Duplicate Photos Remover* [26], *GetSpace* [27] ou *Dr.Clean* [28]. Já a organização por semelhança não foi encontrada em nenhuma aplicação móvel[1].

Após a organização dos conjuntos, existem algumas funcionalidades de ordenação, sendo elas a ordenação por estrelas, por favoritos, por etiquetas ou por metadados (incluindo o *timestamp*). Para além da ordenação por *timestamp* e por favoritos, que se encontrava na aplicação *GetSpace* [27], nenhuma das outras ordenações foi encontrada nas aplicações móveis.

2.2.1.2 Métodos de Comparação

O método de comparação visa apurar a qualidade relativa de uma fotografia em relação a outra. Por exemplo, a comparação lado a lado ou uma de cada vez, mas neste caso com algum mecanismo (por exemplo, o toque no ecrã) que permita facilmente alternar entre as fotografias a comparar.

Na grande maioria das aplicações estudadas não existe a possibilidade de comparação simultânea, dificultando bastante a escolha de qual a fotografia com melhores características técnicas. Mesmo nas aplicações em que existe a funcionalidade de comparação, esta é bastante básica: permitindo a visualização lado a lado mas sem quaisquer gestos de *zoom* ou sincronização para um motivo de interesse; ou permitindo somente a comparação fotografia a fotografia, alternando a visualização entre uma e outra.

[1] No entanto, em aplicações de *desktop*, neste caso macOS, existe o *PhotoSweeper* [29] que faz organização por semelhança.

Para a comparação simultânea de duas fotografias, alguns dos exemplos de aplicações móveis são o *Best Photos* [30] e a *X-Dups* [31]. Já no caso da comparação de fotografia a fotografia, temos o exemplo da *Duplicate Photo Fixer* [32], do *Remove Master* [33] e do *Duplicate Photos Remover* [34].

2.2.2 Análise da Fotografia

Esta secção corresponde ao último passo descrito na Secção 2.2, o passo de análise das fotografias para posterior escolha de quais as fotografias a manter ou eliminar. Nesta secção foram endereçadas duas potenciais funcionalidades para análise de fotografia, nomeadamente, a deteção de Motion Blur e a deteção de Rostos e/ou Objetos.

A deteção de *Motion Blur* foi uma das funcionalidades encontradas nas aplicações móveis no que toca à análise de fotografias. Sendo o seu principal propósito o detetar de quais as regiões da fotografia que ficaram desfocadas por algum movimento do tema, um exemplo de aplicação com esta funcionalidade é a *GetSpace* [27]. A aplicação *GetSpace* cria três grandes conjuntos, Duplicados, *Blurred* e Capturas de ecrã, sendo que o de *Blurred* corresponde a um subconjunto com todas as fotografias que apresentem determinado grau de *Motion Blur*.

A deteção de Rostos e/ou Objetos é uma funcionalidade bastante interessante no conceito de análise de fotografias, pois permite que sejam adicionados mais critérios à escolha, para além da qualidade da fotografia. Por exemplo, num cenário em que o tema da fotografia seja a torre Belém, pode interessar ao utilizador ter mais fotografias da torre, mesmo que com má qualidade. Uma forma de avaliar quantas fotografias da torre foram capturadas, é, precisamente, utilizando a funcionalidade de deteção de objetos. Assim o utilizador teria uma visão mais geral da quantidade de fotografias que tinha da torre antes de proceder à eliminação de fotografias que poderia querer manter. Embora exista esta grande vantagem, a deteção de Rostos e/ou Objetos não foi encontrada em nenhuma das aplicações móveis estudadas.

2.2.3 Outras Propriedades

Nesta Secção são apresentadas outras propriedades interessantes a fazerem parte da solução a desenvolver, nomeadamente, Gravação e Reposição de Estado, Parametrização de Algoritmos e Desenvolvimento *Cross Platform*.

2.2.3.1 Gravação e Reposição de Estado

Esta funcionalidade, Gravação e Reposição de Estado, pretende permitir que as aplicações guardem os processamentos já efetuados, de forma a não necessitar de os calcular de novo, evitando o consumo de recursos do dispositivo, como, por exemplo, a bateria. Uma vez efetuados os processamentos sobre as imagens, se estas ou o seu grupo não sofrerem

quaisquer alterações, não existe a necessidade de efetuar, novamente, todos os cálculos. Um exemplo de aplicação que guarda e repõe o estado é a *X-Dups* [31].

2.2.3.2 Parametrização de algoritmos

Alguns algoritmos, como por exemplo, os de agrupamento por *timestamp* podem ser parametrizados com um valor de entrada (*input*) para condicionar o funcionamento do algoritmo. Para além desta parametrização de valores de entrada nos algoritmos, outra parametrização interessante é o caso em que se possa escolher que algoritmos aplicar sobre as fotografias. Por exemplo, num cenário em que estejam implementados dois algoritmos para aplicar sobre as fotografias, por exemplo, deteção de foco e estimação de *motion blur*, exista a opção de escolher qual a ordem pela qual devem ser aplicados ou a opção de aplicar somente um deles.

No contexto das aplicações móveis, alguns exemplos de parametrizações encontrados foram a definição do intervalo de *timestamp* (no *Duplicate Photo Fixer* [32]) ou de distância na localização (no *Duplicate Photo Fixer* [32]). No entanto, não foi encontrado nas aplicações móveis a possibilidade de escolher quais os algoritmos a aplicar nas análises das fotografias e nem a ordem pela qual estes seriam aplicados.

2.2.3.3 *Cross platform software*

Tal como descrito na Secção 1.3, na página 4, uma das características interessantes para a aplicação é o desenvolvimento *cross platform*, ou seja, o suportar várias (ou todas) as plataformas móveis, bem como ambientes *desktop* e *web*.

Ao efetuar um estudo das aplicações móveis, notou-se que existiam poucas com esta característica e que, as que existiam, acabavam por só suportar as plataformas móveis Android e iOS, ou, então suportar apenas iOS e macOS. Alguns exemplos das aplicações móveis *cross platform* encontradas foram o *Remo Duplicate Photos Remover* [26], o *Phone Cleaner* [35], o *Dr. Clean* [28] e o *Duplicate Photo Fixer* [32].

Por fim, nas aplicações móveis que fossem *cross platform software*, foi ainda verificado se estas tinham presente o conceito de sessões e, concluiu-se, que não.

2.2.4 Síntese

A Tabela 2.1 apresenta as principais funcionalidades e limitações para cada um dos *softwares* analisados. No decorrer dos testes aos *softwares* houve mais foco nos que pertenciam às plataformas móveis Android e iOS, apesar de ter sido tido em conta o facto de suportarem outros ambientes (macOS, Windows). Em todos os testes foi sempre utilizado o mesmo conjunto de imagens, subdivididas em três grupos quando organizadas por *timestamp*, mas que por semelhança apenas seriam subdivididas em dois grupos.

Após uma análise cuidada é possível verificar que estas aplicações móveis acabam por ser muito limitadas uma vez que muitas delas não suportam a comparação de duas

fotografias simultaneamente, tal como não são *cross platform software* e não permitem a escolha sobre quais diretorias deverá a ferramenta executar (em muitos casos, a galeria de fotografias não foi detetada).

Foi ainda possível concluir que nenhuma das soluções existentes cumpria os objetivos de organizar as fotografias por semelhança, ou de poder escolher quais os algoritmos a aplicar sobre as fotografias. Apenas duas das soluções tomavam partido dos processamentos já efetuados, as restantes efetuavam todos os cálculos por cada início/lançamento de aplicação. De forma geral, muitas das aplicações não estavam de todo desenvolvidas para suportar o novo tipo de "fotografias de Movimento" do Android, para além dos inúmeros *bugs* que acabavam por desligar a aplicação abruptamente.

Tabela 2.1: Tabela comparativa de funcionalidades/limitações.

	Subdivisão conjuntos[1]	**Plataformas suportadas**[2]	**Gravação e Reposição de Estado**[3]	**Deteção de *blur***[4]	**Comparação de imagens lado a lado**[5]	**Parametrização de algoritmos**[6]
Duplicate Photos Remover	*Timestamp*	Android	✗	✗	✗	✗
Remove Master	Mensal	iOS	✗	✗	✗	✗
Get Space	*Timestamp*	iOS	✗	✓	✗	✗
Phone Cleaner	✗	iOS e macOS	✗	✗	✗	✗
Remo Dupl. Photos Remover	*Timestamp*	Android e iOS	✗	✗	✗	Intervalo de *timestamp*
Dr. Clean	*Timestamp*	iOS e macOS	✗	✗	✗	✗
Duplicate Photo Fixer	*Timestamp*	Android e iOS	✓	✗	✗	Intervalo de *timestamp* (iOS) e Percentagem de distância na localização (iOS)
Best Photos	✗	iOS	✗	✗	✓	✗
X-Dups	✗	iOS	✓	✗	✓	✗

[1] Subdivide em conjuntos por *timestamp* e/ou por semelhanças entre fotografias;
[2] Plataformas para as quais é fornecido suporte;
[3] Define se a aplicação grava e reutiliza os processamentos já efetuados anteriormente;
[4] Deteta imagens *blurred*;
[5] Suporte a comparação de duas imagens simultaneamente (por ex. lado a lado);
[6] Suporte à escolha de quais os algoritmos a aplicar ou que valores atribuir a estes mesmos.

2.3 Correlação de Imagens

A correlação de imagens consiste no processo que visa obter a correspondência de um ponto de uma imagem noutro ponto de outra imagem. O processo de correlação de imagens pode ser dividido em duas fases: a deteção de *keypoints* para cada fotografia separadamente e depois a correlação dos *keypoints* entre as fotografias.

2.3.1 Deteção de *Keypoints*

Os *keypoints* são pontos específicos distintos numa imagem, ou seja, são pontos que se destacam dos demais pontos na sua vizinhança. Estes pontos são importantes uma vez que se mantêm identificáveis na imagem, quer esta seja rodada, encolhida, ou mesmo caso seja uma outra imagem do mesmo tema, apenas com uma ligeira diferença no ângulo de captura.

A deteção de *keypoints* consiste na aplicação de vários algoritmos sobre a imagem, a fim de identificar os seus pontos relevantes. Na Figura 2.3 é apresentado um exemplo de extração de *keypoints*: uma borboleta a repousar numa folha, sendo a imagem da esquerda a original e, a da direita a imagem com os *keypoints* principais assinalados (mais facilmente visíveis fazendo *zoom* sobre a fotografia da direita).

Figura 2.3: Deteção de *keypoints*.

2.3.2 Correlação de *Keypoints* e Matriz de Homografia

A correlação dos *keypoints* é o segundo e último passo na correlação de duas imagens. Uma vez identificados os pontos de interesse nas duas imagens, segue-se o passo de os fazer corresponder entre as imagens, ou seja, estabelecer a correspondência de um determinado ponto numa fotografia A com o mesmo ponto numa fotografia B. Caso o fotografo tenha deslocado ou rodado (ainda que muito ligeiramente) a câmara entre as fotografias A e B, o ponto da fotografia B pode não ter exatamente as mesmas coordenadas que na fotografia A (ver Figura 2.4). Após a correlação de quatro *keypoints* em duas imagens "A" e "B", é possível a produção da matriz de homografia "AB".

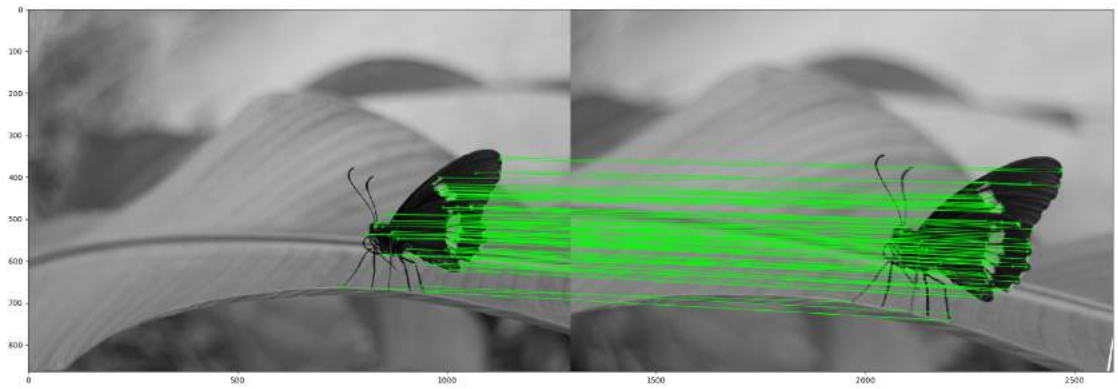


Figura 2.4: Correlação de *keypoints*.

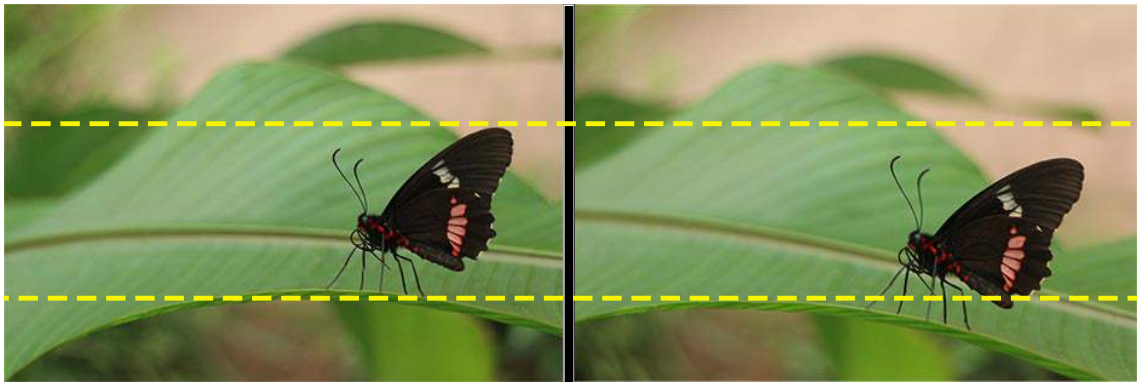
A matriz de homografia é uma matriz que transforma as coordenadas de pontos entre duas imagens, ou seja, é uma matriz que faz a correspondência de um ponto numa imagem "A" no ponto "equivalente" numa imagem "B". Imagine-se duas imagens semelhantes "A" e "B" (ver Figura 2.5a), e uma matriz de homografia "BA" que relaciona os *keypoints* da fotografia B com os da fotografia A. Quando esta matriz é aplicada sobre a imagem B, a imagem B passa a ser uma versão distorcida da imagem A (ver Figura 2.5b).

2.3.3 Questões de Eficiência

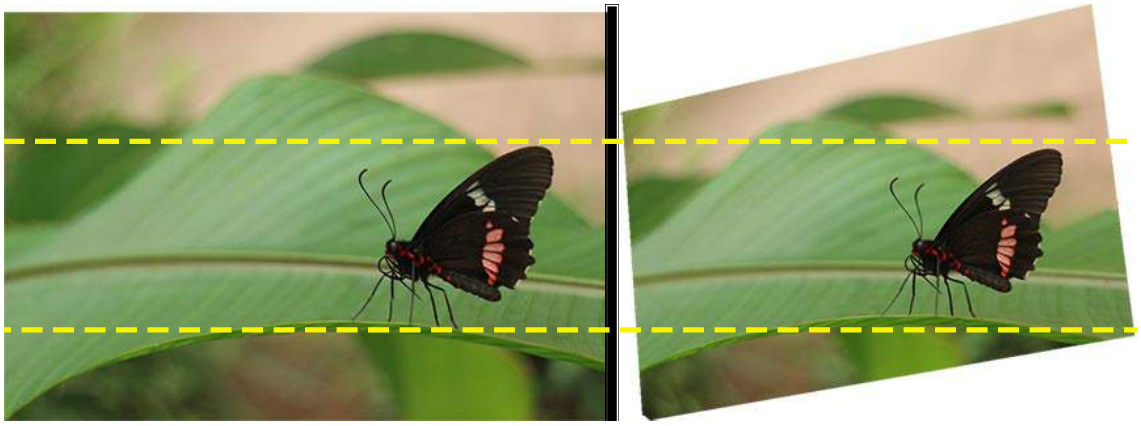
Uma variável importante no processo de produção da matriz de homografia entre duas imagens é a resolução das imagens. O tempo necessário para a identificação dos *keypoints* é demasiado elevado quando se tenta gerar uma matriz de homografia para duas imagens com grande dimensão. Esta lentidão no processamento deve-se ao facto das imagens de alta resolução apresentarem bastantes detalhes, aumentando o número de potenciais *keypoints* e, portanto, o tempo necessário para a sua deteção. Uma abordagem para endereçar este problema é o reduzir da resolução das imagens antes do início do processo de deteção de *keypoints*. Esta abordagem reduz significativamente o tempo de processamento do algoritmo, contudo tem algumas limitações/desvantagens visto retirar detalhe à imagem e, consequentemente, retirar potenciais *keypoints* das imagens.

2.3.4 *OpenCV*

O *Open Source Computer Vision Library* (OpenCV) [36], desenvolvido pela Intel, é uma biblioteca multiplataforma que visa fornecer uma infraestrutura simples e fácil de usar por aplicações na área de visão computacional, incluídas na análise ou processamento de imagens/fotografias. Alguns exemplos de utilização desta biblioteca são a aplicação de filtros às imagens, a deteção e reconhecimento de rostos e a identificação de objetos. O OpenCV disponibiliza mais de 2500 algoritmos e está disponível de forma gratuita para quem quiser aceder, utilizar ou modificar.



(a) Fotografias "A" e "B" originais.



(b) Fotografia "A" original e Fotografia "B" com matriz de homografia aplicada.

Figura 2.5: Exemplo de aplicação de matriz de homografia.

No contexto de correlação de imagens o *OpenCV* é interessante uma vez que permite delegar o processamento (pesado) para um servidor externo, fornecendo alguns algoritmos para a correlação de imagens, por exemplo, o "*FeatureDetector*" para identificar os *keypoints*.

2.4 Ambiente de Desenvolvimento Android

O sistema operativo Android foi criado em 2003 por uma *start-up* chamada *Android Inc*, que pretendia desenvolver "dispositivos móveis mais inteligentes e que estivessem mais cientes das preferências e da localização do seu dono". Àquela data, o principal foco da empresa era criar um sistema inovador para as câmaras digitais, mas como o mercado não era tão amplo como o que estavam à espera, acabaram por focar-se na componente móvel, lançando um novo sistema operativo baseado em *Kernel Linux*. Só em 2005 é que o Android passou a ser propriedade da *Google*, após esta comprar a *start-up Android Inc* para criar a sua própria divisão de pesquisa em tecnologia móvel, a *Google Mobile Division* [37]. O grande passo para o desenvolvimento do atual e conhecido sistema operativo Android aconteceu em 2007 quando diversos fabricantes de telemóveis, operadoras americanas,

fabricantes de hardware e a própria Google, se reuniram num consórcio de tecnologia e fundaram a *Open Handset Alliance*. A *Open Handset Alliance* tinha como objetivo a criação de uma plataforma de código aberto (*open-source*) para *smartphones* [38]. O resultado desta aliança entre as diversas empresas foi o lançamento do primeiro Android no mercado, disponibilizado num *HTC Dream* em Setembro de 2008. Desde então, o Android tem recebido inúmeras atualizações que incrementam substancialmente o sistema, quer seja através da implementação de novas funcionalidades, quer seja corrigindo erros de versões anteriores. A cada grande atualização o nome do sistema muda, em ordem alfabética, normalmente sendo nomes de doces [39]. Segundo as estatísticas da *StatCounter* de dezembro de 2018, o Android é o sistema operativo mais utilizado no mercado, abrangendo cerca de 75.16% da cota mundial de dispositivos móveis [40].

2.4.1 *Java/Kotlin*

Para desenvolver uma aplicação Android existem duas linguagens populares: *Java* e *Kotlin*.

Java foi a primeira linguagem utilizada para o desenvolvimento de aplicações móveis Android. O *Java* tem duas propriedades muito relevantes: é Estaticamente Tipada e é Orientada a Objetos. Uma linguagem estaticamente tipada obriga a que todas as variáveis tenham um tipo de dados. Uma linguagem orientada a objetos é baseada na modelação de objetos e relações entre eles.

Kotlin é a mais recente e melhor opção para o desenvolvimento Android. Algumas das vantagens apresentadas pela empresa que desenvolveu a linguagem foram o facto da linguagem ser mais legível, ter mais segurança/proteção contra valores nulos nas variáveis, ter funções *lambda*, ter classes de dados e ter a possibilidade de declarar as variáveis como mutáveis ou imutáveis. No entanto, uma desvantagem encontrada reside no facto do tamanho final dos projetos desenvolvidos em *Kotlin* acabar por ser ligeiramente maior quando em comparação com um projeto totalmente desenvolvido em *Java* [2].

2.4.2 *Android Studio*

O *Android Studio* é o IDE oficial para desenvolvimento de aplicações Android [41]. Alguns dos principais recursos oferecidos pelo *Android Studio* são:

- Sistema de compilação flexível baseado no *Gradle*[3];
- Emulador de diversas versões do Android, com possibilidade de customização das propriedades de *hardware* e *software*;

[2] De realçar que outra propriedade importante do *Kotlin* é o facto de ter totalmente interoperável com o *Java* permitindo que ambas as linguagens sejam utilizadas em simultâneo no desenvolvimento de um projeto.

[3] *Gradle* é um sistema avançado para gestão de todas as configurações de compilação de um projeto. O *Gradle* não só suporta o *download* automático de todas as dependências e bibliotecas externas, como também permite determinar a ordem em que determinadas tarefas de um projeto devem ser executadas.

- Ambiente unificado de desenvolvimento para todos os dispositivos Android;
- *Instant Run*, que permite aplicar alterações a aplicações em execução sem a necessidade da compilação de um novo *Android Application Pack* (APK);
- *Lint*, uma ferramenta de verificação de código suspeito para detetar problemas de desempenho, usabilidade, compatibilidade com versões, entre outros;
- Compatibilidade embutida com a *Google Cloud Platform*, facilitando a integração do *Google Cloud Messaging* e do *App Engine*.

Capítulo 3

Conceção da Interface e da Interação com o Utilizador da Aplicação (*photo | uniq*)

Num mundo tecnológico como o de hoje, é comum voltar de uma curta viagem de férias com centenas de fotografias digitais, onde os assuntos importantes foram fotografados vezes sem conta, na esperança de que uma dessas fotografias capturasse o momento em toda a sua plenitude. O fotografo amador encontra-se assim perante um cenário de imensas fotografias duplicadas[1] e de uma ausência de ferramentas (aplicações) eficazes no processo de ajuda na escolha da melhor fotografia e consequente eliminação das restantes. Nesta dissertação descreve-se o processo de conceção e desenvolvimento de uma aplicação móvel Android para auxiliar o utilizador na afeição da qualidade técnica das suas fotografias e assim facilitar o processo de eliminação de fotografias duplicadas. Existiram claros desafios no desenvolvimento de uma aplicação desta natureza, nomeadamente na identificação das características técnicas relevantes, na seleção de algoritmos apropriados à avaliação daquelas características, no desenho de uma interface simultaneamente acessível ao utilizador principiante e não limitadora para o utilizador experiente.

Na conceção da interface da aplicação, o foco foi sempre tentar criar uma interface intuitiva e familiar para o utilizador, de forma a garantir que durante a utilização da aplicação, o utilizador nunca se sentiria perdido sobre qual a ação que deveria tomar. Para garantir tais objetivos, foi feita uma análise cuidada do design de aplicações já existentes, tais como as galerias nativas dos dispositivos Android e iOS, e de aplicações para aceder a repositórios remotos como o *Flickr* e o *Google Photos*. Outro requisito importante foi o minimizar da utilização dos recursos do dispositivo móvel (por exemplo, a bateria), tendo-se optado por fazer uso de um servidor na *cloud* para efetuar os processamentos que se revelaram computacionalmente pesados para ambiente móvel.

[1] No âmbito deste capítulo, utilizaremos a palavra duplicadas para nos referirmos a fotografias bastante semelhantes (quase duplicadas).

Assim a aplicação proposta, batizada com o nome de *photo | uniq*, tem como vistas principais, acessíveis através do menu de navegação (ver Figura 3.1):

- A Galeria Local, onde são apresentadas todas as fotografias disponíveis na galeria do dispositivo. Nesta vista o utilizador pode selecionar quais as fotografias da galeria local que deverão ser transferidas para a galeria *photo | uniq*. O utilizador pode também selecionar fotografias que deverão ser ignoradas por esta aplicação (por exemplo, fotografias de documentos ou fotografias que já são únicas).

- A Galeria *photo | uniq*, que serve de ponto de partida para o *workflow* de apoio à avaliação das características técnicas das fotografias e que conduzirá à eliminação das fotografias duplicadas de menor qualidade.

- As Definições, onde é possível configurar os parâmetros de funcionamento da aplicação.

Figura 3.1: Estrutura dos ecrãs das galerias da aplicação.

Ainda relativamente à estrutura da aplicação, os ecrãs relativos às vistas das galerias (local e *photo | uniq*) estão normalmente divididos em três grandes áreas de operação, como ilustrado na Figura 3.1:

- O menu na zona superior, que visa situar o utilizador na aplicação e também apresentar um leque de botões associados a operações válidas no presente contexto;

- A área de trabalho, onde aparecem as fotografias organizadas de acordo com o contexto atual;

- O menu de navegação (na zona inferior), que permite o acesso aos ecrãs principais: Galeria Local (ver Secção 3.2), Galeria *photo* | *uniq* (ver Secção 3.7) e Definições (ver Secção 3.14).

Por questões de segurança e privacidade criámos um sistema de autenticação para identificar unicamente cada um dos utilizadores e permitir que estes se conectem à respetiva galeria *photo* | *uniq* em qualquer um dos seus dispositivos (móvel, *desktop* ou *web*)[2]. De modo a facilitar esta autenticação foram fornecidas diversas opções de *login*, entre elas alguns dos mecanismos mais habituais de registo/*login* com pré-preenchimento automático, como é o caso do *Google* ou *Facebook* (ver Figura 3.2).

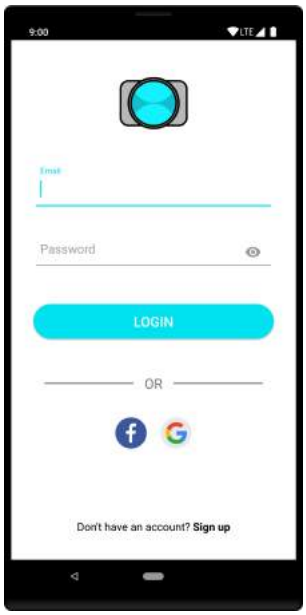


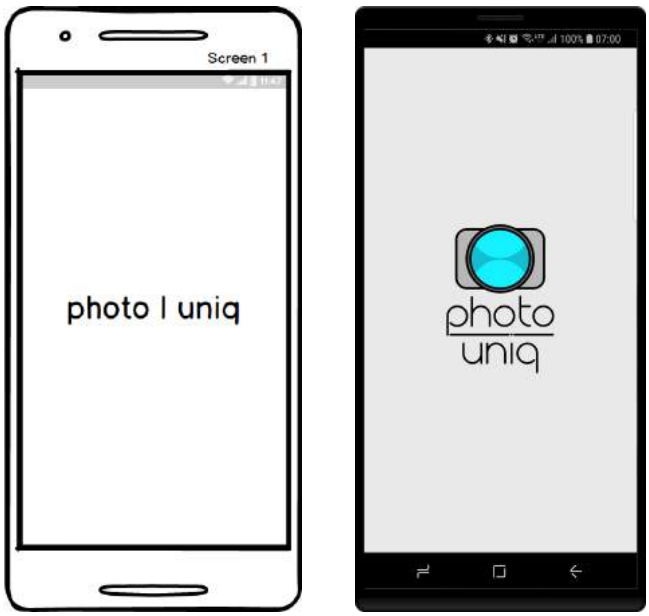
Figura 3.2: Ecrã de *login*/registo na aplicação.

3.1 Inicialização (*Splash screen*)

O *splash screen* (Figura 3.3) é apresentado quando se inicia a aplicação. Este ecrã tem o propósito de apresentar a marca enquanto se realizam alguns processamentos iniciais necessários à utilização da aplicação, tais como:

- A atualização da galeria local, incorporando as novas fotografias tiradas desde a última utilização da aplicação e removendo as que foram apagadas;
- A sincronização do estado entre a aplicação e o servidor *photo* | *uniq*, incorporando na aplicação alterações no estado do servidor causadas pela utilização de outra instância da aplicação noutros dispositivos[3].

[2] A aplicação apresenta uma visão única da galeria *photo* | *uniq*, independentemente da plataforma.
[3] A implementação corrente no protótipo tem funcionalidade limitada, uma vez que servidor ainda não está operacional e, portanto, não é possível efetuar a sincronização.



(a) Versão maqueta. (b) Versão implementada.

Figura 3.3: *Splash screen*

Para atualizar as fotografias existentes na galeria local, a aplicação itera sobre todas as fotografias do dispositivo e, para cada uma das fotografias, atualiza ou cria uma nova entrada na tabela de fotografias da base de dados da aplicação. Após a iteração, a aplicação elimina todas as fotografias que não foram atualizadas, uma vez que estas correspondem às fotografias que foram eliminadas na galeria do dispositivo desde a última utilização da aplicação.

O processo de sincronização entre a aplicação e o servidor começa com uma chamada a um serviço externo, alojado no servidor *photo | uniq*, para efetuar a sincronização do estado entre a aplicação e o servidor. De modo a garantir que o utilizador pode utilizar a aplicação o quanto antes, o processo de sincronização funciona de forma assíncrona, ou seja, a aplicação avança ainda que o serviço não tenha dado resposta. No momento em que a aplicação recebe a resposta do serviço, caso o utilizador esteja numa vista que utilize os dados fornecidos pelo serviço, a vista será atualizada.

Não existem gestos ou comportamentos extras associados a este ecrã de inicialização, sendo o utilizador redirecionado automaticamente para o ecrã de fotografias locais (ver Secção 3.2) assim que as tarefas de inicialização estiverem concluídas.

3.2 Galeria de Fotografias Locais

O funcionamento da aplicação *photo | uniq* está dependente de um pré-processamento que é realizado num servidor externo. Por isso, uma das primeiras tarefas do utilizador é a de selecionar um conjunto de fotografias a enviar para o servidor. Uma aproximação

para enviar as fotografias do utilizador para a galeria *photo | uniq*, poderia ser a partilha através da galeria nativa do dispositivo ou até mesmo de outras aplicações, desde que permitissem a funcionalidade de partilha. O utilizador poderia continuar a utilizar as suas aplicações de gestão de fotografias preferidas, podendo até tomar partido dos seus álbuns previamente criados, tendo somente de efetuar a seleção de fotografias e consequente partilha para a nossa aplicação *photo | uniq*. Porém, uma desvantagem desta abordagem seria o facto do utilizador perder o controlo sobre quais as fotografias que já tinham sido enviadas para a galeria *photo | uniq*.

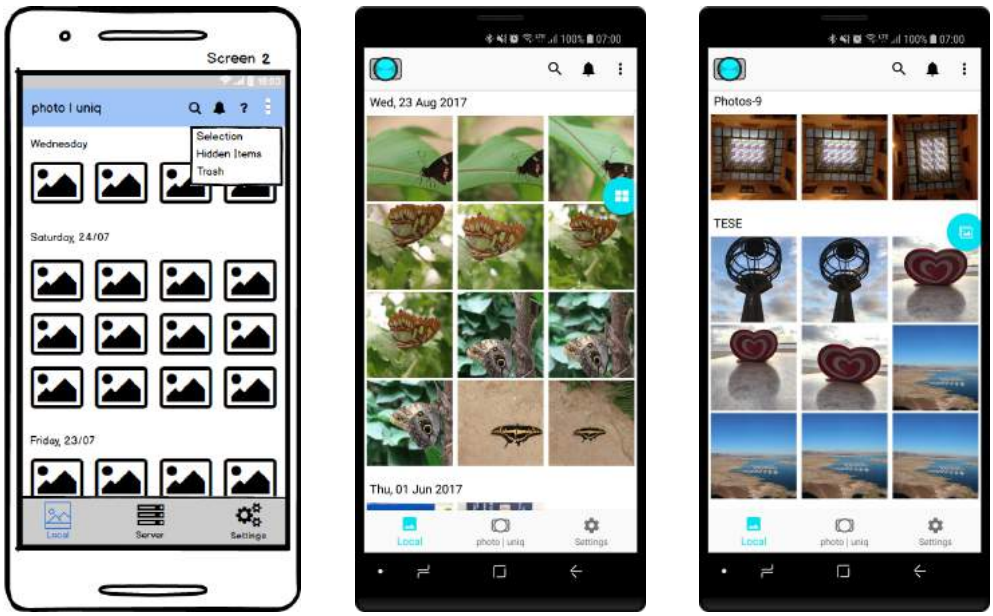
Uma das preocupações no desenvolvimento da aplicação foi a poupança dos recursos do dispositivo, pelo que o utilizador apenas tem de enviar uma única vez cada fotografia para a galeria *photo | uniq*. Para cumprir este requisito decidimos criar, no lado da aplicação, uma vista que permitisse não só a gestão das fotografias existentes no dispositivo, como também a identificação de quais as fotografias já enviadas para a galeria *photo | uniq*. No entanto, nem todas as imagens presentes no dispositivo do utilizador interessam para o processo de remoção de duplicados, como é o caso de fotografias de recibos de compras, capturas de ecrã, entre outros. Para endereçar estes casos, a vista desenvolvida passou a ter funcionalidades extra de filtragem para os diversos tipos de imagens existentes no dispositivo.

Quando o processo de inicialização fica concluído (mencionado na Secção 3.1), a aplicação apresenta um ecrã (ver Figura 3.4) onde o utilizador pode: esconder fotografias não relevantes no âmbito da aplicação; eliminar diretamente fotografias indesejadas; ou enviar um conjunto de fotografias para a galeria *photo | uniq*. Cada uma destas funcionalidades opera sobre um conjunto de imagens, pelo que o seu mecanismo funcional base assenta na seleção múltipla de imagens. A seleção é iniciada sempre que o utilizador pressione durante alguns segundos uma das fotografias ou nome de um grupo[4] de fotografias. Após a escolha das fotografias a operar, o utilizador pode então executar uma das três funcionalidades permitidas pelo ecrã e listadas acima.

Se o utilizador, após a seleção de um conjunto de fotografias, optar por as enviar para a galeria *photo | uniq*, terá ainda de escolher o álbum no servidor onde serão alojadas. Uma vez escolhido o álbum, aparece um novo botão (☁) no menu superior da aplicação de modo a permitir a navegação para o ecrã de envios em progresso (ver Secção 3.3). Todas as fotografias anteriormente escolhidas passam a ter esse mesmo ícone sobreposto, para indicação de que já estão em processo de envio. Já no caso em que o utilizador opta por esconder ou eliminar fotografias, assim que o utilizador clica num dos botões, todas as fotografias selecionadas desaparecem deste ecrã, provocando uma atualização da vista corrente.

Por fim, todas as funcionalidades acima descritas, podem ser efetuadas através de dois tipos de organizações de fotografias diferentes: por data de captura (ver Figura 3.4b) e por álbuns (ver Figura 3.4c). Na organização por álbuns são apresentados os que existem

[4]Neste âmbito, um grupo corresponde a um conjunto de fotografias tiradas no mesmo dia.



(a) Versão maqueta.

(b) Versão implementada.

(c) Versão implementada.

Figura 3.4: Ecrã de galeria local, com fotografias locais organizadas por data de captura (Figura 3.4b) ou por álbuns (Figura 3.4c).

no dispositivo, tanto na memória interna como na externa. Para alternar entre os dois modos de organização, basta que o utilizador clique no botão flutuante azul identificado por 4 quadrados ( )[5].

3.3 Ecrã de Envio (em progresso) de Fotografias para a Galeria *photo | uniq*

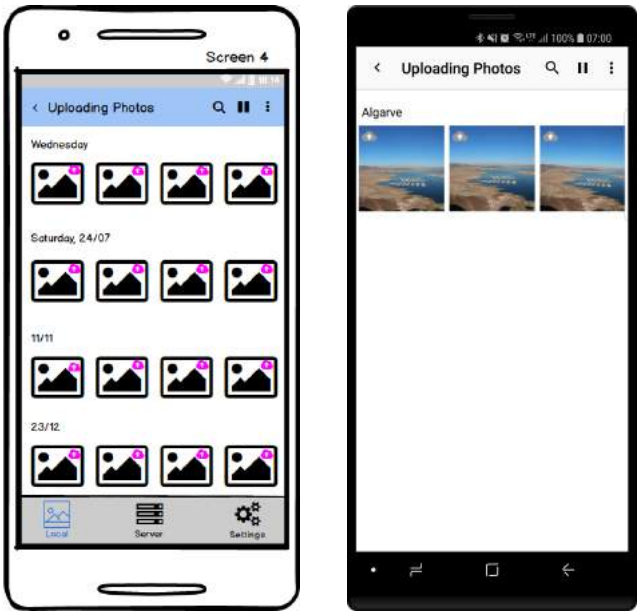
Quando um utilizador está a selecionar quais as fotografias a enviar para a galeria *photo | uniq*, perante uma listagem em que só aparecem as miniaturas das imagens, o utilizador pode, por lapso, selecionar e enviar capturas de ecrã e outras imagens do mesmo género. Considerou-se assim necessário a existência de um mecanismo que pudesse interromper o processo de envio[6].

O ecrã de fotografias em processo de envio, visível na Figura 3.5, identifica quais as fotografias em processo de envio para o servidor *photo | uniq*. Para além do cancelamento do envio de um conjunto de fotografias, é também possível suspender temporariamente o envio de todas as fotografias.

Tal como no ecrã de galeria local, neste ecrã de fotografias em processo de envio para o servidor *photo | uniq*, é possível entrar em modo de seleção de fotografias através do

[5]Sempre que o botão flutuante azul identificado por quatro quadrados ( ) esteja disponível numa vista, terá como funcionalidade alternar entre o tipo de organização por data de captura ou por álbuns.

[6]Para além do cancelamento do envio, existe também a possibilidade de apagar as fotografias se estas já tiverem sido enviadas para o servidor. Esta questão é abordada na Secção 3.7.



(a) Versão maqueta.

(b) Versão implementada.

Figura 3.5: Ecrã de fotografias em processo de envio para o servidor.

Figura 3.6: Ecrã de envios em progresso, com modo de seleção ativado.

toque longo sobre uma das fotografias (ver Figura 3.6). Assim que o modo de seleção é ativado, os envios passam de imediato ao estado de pausa (as fotografias passam a ter o ícone de pausa ( ) sobreposto) para permitir que o utilizador possa escolher calmamente quais as fotografias para as quais pretende cancelar o envio. Quando o utilizador sai do modo de seleção, quer seja por ter cancelado algum envio, quer seja por voltar atrás[7], o processo de envio é retomado. Sempre que uma fotografia é transferida com sucesso, é retirada da fila de espera e, no ecrã de galeria local, é mudado o ícone atual ( ) para o de nuvem cheia ( ) e assim simbolizar que aquela fotografia já se encontra na galeria *photo | uniq*. No caso de ocorrer alguma falha na transferência, após três tentativas de envio (falhadas), a fotografia é removida da fila e o seu estado anterior é reposto, ou seja, a fotografia passa ao estado de estar somente disponível na galeria do dispositivo (sem nenhum ícone sobreposto).

3.4 Fotografias Ignoradas

No conjunto de fotografias da galeria do dispositivo podem existir algumas fotografias que embora sejam importantes para manter no dispositivo, por exemplo fotografias de recibos de compras, não são relevantes para o processo de remoção de duplicados. Uma das funcionalidades suportadas pela galeria local é a de esconder fotografias irrelevantes para o contexto de remoção de duplicados. Caso o utilizador esconda por engano fotografias relevantes, existe um mecanismo que lhe permite restaurar as fotografias para a galeria local.

O ecrã de fotografias escondidas, visível na Figura 3.7, permite ao utilizador rever todas as fotografias já escondidas e restaurar um conjunto destas para a galeria local. Para simplificar o processo de eliminação de fotografias escondidas, o ecrã tem um botão de lixo ( ) no menu superior, que permite mover um conjunto de fotografias escondidas diretamente para o lixo.

Para efetuar as funcionalidades acima descritas, o utilizador deverá pressionar ininterruptamente uma ou várias fotografias durante alguns segundos e executar uma das operações disponibilizadas.

Tanto no ecrã de envios de fotografias para a galeria *photo | uniq*, como no de fotografias escondidas, são visíveis as semelhanças na organização das fotografias. Embora esta semelhança seja propositada, se for excessiva, o utilizador pode sentir alguma dificuldade em identificar em que ecrã se encontra. De forma a tentar minimizar este efeito, no ecrã de fotografias escondidas as fotografias aparecem com um efeito *fade* e com um ícone de um olho riscado ( ) sobreposto. Deste modo, o utilizador consegue identificar o ecrã não só pelo título (situado no menu superior) mas também pela forma como as fotografias são apresentadas[8].

[7] Voltar atrás no sentido de clicar no botão "*back*" do dispositivo.

[8] A técnica de sobreposição de um determinado ícone sobre as fotografias foi aplicada ao longo de diversos dos ecrãs acessíveis pela galeria local, sempre com o intuito de facilitar para o utilizador a identificação do

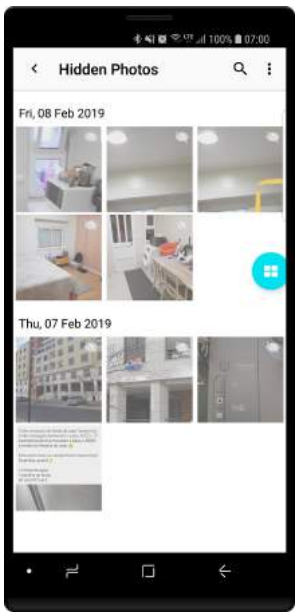


Figura 3.7: Ecrã de fotografias ignoradas.

Por fim, o ecrã de fotografias escondidas apenas foi pensado no momento do desenvolvimento da aplicação, pelo que não existe uma maqueta pré-concebida.

3.5 Fotografias no Lixo

Similarmente à funcionalidade de esconder fotografias, o utilizador pode eliminar fotografias indesejadas diretamente da galeria local. Se assumíssemos que o utilizador nunca se arrependeria de eliminar uma fotografia, não seria necessário conceber um mecanismo de restauro de fotografias eliminadas. Como não é possível fazer tal afirmação, a nossa solução passou por desenvolver uma vista dedicada à listagem e restauro de todas as fotografias eliminadas pelo utilizador no âmbito da aplicação *photo | uniq*.

As grandes funcionalidades do ecrã de fotografias no lixo, visível na Figura 3.8, são o restaurar de um conjunto de fotografias para a galeria local e o eliminar definitivo de fotografias (eliminadas através da aplicação) no dispositivo do utilizador. Novamente, para executar estas operações, o utilizador tem de passar pelo processo de seleção de fotografias e escolha de qual a operação a efetuar.

Com o surgimento do ecrã de fotografias no lixo, os utilizadores que queriam efetivamente eliminar as fotografias na sua primeira operação, por exemplo através da galeria local da aplicação, teriam a tarefa adicional de: abrir a vista de fotografias no lixo, selecionar as fotografias previamente eliminadas e clicar no botão de eliminar definitivamente(). Para endereçar esta situação, existe uma opção de controlo da eliminação de fotografias no ecrã de Definições (ver Secção 3.14). Através da definição de eliminação automática, o utilizador pode escolher se o ato de eliminação, para outros ecrãs que não este, será logo

ecrã em que se encontra.

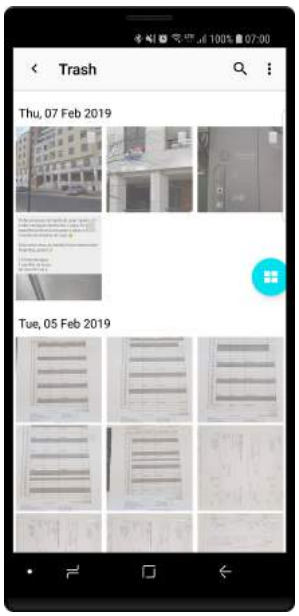


Figura 3.8: Ecrã de fotografias no lixo.

definitivo, ou seja, pode decidir se esta funcionalidade de lixo temporário deverá ou não existir. O ecrã de fotografias no lixo só será apresentado ao utilizador caso a eliminação automática/definitiva esteja desligada no ecrã de Definições.

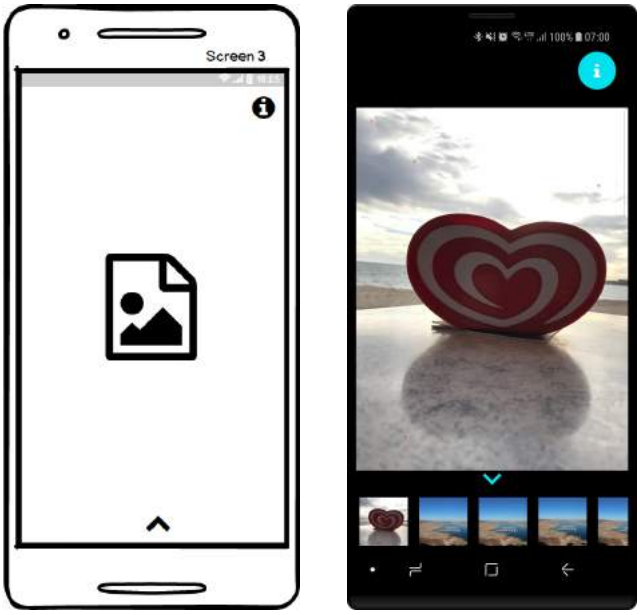
3.6 Pré-visualização de Fotografias

A pré-visualização de fotografias, cujo ecrã é visível na Figura 3.9, tem como principal objetivo fornecer um mecanismo que permita a visualização de cada uma das fotografias da galeria local no seu formato/resolução original. A visualização das fotografias neste formato facilita a análise e avaliação das características técnicas das fotografias. Quando uma fotografia está a ser visualizada neste modo, são suportadas quatro operações:

- *zoom in*, sendo que o utilizador deve:
  - efetuar duplo toque com um dedo na área de interesse;
  - ou juntar dois dedos e afastá-los;
- *zoom out*, através:
  - de duplo toque com um dedo em qualquer local da fotografia (apenas quando está *zoomed in*);
  - da junção de dois dedos afastados;
- alteração do ponto que está no centro do ecrã, com o arrastar de um dedo pelo ecrã[9];

[9]Gesto somente disponível quando a fotografia está com algum *zoom* aplicado, ou seja, quando existem zonas da fotografia que não aparecem no ecrã (por estarmos perante *zoom*).

- navegação para a fotografia anterior ou seguinte, com o deslizar de um dedo da esquerda para a direita, ou da direita para a esquerda, respetivamente.



(a) Versão maqueta. (b) Versão implementada.

Figura 3.9: Ecrã de pré-visualização de fotografias.

O botão ( ) permite aceder/visualizar os metadados da fotografia correntemente visualizada.

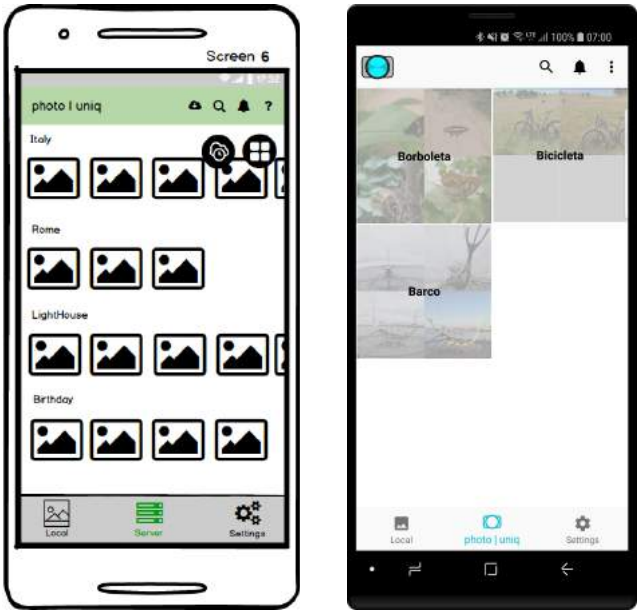
Na parte inferior do ecrã foi implementada uma lista horizontal com todas as fotografias da coleção, para que o utilizador possa navegar rapidamente para outra fotografia que não a imediatamente a anterior/seguinte da que está a ser visualizada. Caso o utilizador se encontre num dos extremos do conjunto de fotografias, a primeira ou última fotografia, não existirá fotografia anterior, ou seguinte, consoante o extremo em causa.

3.7 Galeria *photo* | *uniq*

A galeria *photo* | *uniq* (visível na Figura 3.10) lista todos os álbuns[10] que o utilizador transferiu para a galeria *photo* | *uniq*, através do seu dispositivo móvel ou de outros, tais como o seu computador portátil ou *tablet*. O principal objetivo da galeria *photo* | *uniq* é fornecer uma navegação fluída entre os álbuns para que o utilizador identifique rapidamente qual o álbum onde pretende iniciar o processo de remoção de duplicados. De forma a encontrar um álbum em particular, o utilizador pode, a qualquer momento, utilizar o botão de pesquisa disponibilizado no menu superior. Para além do mecanismo de pesquisa, cada um dos álbuns apresenta ainda quatro fotografias desse álbum para auxiliar

[10] Sendo um álbum um conjunto de fotografias com o mesmo tema.

o utilizador a identificar o conteúdo do álbum. Assim que escolher qual o álbum a operar, o utilizador é redirecionado para o ecrã de grupos de semelhança do álbum selecionado (ver Secção 3.8).



(a) Versão maqueta.

(b) Versão implementada.

Figura 3.10: Ecrã de galeria *photo* | *uniq*, com fotografias remotas.

No caso do ecrã da galeria *photo* | *uniq*, existem algumas diferenças ao nível da organização das fotografias na área de trabalho entre a versão concebida na maqueta e a efetivamente implementada. Na maqueta da galeria *photo* | *uniq*, visível na Figura 3.10a, a navegação vertical é constituída por todos os álbuns e a horizontal pelas fotografias existentes nestes. Na implementação, verificou-se que a navegação entre os álbuns era demasiado pesada, devido à coexistência do *scroll* vertical e do horizontal. Sendo um dos requisitos da galeria *photo* | *uniq* a navegação rápida entre álbuns e a fácil identificação do álbum a operar, foi necessária a criação de uma nova interface, visível na Figura 3.10b. A versão implementada da galeria *photo* | *uniq* passou de complexa e lenta para bastante mais simples, compacta e rápida, sendo a interface inspirada em algumas existentes de aplicações de gestão de fotografias (como, por exemplo, no *Google Photos*). No novo *design* da galeria *photo* | *uniq* são apresentadas somente quatro fotografias identificadoras do conteúdo de cada um dos álbuns, persistindo apenas o *scroll* vertical. Os botões flutuantes visíveis na maqueta, foram omitidos por já não fazerem sentido no âmbito da nova interface concebida para o ecrã. Estes botões removidos tinham por função: trocar a ordem pela qual eram apresentadas as fotografias (*timestamp* ou data de envio para o servidor); e alterar a organização das fotografias, passando a ser uma grelha de álbuns e respetivas fotografias, tal como no ecrã de fotografias locais (ver Figura 3.4).

Por fim, visto a galeria *photo | uniq* poder conter fotografias que não existem no dispositivo, existe no menu superior um botão identificado por uma nuvem com uma seta para baixo (), que permite aceder a uma listagem bastante semelhante à da galeria local, mas para as fotografias presentes na galeria *photo | uniq*. Quando o utilizador acede a esta vista, pode fazer o *download* ou a eliminação de um conjunto das fotografias lá presentas[11].

3.8 Grupos de Semelhança de um Álbum

Por norma, quando um utilizador cria um álbum novo através da seleção de um conjunto de fotografias que de alguma forma se relacionam, quer seja pela sua localização, data em que as fotografias foram capturadas, ou pelo seu conteúdo. No entanto, este conjunto de atributos comuns pode não ser suficiente para se identificar quais as fotografias efetivamente semelhantes.

Imaginando que o utilizador faz uma viagem a Roma, em que fotografa diversos monumentos e, na altura de organizar as suas fotografias, cria um único álbum com o nome da cidade (Roma) para todas as fotografias. No momento em que o utilizador escolhe enviar as fotografias para a galeria *photo | uniq*, assumindo que o utilizador não cria sub-álbuns para cada um dos monumentos com que se deparou na viagem, o ponto de relação entre as fotografias seria o local onde tinham sido capturas (Roma). Assim, para casos como este, em que as fotografias se relacionam sem ser pelo seu assunto, surgiu o conceito de grupos de semelhança de fotografias, que agrupam fotografias que são efetivamente semelhantes de conteúdo.

A criação de grupos de semelhança de fotografias é um dos primeiros processamentos efetuados pelo servidor *photo | uniq*, uma vez que pela sua complexidade se tornaria demasiado pesado para o ambiente móvel. Esta operação consiste em aplicar algoritmos de classificação e análise de conteúdo sobre as fotografias de forma a obter um grau de semelhança entre as fotografias e consequente formação de um grupo de semelhança[12].

O principal objetivo do ecrã de grupos de semelhança (visível na Figura 3.11) é permitir ao utilizador identificar rapidamente todos os grupos de semelhança existentes para o álbum, de forma a que este possa escolher qual o grupo em que pretende começar o processo de remoção de duplicados.

O utilizador tem duas opções para iniciar o processo de remoção de duplicados:

- A seleção automática da "melhor" fotografia sugerida pelo servidor para cada um dos grupos de semelhança do álbum, através do toque no botão "*best photo*" ()[13];

[11] A implementação corrente no protótipo tem funcionalidade limitada, uma vez que o ecrã de listagem de todas as fotografias presentes no servidor não se encontra desenvolvido.

[12] A implementação corrente no protótipo tem funcionalidade limitada, sendo os grupos de semelhança gerados localmente (cada conjunto de fotografias selecionado ao mesmo tempo para envio, corresponde a um grupo de semelhança para o álbum selecionado).

[13] Na prática, esta opção serve para simplificar todo este processo de seleção da melhor fotografia, para os utilizadores que confiem na escolha gerada pelos algoritmos.

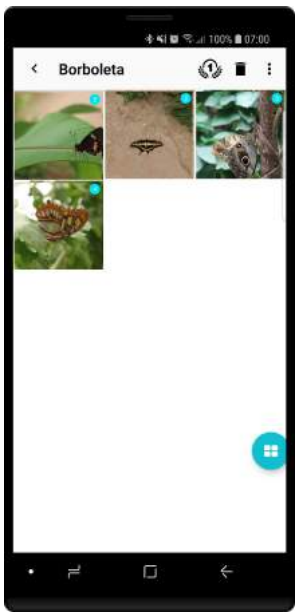


Figura 3.11: Ecrã de grupos de semelhança de um álbum.

- Ou a seleção manual da melhor fotografia para cada um dos grupos de semelhança, através do toque num dos grupos. Com o toque num dos grupos de semelhança o utilizador é redirecionado para a vista de pré-visualização para comparação de fotografias semelhantes (ver Secção 3.10).

Caso o utilizador opte pela primeira opção, a seleção automática de "melhor fotografia" para todos os grupos de semelhança do álbum, cada um dos grupos de semelhança é reduzido automaticamente a uma única fotografia, sendo as restantes fotografias movidas para o lixo do respetivo álbum (ver Secção 3.12). Para aceder ao ecrã de fotografias no lixo, basta que o utilizador toque no botão de lixo (🗑).

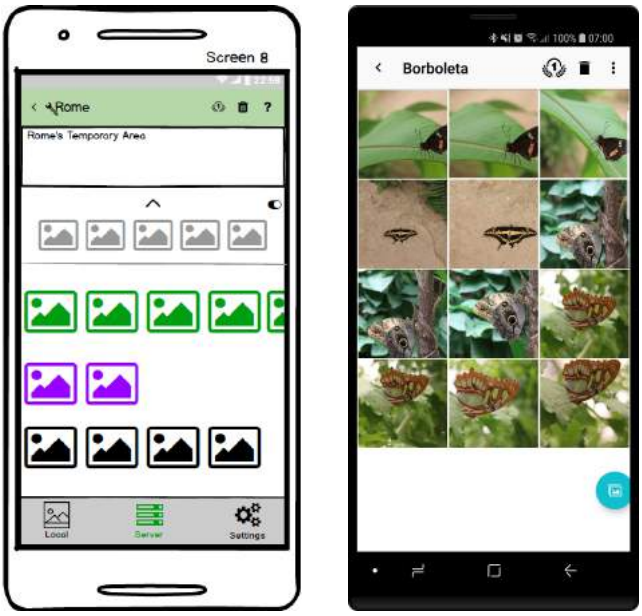
Para que o utilizador aceda ao ecrã de grupos de semelhança de um determinado álbum e dê início ao processo manual, basta que o utilizador toque num dos álbuns existentes na galeria *photo | uniq*. Assumindo que o utilizador tocaria num álbum com o nome "borboleta", o ecrã seguinte apresentado seria o da listagem de todos os grupos de semelhança do álbum "borboleta", como apresentado na Figura 3.11.

Por fim, os grupos de semelhança são apresentados de forma compacta, com apenas uma fotografia identificadora e com o número de fotografias existentes no grupo sobreposto num círculo azul.

3.9 Edição/Reorganização de Grupos de Semelhança de um Álbum

Voltando ao exemplo em que o utilizador faz uma viagem a Roma e cria um único álbum para todas as fotografias lá capturadas, embora os grupos de semelhança gerados pelo

nosso servidor utilizem algoritmos robustos, o utilizador pode discordar dos agrupamentos criados pelos algoritmos e, portanto, querer mover algumas das fotografias para outros grupos ou até mesmo criar um grupo de semelhança. Outra situação que pode acontecer é o utilizador verificar que uma determinada fotografia não devia estar no presente álbum, mas sim noutro. De forma a possibilitar todas estas operações, surgiu o ecrã visível na Figura 3.12.



(a) Versão maqueta.

(b) Versão implementada.

Figura 3.12: Ecrã de edição/reorganização de fotografias de um álbum.

O principal objetivo do ecrã de edição/reorganização de grupos de semelhança passa por fornecer ao utilizador uma visão total de todas as fotografias existentes num determinado álbum, com a particularidade de que estas são apresentadas segundo uma ordem decrescente, da melhor para a pior fotografia, de acordo com as qualidades técnicas das fotografias. Para além da automatização na organização das fotografias, é ainda oferecido um conjunto de operações sobre a área de trabalho: criar um grupo de semelhança; mover uma fotografia de um grupo de semelhança para outro; remover uma fotografia de um grupo de semelhança; e mover uma fotografia do álbum corrente para um outro álbum. Em complemento a estas operações, existem também os dois botões apresentados na Secção 3.8: o botão "*best photo*" (🕘) para seleção automática da melhor fotografia de cada um dos grupos de semelhança; e o botão de lixo (🗑) para aceder ao lixo do álbum corrente.

Ao ser feita uma comparação entre a versão concebida na maqueta (ver Figura 3.12a) e a implementada no protótipo (ver Figura 3.12b), são notórias algumas alterações, resultado de uma simplificação da interface. De forma a manter a interface consistente, nomeadamente na organização das fotografias e na uniformidade no método de interação com a aplicação, surgiram as seguintes alterações da maqueta para a implementação:

- Remoção da área temporária, que tinha como propósito fornecer uma área para a qual o utilizador podia arrastar as fotografias às quais pretendia alterar o grupo;
- Remoção da indicação das fotografias que estão em processo de envio para o álbum corrente (identificadas pela cor cinzenta, por baixo da área temporária na Figura 3.12a);
- A alteração da visualização dos grupos de fotografias semelhantes, passando a ser organização em grelha em vez de subdivisão em grupos de semelhança com *scroll* horizontal.

A funcionalidade da área temporária, uma área para a qual o utilizador podia arrastar fotografias, pode ser agora obtida através do arrastamento direto de uma fotografia para um dos grupos de semelhança existentes.

Ainda relativamente à estrutura implementada no ecrã de edição/reorganização de grupos de semelhança, os grupos de semelhança são listados todos de seguida, sem intervalos entre eles e as fotografias dos respetivos grupos são apresentadas por ordem da melhor para a pior fotografia, relativamente às características técnicas das mesmas.

Tal como para a grande parte dos ecrãs da aplicação, as operações disponibilizadas têm início com um toque longo sobre uma das fotografias ou grupo de fotografias. Ao ser efetuado o toque longo sobre uma fotografia, a fotografia sobressaí das outras e pode ser movida para qualquer outro grupo, sendo cada um dos outros grupos destacado à medida que o utilizador por lá arrasta a fotografia. Ainda durante o arrastamento da fotografia pelo ecrã, surgem dois botões flutuantes no fundo do ecrã: o botão para criação de um novo grupo de semelhança (⊕) e o botão para remoção da fotografia do seu grupo de semelhança (⊖). Sempre que exista alguma alteração ao nível dos grupos, por ordem de uma destas operações, aparece um novo botão na barra de topo, o botão de atualização (↻). Este botão tem como propósito forçar a reordenação das fotografias para cada um dos grupos de semelhança, através de uma chamada ao nosso servidor e consequente atualização da vista corrente. Optámos por colocar esta funcionalidade como um botão e não como algo automático após qualquer modificação nos grupos, para prevenir que o utilizador perdesse alguma alteração em andamento no momento da atualização[14].

Por fim, no caso em que o utilizador pretenda mover uma fotografia para um outro álbum, o utilizador apenas necessita de clicar no botão de mais opções (⋮) e aparecerá um novo *popup*. No novo *popup* apresentado, uma das opções será "mover para álbum". Assim que o utilizador clique no botão de "mover para álbum", surgirá uma janela com a função de browser, listando todos os álbuns que o utilizador criou na galeria *photo | uniq*. Após escolha do álbum para onde mover a fotografia, a fotografia é removida do grupo de semelhança e consequentemente do álbum em que está, e é movida para o novo álbum. Quando uma fotografia é movida de um álbum para outro, existe uma atualização para

[14] A implementação corrente no protótipo tem funcionalidade limitada, uma vez que a funcionalidade de atualização não se encontra implementada.

ambos os álbuns, de forma a remover a fotografia do álbum anterior e a colocá-la no grupo de semelhança correto (seja um grupo de semelhança novo ou um já existente). Sendo o principal objetivo da aplicação *photo* | *uniq* ajudar o utilizador a remover os duplicados de fotografias, optou-se por não permitir que o utilizador efetuasse cópias de fotografias. Ou seja, o utilizador pode somente: mover uma fotografia entre grupos de semelhança ou álbuns da galeria *photo* | *uniq*; ou então eliminar definitivamente uma fotografia da galeria *photo* | *uniq*.

3.10 Pré-Visualização para Comparação de Fotografias Semelhantes

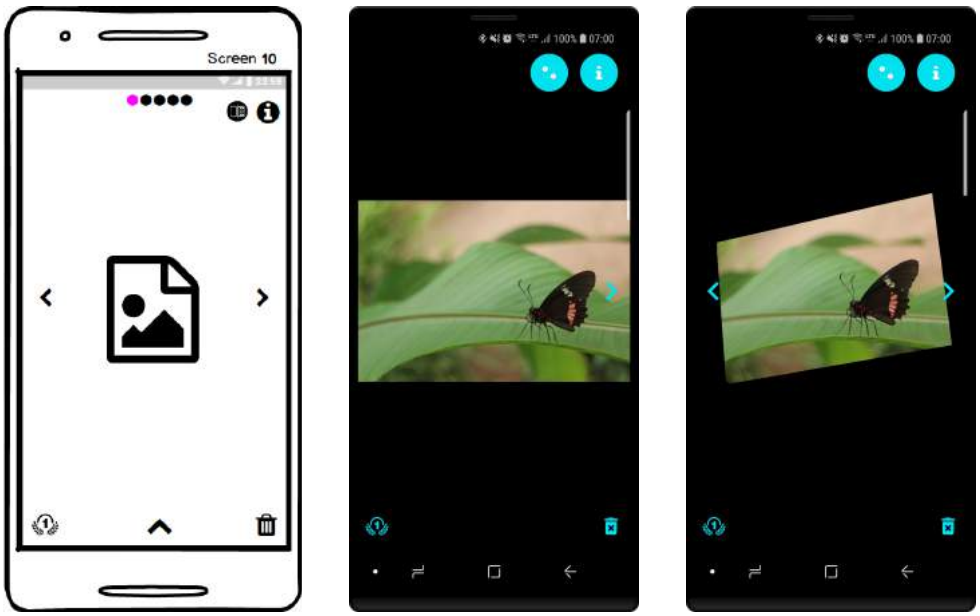
Por omissão, as fotografias num grupo de semelhança estão ordenadas por ordem decrescente da sua qualidade técnica, tal como avaliadas pela aplicação *photo* | *uniq*. No entanto, para atingir o objetivo de remoção de duplicados e ficar com apenas um bom exemplar é necessário passar pelo processo de confirmação da ordem proposta por comparação das fotografias duas-a-duas.

A pré-visualização para comparação de fotografias, cujo ecrã está visível na Figura 3.13, apresenta as fotografias do grupo de semelhança, em sequência, seguindo a ordenação proposta da melhor para a pior fotografia. A pré-visualização para comparação de fotografias tem como principais requisitos: apresentar as fotografias no seu formato original; permitir a comparação das fotografias de cada grupo de semelhança em função de uma fotografia pivô (também pertencente ao grupo de semelhança); e ajudar a escolher a melhor fotografia do grupo de semelhança (descartando todas as demais).

No ecrã de pré-visualização para comparação de fotografias, as fotografias são apresentadas no seu tamanho/formato original e são permitidos os convencionais gestos de *zoom* e navegação pelo ecrã:

- *Zoom in*, disponível através dos seguintes gestos:
  - Efetuar duplo toque com um dedo na área de interesse;
  - Ou o colocar de dois dedos próximos e afastar;
- *Zoom out*, através de:
  - De duplo toque com um dedo em qualquer área[15];
  - Da colocação de dois dedos afastados e consecutiva aproximação;
- Alteração da área de foco, com o arrastar de um dedo pelo ecrã[15];
- Navegação para a fotografia anterior ou seguinte, com o toque nas setas para a esquerda (‹) e para a direita (›), respetivamente.

[15]Gesto somente disponível quando a fotografia está com algum *zoom* aplicado, ou seja, quando existem zonas da fotografia que não aparecem no ecrã (por estarmos perante *zoom*).



(a) Versão maqueta. (b) Versão implementada. (c) Versão implementada.

Figura 3.13: Ecrã de pré-visualização para comparação de fotografias semelhantes de um álbum.

A transição entre as fotografias nos ecrãs de pré-visualização simples e de pré-visualização para comparação de fotografias difere. No caso da navegação no ecrã de pré-visualização simples de fotografias, quando é feito um deslizar do dedo da esquerda para a direita, existe um arrastar da fotografia atual nesta mesma direção, trazendo a nova fotografia para o ecrã ao mesmo tempo que a atual fotografia desaparece. Por outro lado, quando o utilizador quer comparar as fotografias no ecrã de pré-visualização para comparação, esta animação torna-se não só desnecessária como nefasta, pois dificulta a perceção das diferenças entre as fotografias. Assim, optámos por efetuar uma transição instantânea entre as fotografias, permitindo assim ao utilizador aperceber-se mais facilmente das diferenças entre as fotografias.

No momento em que o utilizador é redirecionado para o ecrã de pré-visualização para comparação de fotografias, existe um conceito de fotografia pivô associado a uma das fotografias do grupo de semelhança. A fotografia pivô é a fotografia com a qual todas as outras fotografias semelhantes se vão relacionar. Quando o ecrã de pré-visualização abre, a fotografia pivô é a fotografia apresentada mais à esquerda na listagem de fotografias semelhantes, ou seja, é a primeira fotografia da listagem e a primeira fotografia a ser apresentada ao utilizador. Uma fotografia pode ser definida como pivô de duas formas: através de um toque sobre uma fotografia no ecrã de edição/reorganização de grupos de semelhança; ou através de um toque sobre um grupo de semelhança no ecrã de grupos de semelhança. No caso em que o utilizador efetuou um toque sobre um grupo de semelhança, a fotografia pivô será a primeira fotografia segundo a proposta de ordenação

sugerida pela aplicação *photo | uniq*. Sempre que uma das fotografias é definida como a pivô, as fotografias são correlacionadas e são geradas as matrizes de homografia entre as fotografias (ver Secção 2.3), de modo a mapear as demais fotografias semelhantes em função da fotografia pivô. Nas Figuras 3.13b e 3.13c é possível ver um exemplo de uma fotografia de uma borboleta subordinada (Figura 3.13c) em função de uma outra fotografia da mesma borboleta (Figura 3.13b). Neste exemplo, o mapeamento da segunda fotografia em função da primeira, correspondeu maioritariamente a uma rotação na segunda fotografia.

O processo de escolha da melhor fotografia identificadora de um grupo de semelhança pode ser visto como um processo iterativo por eliminação sucessiva. Este processo iterativo consiste na visualização sequencial das fotografias semelhantes, subordinadas em função da fotografia pivô, podendo cada uma das fotografias ser assinalada como a melhor fotografia do grupo de semelhança (sendo "promovida" a pivô) ou como uma fotografia a eliminar do grupo de semelhança. Para assinalar as fotografias, o utilizador tem à sua disposição dois botões:

- O botão de "*best photo*" (), que assinala a presente fotografia como a "melhor" fotografia do seu conjunto de fotografias semelhantes. Quando o utilizador seleciona esta opção, a ronda termina e as restantes fotografias do grupo de semelhança são descontadas, reduzindo o grupo de semelhança à fotografia selecionada.

- O botão de lixo (), que marca a presente fotografia como "má", movendo-a do grupo de semelhança para o lixo do álbum corrente e, consequentemente, passando para a próxima fotografia na ronda eliminatória.

No decorrer da ronda eliminatória de fotografias semelhantes, o utilizador pode querer comparar um determinado detalhe da fotografia pivô nas restantes fotografias semelhantes. Para evitar que o utilizador tenha de navegar para trás e para a frente para conseguir comparar o detalhe entre as fotografias, foi introduzido um outro modo de comparação de fotografias semelhantes: a comparação de pares de fotografias semelhantes (ver Secção 3.11). O utilizador pode aceder ao ecrã de comparação de pares de fotografias semelhantes através de um toque no botão flutuante de fragmentação de ecrã ().

Por fim, tal como para a pré-visualização de fotografias é possível visualizar os metadados da fotografia correntemente visualizada através do botão flutuante .

3.11 Comparação de Pares de Fotografias Semelhantes

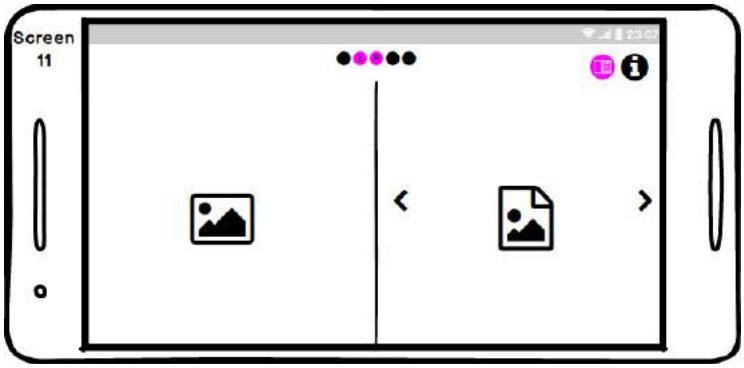
No processo de remoção de fotografias duplicadas, a comparação de pares de fotografias é o passo que permite ao utilizador identificar, através das qualidades técnicas das fotografias, qual é de facto a melhor fotografia dentro de um grupo de fotografias semelhantes.

A comparação de pares de fotografias, implementada no ecrã visível na Figura 3.14, é a nossa proposta de abordagem para esta questão da comparação de fotografias num

dispositivo móvel, o qual pode ter um ecrã bastante reduzido. A comparação de pares de fotografias tem como principais objetivos:

- Garantir que o utilizador consegue ter as condições necessárias para avaliar as características de cada uma das fotografias e de as comparar com as suas demais fotografias semelhantes;

- Fornecer um mecanismo de seleção da melhor fotografia de um grupo de semelhança, de acordo com os gostos do utilizador, facilitando a eliminação das restantes.



(a) Versão maqueta.

(b) Versão implementada.

Figura 3.14: Ecrã fragmentado de comparação de pares de fotografias de um grupo de semelhança de um álbum.

Quando o utilizador é redirecionado para o ecrã de comparação de pares de fotografias, são apresentadas duas fotografias no ecrã: a fotografia pivô (a fotografia que se encontra no lado esquerdo do ecrã); e a fotografia a ser subordinada em função da fotografia pivô (a fotografia que se encontra no lado direito do ecrã). Neste modo de comparação de duas fotografias lado a lado, o utilizador pode promover a fotografia subordinada a fotografia pivô através de um toque no botão flutuante assinalado com duas setas opostas ( ). A primeira fotografia pivô apresentada é a fotografia que estava a ser visualizada no

momento em que o utilizador clicou no botão flutuante de fragmentação (), no ecrã de pré-visualização para comparação de fotografias semelhantes.

Similarmente ao ecrã de pré-visualização para comparação de fotografias, o processo de escolha da melhor fotografia do grupo de semelhança pode também ser feito como uma ronda eliminatória. Contudo, para simplificar a interface apresentada no ecrã de comparação de pares de fotografias, apenas a fotografia pivô tem o botão de *"best photo"* () e apenas a fotografia subordinada em função da fotografia pivô tem o botão de lixo (). Em adição às semelhanças entre estes ecrãs, são também disponibilizados os convencionais gestos de *zoom* — sobre cada uma das fotografias — e navegação entre as fotografias:

- *Zoom in*, disponível através dos seguintes gestos:
  - Efetuar duplo toque com um dedo na área de interesse;
  - Ou o colocar de dois dedos próximos e afastar;
- *Zoom out*, disponível através de dois gestos:
  - De duplo toque com um dedo em qualquer área;
  - Da colocação de dois dedos afastados e consecutiva aproximação;
- Alteração da área de foco, com o arrastar de um dedo pelo ecrã[16];
- Navegação para a fotografia subordinada anterior ou seguinte, com o toque nas setas para a esquerda () e para a direita (), respetivamente.

Uma característica inovadora da aplicação *photo | uniq* nesta comparação de pares de fotografias é o *zoom* e alteração da área de foco sincronizados. Quando o utilizador efetua *zoom* numa das duas fotografias visualizadas, o *zoom* é aplicado não só nessa fotografia como também na outra. O mesmo se aplica à alteração da área de foco. Mas, mais importante ainda do que o *zoom* ser efetuado sobre as duas fotografias em simultâneo, é o facto de o *zoom* ser efetuado sobre o mesmo motivo nas duas fotografias e não sobre a mesma área. Todo este processo de *zoom* sobre o mesmo motivo em ambas as fotografias é possível graças ao mapeamento de uma fotografia (a fotografia subordinada) em função de uma outra fotografia (a fotografia pivô), tirando partido das matrizes de homografia. De forma a ilustrar o conceito de *zoom* sobre o mesmo motivo e *zoom* sobre a mesma área, na Figura 3.15 é possível encontrar duas fotografias semelhantes, nos dois modos de *zoom* mencionados acima.

No primeiro modo de *zoom*, visível na Figura 3.15a, a área de *zoom* para a fotografia da esquerda é a suspensão da roda dianteira da bicicleta e a área de *zoom* para a fotografia da direita é o quadro da bicicleta. Ou seja, área de *zoom* para ambas fotografias é a mesma. Já no caso do segundo modo de *zoom*, visível na Figura 3.15b, tanto a fotografia

[16]Gesto somente disponível quando a fotografia está com algum *zoom* aplicado, ou seja, quando existem zonas da fotografia que não aparecem no ecrã (por estarmos perante *zoom*).

(a) *Zoom* sobre a mesma área das fotografias.

(b) *Zoom* sobre o mesmo motivo das fotografias.

Figura 3.15: Modos de *zoom* sobre pares de fotografias.

da esquerda como a da direita têm como área de foco a suspensão da roda dianteira da bicicleta, resultado da transformação do ponto (x,y) no ponto (h(x),h(y)) usando a matriz de homografia. Neste caso, a área de *zoom* para ambas as fotografias é diferente, mas o motivo é o mesmo (a suspensão da roda dianteira da bicicleta).

3.12 Fotografias no Lixo de um determinado Álbum

Para reduzir o impacto de possíveis enganos por parte do utilizador na eliminação de fotografias na galeria *photo* | *uniq*, foi implementada uma funcionalidade de *backup* de todas as fotografias eliminadas no âmbito de cada um dos álbuns da galeria *photo* | *uniq*. Enquanto que na galeria local existe um único caixote de lixo para as fotografias (eliminadas no âmbito da galeria local), na galeria *photo* | *uniq* cada álbum tem o seu caixote de lixo.

O ecrã apresentado na Figura 3.16[17] tem como propósito listar todas as fotografias eliminadas de um determinado álbum da galeria *photo | uniq*, de modo a que o utilizador as possa restaurar para o respetivo álbum ou eliminar definitivamente da galeria do telemóvel. Caso o utilizador opte por eliminar definitivamente as fotografias da galeria do telemóvel, estas serão também automaticamente eliminadas da galeria *photo | uniq*.

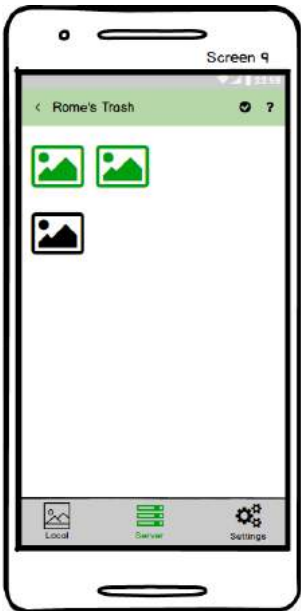


Figura 3.16: Ecrã de fotografias no lixo de um determinado álbum.

Tal como para o ecrã de fotografias no lixo da galeria local, o ecrã de fotografias no lixo de um determinado álbum, tem a sua existência condicionada pela definição de eliminação automática no ecrã de definições (ver Secção 3.14).

Por fim, para que o utilizador possa executar as operações disponibilizadas pelo ecrã, basta que o utilizador pressione durante alguns segundos uma ou várias fotografias. Após a ativação do modo de seleção, o utilizador pode então escolher restaurar ou eliminar definitivamente as fotografias selecionadas.

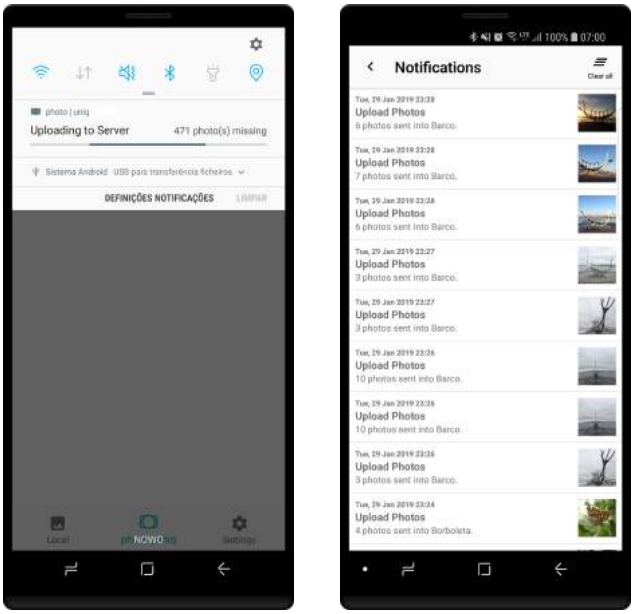
3.13 Sistema de Notificações

As notificações podem ser um mecanismo prático e eficaz para enviar pequenas mensagens aos utilizadores que tenham a aplicação instalada. As notificações podem ser geradas internamente pela própria aplicação ou serem geradas por uma fonte externa à aplicação (por exemplo, um servidor alojado na *cloud*). Estas mensagens podem ser meramente informativas ou então podem despoletar ações nas aplicações.

Na versão corrente da aplicação *photo | uniq*, é a própria aplicação quem envia notificações para o utilizador. Uma vez que o envio de fotografias para a galeria *photo | uniq* tira

[17] A implementação corrente no protótipo tem funcionalidade limitada, uma vez que este ecrã não se encontra implementado.

partido de um serviço remoto, foi decidido que deveria existir algum mecanismo que informasse o utilizador sobre o estado do processo de envio das fotografias. Assim sendo, sempre que é iniciado um novo pedido de envio de fotografias para o servidor, é gerada uma notificação com o número de fotografias a enviar. Com o decorrer dos envios, o texto da notificação é atualizado (ver Figura 3.17a).



(a) Versão implementada.

(b) Versão implementada.

Figura 3.17: Ecrã de notificações da aplicação.

No ecrã apresentado na Figura 3.17, é visível um histórico de todas as notificações que a aplicação recebeu. Este ecrã é acessível através de qualquer um dos três acessos principais da aplicação (Galeria Local, Galeria *photo | uniq* e Definições), bastando apenas que o utilizador clique no botão assinalado com um sino (🔔), disponibilizado no menu superior.

3.14 Definições da Aplicação

Através do ecrã de definições o utilizador pode configurar uma série de parâmetros da aplicação de acordo com as suas preferências. Por exemplo, a ativação (ou desativação) do serviço de notificações da aplicação.

No caso da aplicação *photo | uniq*, é disponibilizado um ecrã de definições (visível na Figura 3.18), para que o utilizador:

- Decida se pretende receber ou não notificações relacionadas com a aplicação;
- Decida se as fotografias eliminadas devem ser de imediato eliminadas ou movidas para um caixote do lixo temporário;

- Altere o idioma da aplicação[18];
- Escolha qual o tamanho da fonte a ser utilizada ao longo da aplicação;
- Aceda a um pequeno descritivo da aplicação;
- Possa partilhar a aplicação *photo | uniq*com os seus amigos, através de um link;
- Possa fazer *logout*;
- Possa restaurar para as definições padrão.

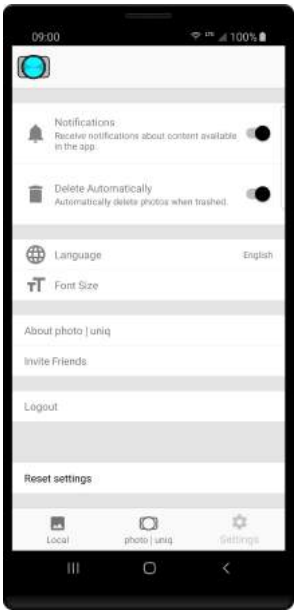


Figura 3.18: Ecrã de definições da aplicação.

[18]De momento, a aplicação apenas está disponível em inglês.

Capítulo 4

Desenvolvimento da Aplicação (*photo | uniq*)

Neste capítulo apresenta-se informação detalhada relevante para a especificação e a implementação da aplicação *photo | uniq*, nomeadamente: a arquitetura da aplicação; o modelo de dados; a interação aplicação/servidor; os algoritmos que suportam a comparação de fotografias lado a lado; e ainda as bibliotecas externas utilizadas.

4.1 Arquitetura

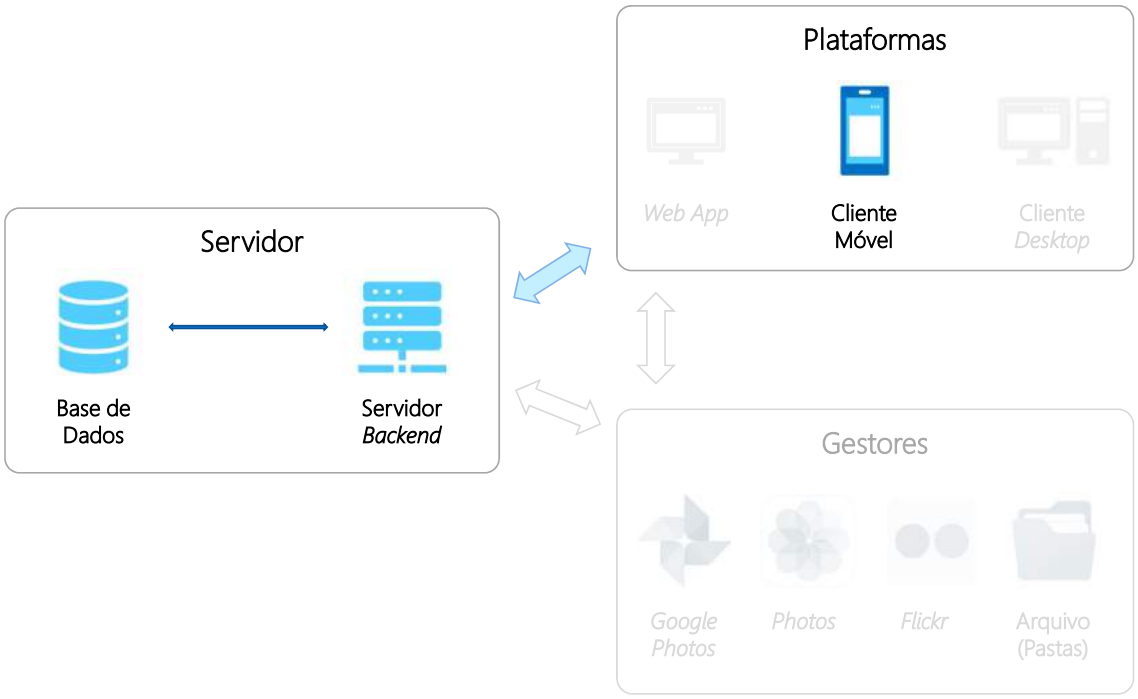


Figura 4.1: Componentes implementados da arquitetura proposta.

No desenvolvimento da aplicação *photo | uniq* seguiu-se a arquitetura apresentada na Figura 1.5, na página 4, mas focando numa solução específica para a plataforma móvel Android e respetivos serviços necessários por parte do servidor (*backend*). Na Figura 4.1 encontram-se coloridos os componentes relevantes e efetivamente implementados no protótipo desenvolvido. [1]

A aplicação é composta por quatro componentes diferentes:

- **A base de dados do telemóvel**, que contém todos os ficheiros media do dispositivo;
- **A base de dados da aplicação**, que contém ligações para um subconjunto de fotografias da base de dados do telemóvel;
- **O repositório de informação volátil**, que mantém em memória todos os dados necessários para o (rápido) funcionamento da aplicação;
- **As vistas da aplicação**, que correspondem a todos os ecrãs disponibilizados pela aplicação.

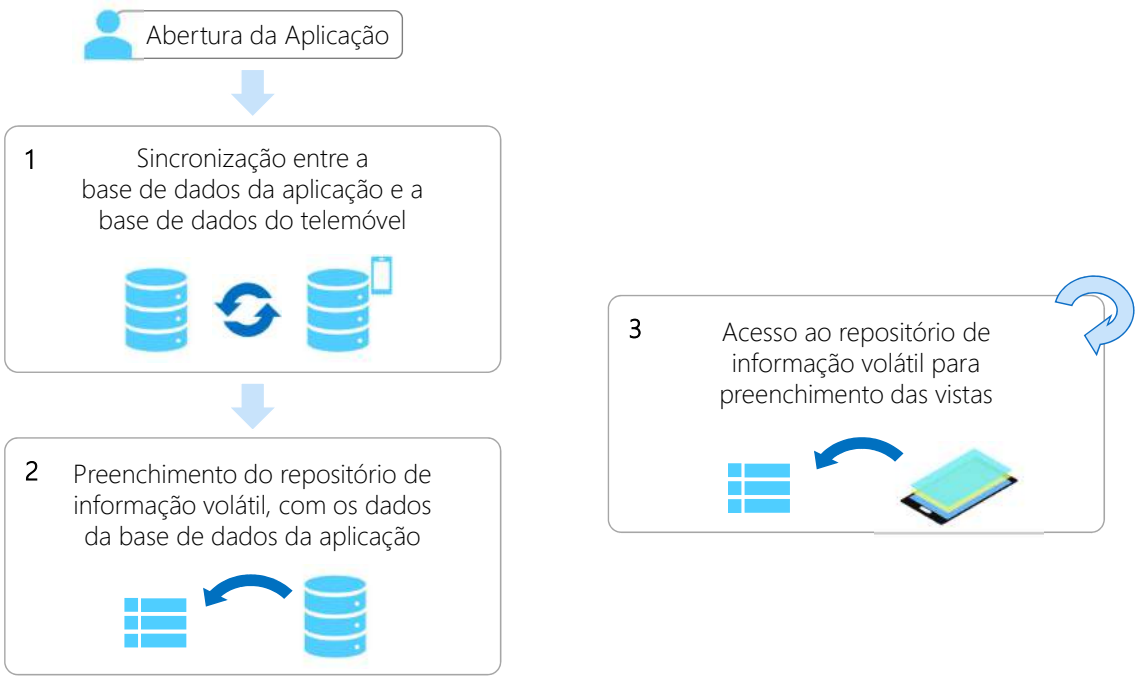


Figura 4.2: *Workflow* do funcionamento da aplicação.

Na Figura 4.2 é apresentado o *workflow* do funcionamento da aplicação. Quando a aplicação inicia, existe uma sincronização entre a base de dados do telemóvel e a base de dados da aplicação. Este é o passo responsável pela atualização das fotografias apresentadas na vista de galeria local, garantindo que todas as fotografias tiradas após a última

[1] Embora a componente do servidor esteja colorida, no âmbito desta dissertação apenas está contemplada a componente do cliente móvel.

utilização da aplicação são incorporadas na vista local e que todas as apagadas são também removidas. No segundo passo os dados são estruturados e organizados numa classe dedicada, de forma a serem acessíveis rápida e facilmente por todas as vistas da aplicação. Como último passo temos o acesso, por parte das vistas, à informação volátil. Este último passo repete-se tantas vezes quantas as que uma nova vista seja aberta. Se a aplicação tiver as credenciais do utilizador armazenadas, o *workflow* inicial passa a ter mais um passo após a sincronização da base de dados: o pedido de sincronização de estado entre a base de dados local e a base de dados do servidor *photo | uniq*, garantindo que a aplicação fica com os dados atualizados com o servidor *photo | uniq*.

Por fim, para garantir que o repositório de informação volátil é carregado com os dados corretos/atualizados, cada ação da aplicação que afeta os dados é propagada não só para o repositório de dados voláteis como também para a base de dados da aplicação. No entanto, o repositório de informação volátil só é totalmente recarregado com os dados da base de dados da aplicação quando a aplicação é aberta de raiz (*cold start* [42]), ou seja, quando o utilizador tenta abrir a aplicação e o sistema operativo do dispositivo ainda não tinha criado o processo dessa aplicação.

Listagem 4.1: Método responsável pela sincronização das bases de dados e pela inicialização do repositório volátil.

```
1  private fun sync() {
2    var id: Int
3    val photos = mutableListOf<PhotoItem>()
4    val photosIDs = mutableListOf<Int>()
5
6    val maxID = App.database.maxID
7
8    val uri = MediaStore.Images.Media.EXTERNAL_CONTENT_URI
9    val projection = arrayOf(MediaStore.Images.Media.DATA, MediaStore.
       ↪ Images.Media.BUCKET_DISPLAY_NAME, MediaStore.Images.Media.
       ↪ DATE_TAKEN, MediaStore.Images.Media._ID)
10   val selection = MediaStore.Images.Media.MIME_TYPE + "␣not␣like␣?"
11   val mimeType = MimeTypeMap.getSingleton().getMimeTypeFromExtension("gif
       ↪ ")
12   val selectionArgsPdf = arrayOf(mimeType)
13
14   val cursor = contentResolver.query(uri, projection, selection,
       ↪ selectionArgsPdf, null)
15
16   cursor?.use {
17     while (cursor.moveToNext()) {
```

```
18      id = cursor.getInt(cursor.getColumnIndexOrThrow(MediaStore.Images.
            ↪ Media._ID))
19
20      if (id > maxID) {
21        val photoItem = PhotoItem(id)
22        photoItem.path = cursor.getString(cursor.getColumnIndexOrThrow(
              ↪ MediaStore.Images.Media.DATA))
23        photoItem.dateTime = cursor.getLong(cursor.getColumnIndexOrThrow(
              ↪ MediaStore.Images.Media.DATE_TAKEN))
24        photoItem.folderName = cursor.getString(cursor.
              ↪ getColumnIndexOrThrow(MediaStore.Images.Media.
              ↪ BUCKET_DISPLAY_NAME))
25
26        photos.add(photoItem)
27      }
28
29      photosIDs.add(id)
30    }
31  }
32
33  App.database.updateDatabase(photos, photosIDs, this)
34
35  val allPhotos = App.database.allPhotos
36
37  App.instance.setVolatilRepo(allPhotos)
38
39 }
```

Na Listagem 4.1 é apresentado o método responsável pela sincronização das bases de dados (do telemóvel e da aplicação) e pela inicialização do repositório volátil. Este método consiste na seguinte sequência de passos:

1. Obter o ID (da fotografia) com valor mais alto na base de dados da aplicação (linha 6);

2. Definir a consulta a efetuar à base de dados do dispositivo (linhas 8 a 12);

3. Executar a consulta por imagens (linha 14);

4. Iterar o conjunto de fotografias retornado pela consulta, construindo dois conjuntos (linhas 16 a 31):

- Fotografias Novas, fotografias a adicionar à base de dados da aplicação, sendo este conjunto definido por todas as fotografias que tenham um ID superior ao retornado no passo 1 (linhas 20 a 27);
- Identificadores Únicos, conjunto de todos os identificadores únicos das fotografias existentes na base de dados do telemóvel (linha 29).

5. Atualizar a base de dados da aplicação, adicionando as novas fotografias e eliminando todas as fotografias que não constem no conjunto de identificadores únicos (linha 33);

6. Obter o conjunto (atualizado) de fotografias presentes na base de dados da aplicação (linha 35);

7. Preencher o repositório de informação volátil, utilizando o conjunto de fotografias obtido no passo 6 (linha 37).

Com esta abordagem todas as novas fotografias na galeria do dispositivo são inseridas na base de dados da aplicação e todas as fotografias apagadas da galeria do dispositivo são também eliminadas da aplicação. Assim, o único caso que ficou por cobrir foi o da atualização do caminho de uma fotografia no caso de, por exemplo, o utilizador mover uma fotografia de uma pasta para outra. Para casos como este, a aplicação apenas fará a atualização *on-demand*, ou seja, a aplicação apenas tentará atualizar o caminho da fotografia quando a tentar carregar numa das vistas da aplicação e o carregamento falhar. Caso o carregamento falhe, a aplicação faz uma pesquisa pelo ID da fotografia em questão à base de dados do dispositivo e, caso a pesquisa suceda, a aplicação atualiza a sua base de dados com o novo caminho da fotografia. Caso contrário, a aplicação elimina a fotografia da sua base de dados[2].

4.2 Modelo de Dados

O modelo de dados é apresentado na Figura 4.3. Nesta Figura é possível identificar 5 classes: a *PhotoItem*, o *SimilarityGroup*, o *Album*, a *Homography* e o *UploadJob*.

A classe *PhotoItem* representa uma fotografia, podendo esta ser uma fotografia da galeria local do dispositivo móvel ou da galeria *photo | uniq*. Os seus atributos podem ser agrupados em três grandes grupos:

- Atributos básicos, tais como o nome da fotografia, o caminho para a sua localização no dispositivo[3], o nome da pasta à qual a fotografia pertence, a sua data de captura, entre outros.

[2]Como considerámos que estes casos de atualização teriam uma probabilidade bastante baixa para acontecer, não investimos muito mais tempo a tentar procurar uma solução melhor.

[3]No caso das fotografias existentes apenas na galeria *photo | uniq* (não existentes na galeria local), o caminho é o *url* de *download* da respetiva fotografia.

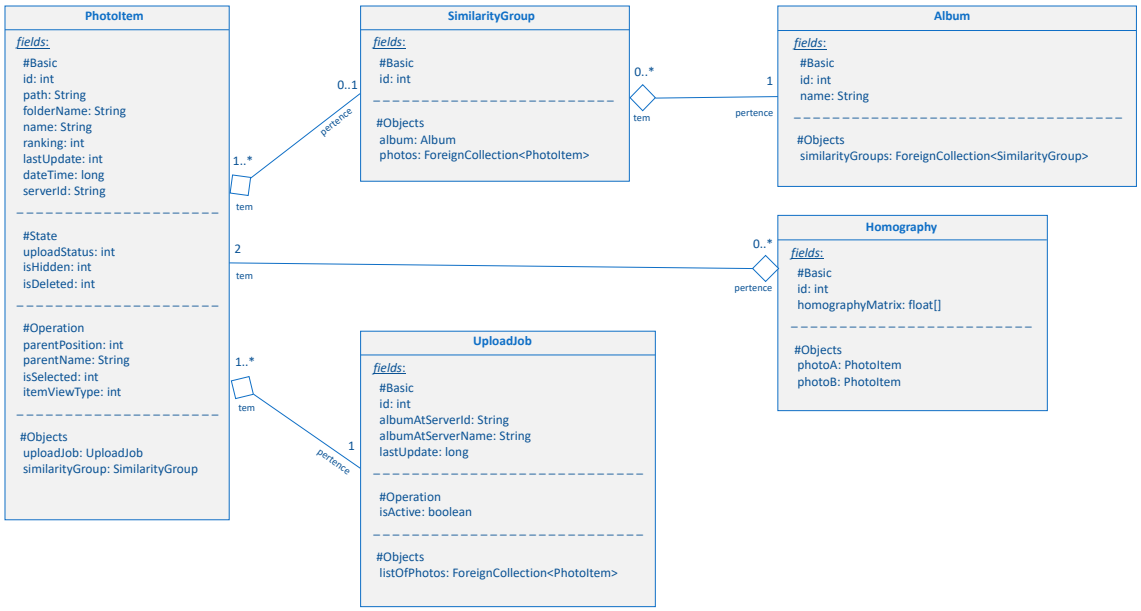


Figura 4.3: Modelo de dados.

- Atributos de estado, como é o caso de *isHidden*, *isDeleted* e *uploadStatus*. Estes atributos de estado têm como objetivo possibilitar a filtragem das fotografias, por exemplo, se uma fotografia estiver identificada como *isHidden* significa que foi anteriormente escondida pelo utilizador e, por isso, apenas aparecerá no ecrã de fotografias ignoradas.

- Atributos de operação, tais como o *parentPosition*, o *parentName*, o *isSelected* e o *itemViewType*. Este último tipo de atributos refere-se a atributos utilizados para funcionalidades específicas da aplicação, como por exemplo a seleção de fotografias.

Para além dos atributos acima listados, a classe *PhotoItem* contêm ainda duas referências para objetos: o *uploadJob*, que corresponde ao *job* através do qual a fotografia foi enviada para o servidor; e o *similarityGroup*, que corresponde ao grupo de semelhança em que a fotografia está inserida[4].

Enquanto que a classe *SimilarityGroup* representa um grupo de semelhança e tem como atributos uma coleção de fotografias e o identificador único do respetivo grupo de semelhança, a classe *UploadJob* representa um *job* de envio de fotografias em segundo plano para a galeria *photo | uniq*. A classe *UploadJob* tem como atributos a coleção de fotografias a enviar, o nome e identificador único do álbum para o qual as fotografias vão ser enviadas[5], uma variável de estado para indicar se tarefa ainda se encontra ativa (caso

[4]Para simplificar a lógica da implementação de grupos de semelhança, todas as fotografias do servidor têm um grupo de semelhança. No caso de fotografias singulares (fotografias que não têm fotografias consideradas semelhantes), o grupo de semelhança corresponde à própria fotografia.

[5]Caso o envio seja para um álbum novo, existe uma primeira chamada ao serviço de criação de um álbum, de forma a ser obtido o novo id para o qual enviar as fotografias.

ainda se encontrem fotografias na fila de espera de envios), e o identificador único do *job* em causa.

A classe *Album* representa um álbum e tem como atributos o seu nome e identificador único, para além de também um conjunto de grupos de semelhança.

Por fim, a classe *Homography* representa uma matriz de homografia entre duas fotografias *photoA* e *photoB*. Para além das duas referências de *PhotoItem*, a classe *Homography* tem como atributos o seu identificador único e a matriz de homografia entre as duas fotografias.

4.3 Interação Aplicação/Servidor

O funcionamento da aplicação *photo | uniq* faz uso de alguns processos com custos de processamento demasiado elevados para o ambiente móvel, pelo que estes foram transferidos para um servidor externo. Para além de realizar processamentos computacionalmente mais exigentes, o servidor é também utilizado para armazenar não só todas as fotografias, grupos de semelhança e álbuns, como também meta-dados e matrizes de homografia para uso na aplicação.

Todo o suporte aplicacional para a interação aplicação/servidor está construído em *Representational State Transfer* (REST), seguindo as rotinas e funcionalidades definidas na *Application Programming Interface* (API) disponibilizada pelo servidor *photo | uniq*.

Como referido acima, a aplicação *photo | uniq* tem diversos momentos de interação com o servidor, podendo eles ser categorizados em cinco tipos:

- Autenticação;
- Sincronização Inicial;
- *Upload*/*Download*/Remoção de Fotografias;
- Gestão de Álbuns e de Grupos de Semelhança;
- Sincronização de Operações Multi-Dispositivo[6].

Nas próximas secções serão listados todos os serviços fornecidos pela API do servidor *photo | uniq*. Os serviços serão apresentados em tabelas sendo mencionado para cada um deles o método, o *endpoint*, os parâmetros e o resultado.

4.3.1 Autenticação

A autenticação na aplicação *photo | uniq* é obrigatória a partir do momento em que o utilizador quer tirar partido das funcionalidades fornecidas pela galeria *photo | uniq*. O processo de autenticação é conseguido através de um dos dois serviços existentes na API:

[6] A sincronização de operações multi-dispositivo não se encontra implementada no protótipo da aplicação, mas está no planeamento para trabalho futuro.

um de registo (serviço com ID 4.1.2 na Tabela 4.1) e um de *login* (serviço com ID 4.1.1 na Tabela 4.1) — usados quando um registo já foi efetuado anteriormente ou quando o *login* é efetuado através das redes sociais, respetivamente.

Tabela 4.1: Tabela descritiva dos serviços de autenticação.

ID	Tipo	Serviço	Parâmetros	Resposta
4.1.1	GET	/user/login	<*username*>; <password>;	id do utilizador no servidor
4.1.2	GET	/user/register	<nome>; <*username*>; <email>; <password>;	id do utilizador no servidor

4.3.2 Sincronização Inicial

Sempre que a aplicação *photo | uniq* é iniciada, é verificada a presença de sessão (*login*). Caso exista uma sessão iniciada, é chamado um serviço de sincronização entre o estado (local) da aplicação e o estado presente no servidor *photo | uniq* (serviço com ID 4.2.1 na Tabela 4.2).

Tabela 4.2: Tabela descritiva do serviço de sincronização.

ID	Tipo	Serviço	Parâmetros	Resposta
4.2.1	GET	/user/getSession	<id do utilizador>;	estado da sessão

4.3.3 Gestão de Fotografias

No decorrer da utilização da aplicação, mais particularmente dentro da galeria *photo | uniq*, existem diversas situações que requerem acesso ao servidor (através dos seus serviços). Situações estas que podem ser desencadeadas por ordem do utilizador ou por funcionalidades base da aplicação. Encontram-se, de seguida, enumerados alguns destes momentos (ver Tabela 4.3):

- Listagem de todas as fotografias na galeria local (serviço com ID 4.3.1);
- *Upload* de fotografias para a galeria *photo | uniq* (serviço com ID 4.3.2);
- *Download* de fotografias a partir da galeria *photo | uniq* (serviço com ID 4.3.3);
- Atualização do grupo de semelhança de uma fotografia (serviço com ID 4.3.4);
- Eliminação de fotografias da galeria *photo | uniq* (serviço com ID 4.3.5);

- Comparação de fotografias lado a lado (serviço com ID 4.3.6).

Tabela 4.3: Tabela descritiva dos serviços de manipulação de fotografias.

ID	Tipo	Serviço	Parâmetros	Resposta
4.3.1	GET	/photos	<id do utilizador>; <id do álbum>;	lista de fotos do álbum
4.3.2	POST	/photo	<id do utilizador>; <id do álbum>; <foto>;	id da foto no servidor
4.3.3	GET	/photo	<id do utilizador>; <id do álbum>; <id da foto no servidor>;	foto
4.3.4	PUT	/photo	<id do utilizador>; <id do álbum>; <foto atualizada>;	n.a.
4.3.5	DELETE	/photo	<id do utilizador>; <id do álbum>; <id da foto no servidor>;	n.a.
4.3.6	GET	/photos/matrix	<id da foto A no servidor>; <id da foto B no servidor>; <id do utilizador>; <id do álbum>;	matriz de homografia AB

4.3.4 Gestão de Álbuns e de Grupos de Semelhança

A aplicação *photo* | *uniq* permite criar álbuns de fotografias na galeria *photo* | *uniq* para que o utilizador consiga não só organizar as suas fotografias, como também identificar rápida e facilmente em que álbum estão determinadas fotografias. Apesar do utilizador apenas escolher um conjunto de fotografias e álbum para o qual enviar o respetivo conjunto, quando o servidor *photo* | *uniq* recebe as fotografias, existe um passo extra além da associação das fotografias ao álbum: a criação dos grupos de semelhança para o conjunto de fotografias recebido. Os grupos de semelhança gerados correspondem a subconjuntos (menores) de fotografias efetivamente semelhantes do conjunto inicial enviado pelo utilizador.

Na aplicação *photo* | *uniq*, para além de ser possível gerir as fotografias (por exemplo, escondendo ou enviando-as para a galeria *photo* | *uniq*), é também possível gerir os álbuns e grupos de semelhança através das seguintes operações (ver Tabela 4.4):

- Listagem de álbuns (serviço com ID 4.4.1);
- Criação de álbuns (serviço com ID 4.4.2);
- Listagem de grupos de semelhança de um álbum (serviço com ID 4.4.4);

- Criação de grupos de semelhança (serviço com ID 4.4.5);
- Eliminação de grupos de semelhança (serviço com ID 4.4.7);

Tabela 4.4: Tabela descritiva dos serviços de gestão de álbuns e de grupos de semelhança.

ID	Tipo	Serviço	Parâmetros	Resposta
4.4.1	GET	/albums	<id do utilizador>;	lista de álbuns
4.4.2	POST	/album	<id do utilizador>; <id do álbum>; <álbum>;	id do álbum
4.4.3	GET	/album	<id do utilizador>; <id do álbum>;	album
4.4.4	GET	/groups	<id do utilizador>; <id do álbum>;	lista de grupos de semelhança do álbum
4.4.5	POST	/group	<id do utilizador>; <id do álbum>;	id do grupo de semelhança
4.4.6	GET	/group	<id do utilizador>; <id do grupo de semelhança>;	grupo de semelhança
4.4.7	DELETE	/group	<id do utilizador>; <id do grupo de semelhança>;	n.a.

4.4 *Login*

O *login* permite identificar unicamente cada um dos utilizadores da aplicação *photo | uniq*, para além de ser também o ponto de partida para a utilização *cross platform* da aplicação *photo | uniq*. Sem esta funcionalidade o estado e dados da aplicação apenas existiriam no âmbito do dispositivo utilizado, impossibilitando assim o começo de uma sessão de eliminação de fotografias repetidas num dispositivo e término noutro.

Na aplicação *photo | uniq* é possível efetuar *login* de três maneiras: Customizada, com o Facebook ou com a Google.

O primeiro tipo de *login* apresentado, o *login* customizado, tem como requisito um registo na aplicação através do preenchimento de um formulário que tem como dados obrigatórios o email e a password. Para além destes dados obrigatórios, o utilizador pode ainda inserir o seu nome num campo disponibilizado para tal[7]. Assim que o utilizador preencha os campos obrigatórios e submeta o formulário, a aplicação chama o serviço de registo. No momento em que é obtida resposta do serviço, consoante a existência (ou não) do email no servidor, existem dois cenários possíveis:

[7] O formulário do ecrã registo é acessível através do ecrã de *login*.

- O email inserido ainda não existe na base de dados do servidor *photo | uniq*: Neste caso é enviado um email de confirmação de registo para o email inserido no formulário e, assim que seja feita a confirmação do email, o registo é dado como bem-sucedido e o utilizador pode começar a utilizar os dados previamente inseridos no formulário de registo para efetuar o *login*.

- O email já existe na base de dados do servidor *photo | uniq*: Neste caso, é apresentado um erro ao utilizador e este pode escolher alterar o email previamente inserido ou efetuar um pedido de recuperação de password.

Tanto o *login* com o Facebook como o *login* com a Google, necessitam da integração dos respetivos *SDKs* na aplicação. Após a importação dos *SDKs*, segue-se o registo do projeto (aplicação) nos respetivos portais: *Facebook for Developers* [43] e *Firebase* [44].

No processo de registo do projeto no *Facebook for Developers*, existe uma primeira pré-seleção da plataforma para o qual o *login* será integrado. De seguida, é necessário preencher alguns dados sobre a aplicação, nomeadamente o nome, a categoria e o *package name*, assim como o *upload* do ícone da aplicação e da política de privacidade. Assim que estas informações estejam inseridas, segue-se a conexão da aplicação *photo | uniq* com o projeto criado. Para fazer a conexão, para além da exigida importação do *SDK*, é necessário inserir no código relativo ao *SDK* dois valores fornecidos pelo projeto criado: o *App ID* e o *App Secret*.

Relativamente ao *Firebase*, antes da configuração dos dados da aplicação, é necessária a criação de um projeto. Este projeto difere do projeto do *Facebook for Developers*, uma vez que pode agregar diversas aplicações e não apenas uma. Assim que o projeto esteja criado segue-se a criação da aplicação, tendo novamente de ser escolhida a plataforma com a qual o *login* será integrado. Só após a escolha da plataforma é que aparecem os campos a preencher com os dados da aplicação (nome e *package name*). Tal como para o Facebook, são também gerados dois códigos a colocar no código: a "Chave de API da Web" e o "Código do aplicativo".

Por fim, registada a aplicação em ambos os portais — Google e Facebook —, só fica em falta a configuração do código na aplicação. A configuração é bastante simples para ambos os sistemas, sendo até fornecida uma documentação, passo-a-passo para a inserção do código necessário para cada um dos sistemas e mencionados os locais onde inserir os valores gerados anteriormente no registo da aplicação nos portais[8].

4.5 Serviços de Segundo Plano

A aplicação *photo | uniq* utiliza serviços em segundo plano para executar algumas tarefas, como é o caso do envio de fotografias para o servidor *photo | uniq* e atualização de algumas

[8] A documentação da Google e do Facebook estão disponíveis nos seguintes links [44, 45], respetivamente.

das vistas da aplicação. Estes serviços trazem uma grande vantagem à aplicação: a diminuição significativa do tempo de espera, por parte do utilizador, para utilizar a aplicação *photo | uniq*.

Quando o utilizador executa a operação de enviar um conjunto de fotografias para a galeria *photo | uniq* a aplicação cria um serviço — que fica a correr em segundo plano — com uma fila de espera de fotografias a enviar para o servidor *photo | uniq*. Este serviço permite não só que o utilizador continue a utilizar a aplicação como garante também que, se a aplicação for fechada, o envio de fotografias se mantém ativo.

Sempre que alguma vista da aplicação efetua um pedido ao servidor *photo | uniq*, o pedido é efetuado em segundo plano. Quando o serviço retorna resposta, a vista pela qual o pedido foi iniciado é notificada. A abertura da vista da galeria *photo | uniq* é um exemplo onde é sempre efetuado um pedido ao servidor em segundo plano: a verificação de atualizações para o conteúdo apresentado na vista (álbuns, grupos de semelhança, entre outros). No momento em que o utilizador clica no menu de galeria *photo | uniq*, a vista faz um pedido de atualização ao servidor. No momento que o serviço retorna a resposta, caso existam atualizações, a vista da galeria *photo | uniq* é atualizada, apresentando as novas alterações retornadas pelo serviço.

4.6 Comparação de Fotografias Lado a Lado

A comparação de fotografias lado a lado é a funcionalidade que permite ao utilizador identificar, através das qualidades técnicas das fotografias, qual é de facto a fotografia melhor qualificada entre um conjunto de fotografias semelhantes. A aplicação *photo | uniq* tem como grande foco nesta funcionalidade disponibilizar um mecanismo que faça a sincronização (automática) no mesmo motivo para duas fotografias visualizadas lado a lado (ver Figura 3.15, na página 50), para além do suporte aos convencionais gestos de *zoom* e *panning* (mencionados na Secção 3.11, na página 47) sobre ambas as fotografias.

O primeiro passo para a comparação de fotografias lado a lado é a fragmentação do ecrã (em modo *landscape*) em dois *containers* para uma apresentação simultânea de duas fotografias. Cada *container* tem a origem do seu referencial cartesiano no limite superior esquerdo e o *container* do lado esquerdo carregará a fotografia pivô e o *container* do lado direito a fotografia subordinada (ver Figura 4.4). Utilizámos as cores laranja e azul para identificar o *container* da fotografia pivô e o *container* da fotografia subordinada, respetivamente.

O segundo passo na comparação de fotografias lado a lado passa por utilizar transformações geométricas 2D para: fazer o carregamento das fotografias com o tamanho certo e devidamente centradas no *container*; e apresentar o mesmo motivo de foco, e não a mesma área de foco, em ambas as fotografias (pivô e subordinada).

Por último, segue-se o suporte aos gestos de *zoom* e *panning*, garantindo que é sempre focado/sincronizado o mesmo motivo em ambas as fotografias.

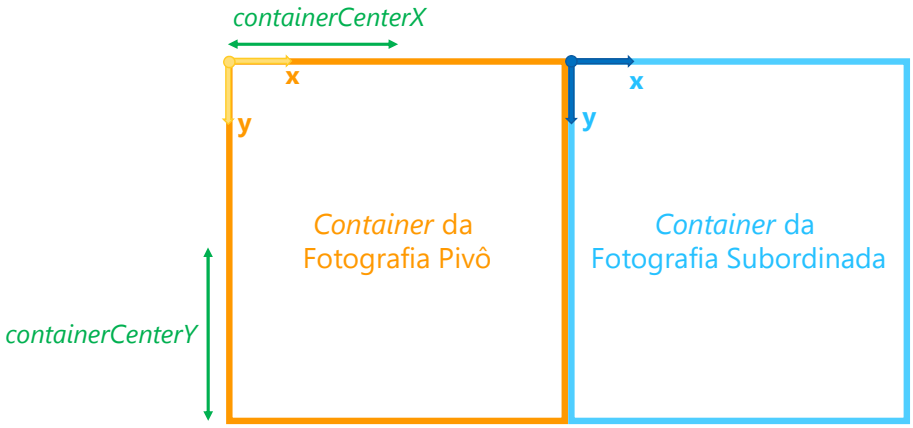


Figura 4.4: Fragmentação do ecrã em dois *containers*: Fotografia Pivô e Fotografia Subordinada (unidades em *pixels*).

O processo de comparação de fotografias pode dividir-se em duas fases onde as transformações são aplicadas: a Apresentação Inicial das Fotografias e a Interação (do utilizador) com as Fotografias. Nas próximas secções serão apresentadas todas as transformações aplicadas em cada uma destas fases, sendo algumas transformações diferentes para a fotografia pivô e para a fotografia subordinada. Seguem-se algumas notas a ter em conta nas próximas secções:

- Existem dois tipos de coordenadas ao efetuar as transformações[9]:
  - Coordenadas de ecrã, que correspondem às coordenadas compreendidas no *container* de cada fotografia;
  - Coordenadas de fotografia, que correspondem às coordenadas na resolução de cada fotografia.
- Na Listagem 4.2 é apresentado o método responsável por aplicar as transformações base sobre as fotografias. Este método assenta na seguinte sequência de transformações, sendo que os passos 1 e 2 trabalham com coordenadas de fotografia e o passo 3 com coordenadas de ecrã:
  1. Deslocar um ponto específico da fotografia, para o qual sabemos a sua localização pretendida no *container*, para a origem do referencial cartesiano correspondente (linha 5 do Listagem 4.2);
  2. Aplicar a escala pretendida sobre a fotografia (linha 6 do Listagem 4.2);
  3. Mover o ponto que se encontra na origem do referencial para o centro do *container* da fotografia (linha 7 do Listagem 4.2).
- As transformações são guiadas pelos valores de duas variáveis de estado definidas para cada uma das fotografias representadas nos *containers*:

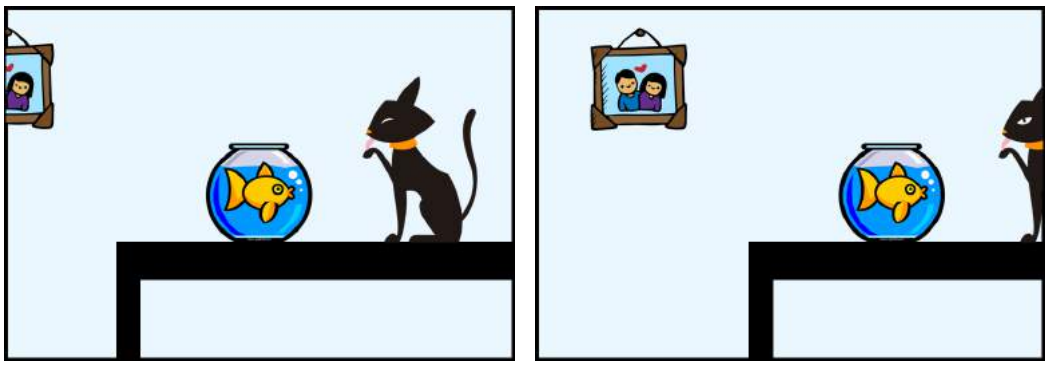
[9]Existem coordenadas de ecrã e coordenadas de fotografia uma vez que a resolução da fotografia pode ser maior/menor que a do *container*.

  - ***photoCenter***: Coordenadas do ponto que está no centro do *container*;
  - ***scaleSXY***: Escala aplicada à fotografia carregada no *container*.

- As fotografias presentes nas Figuras 4.5a e 4.5b serão utilizadas como exemplos para a demonstração da comparação lado a lado, sendo a Figura 4.5a a representante da fotografia pivô e a Figura 4.5b a da fotografia subordinada.
- Nas listagens apresentadas ao longo das próximas secções pode não constar todo o código efetivamente existente para as respetivas classes e/ou métodos.

Listagem 4.2: Método responsável pelas transformações base aplicadas sobre as fotografias.

```
1
2  open fun applyTransformation() {
3    val transformationMatrix = Matrix()
4
5    transformationMatrix.postTranslate(-photoCenter.x, -photoCenter.y)
6    transformationMatrix.postScale(scaleXY, scaleXY)
7    transformationMatrix.postTranslate(containerCenter.x, containerCenter.y
       ↪ )
8
9    photo.imageMatrix = transformationMatrix
10 }
```



(a) Fotografia pivô exemplo. (b) Fotografia subordinada exemplo.

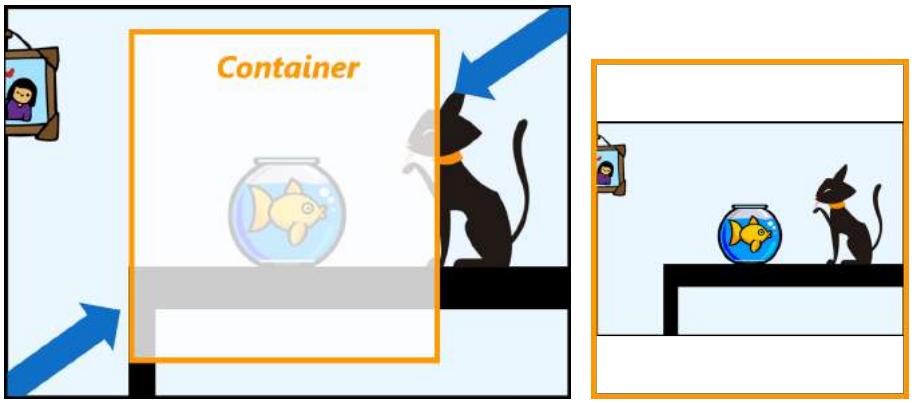
Figura 4.5: Exemplos de fotografias para demonstração do funcionamento da comparação lado a lado.

4.6.1 Apresentação Inicial das Fotografias

A apresentação inicial das fotografias no ecrã consiste em duas etapas: a visualização total e centrada da fotografia pivô dentro do seu *container*; e a sincronização/foco do mesmo

motivo entre a fotografia pivô e a fotografia subordinada. A apresentação sincronizada no mesmo motivo está dependente de uma matriz de homografia que correlaciona as coordenadas da fotografia pivô com as coordenadas da fotografia subordinada. Caso a matriz de homografia ainda não esteja presente na base de dados da aplicação (no caso da primeira comparação entre duas fotografias) é necessário efetuar um pedido ao servidor para retorno da matriz de homografia entre a fotografia pivô e a fotografia subordinada[10]. Para obter as coordenadas da fotografia pivô através de coordenadas da fotografia subordinada é necessário calcular a inversa da matriz de homografia retornada pelo servidor *photo | uniq*[11].

Na Figura 4.6a é apresentado um exemplo de uma fotografia com maior resolução e formato diferente do *container* onde será carregada. Não sendo possível alterar o formato da fotografia ou o formato do *container* e, sendo a resolução da fotografia maior do que a do seu *container*, é necessário encontrar uma escala uniforme que permita o carregamento da fotografia totalmente e com o seu formato original no *container*. Depois de obtida a escala, segue-se o momento de colocar as coordenadas do centro da fotografia no centro do *container*, de forma a obter o resultado apresentado na Figura 4.6b. Assim que efetuados estes passos, a primeira etapa da apresentação inicial das fotografias está completa.



(a) Fotografia com maior resolução do que o seu *container*.

(b) Fotografia centrada e totalmente visível dentro do *container*.

Figura 4.6: Apresentação inicial da fotografia pivô (visualização total e centrada da fotografia no *container*).

Na Listagem 4.3 são apresentados os métodos responsáveis por apresentar a fotografia pivô centrada e na sua totalidade no *container*. O primeiro passo consiste em obter quais as coordenadas do ponto que se encontra no centro da fotografia (na linha 4) e qual o valor menor entre as duas escalas (linhas 6 a 8) para colocar a fotografia com o mesmo

[10] A ordem com que as fotografias são enviadas para o servidor importa, uma vez que se o servidor receber a fotografia A como primeiro parâmetro e a fotografia B como segundo, a matriz de homografia retornada correlacionará os pontos da fotografia A com os da fotografia B, e não o contrário.

[11] Outra possibilidade para obter a matriz de homografia que faz a correspondência dos pontos da fotografia subordinada com os pontos da fotografia pivô poderia ser uma nova chamada ao serviço de homografia, enviando as fotografias nos parâmetros pela ordem contrária à do primeiro pedido.

formato, centrada e inteiramente no ecrã. De seguida, é chamado o método (linha 10) que contém os passos base ilustrados acima, no final da Secção 4.6, para as transformações base. Como último passo do método de apresentação inicial da fotografia pivô, é chamado o método responsável por sincronizar o mesmo motivo para ambas as fotografias pivô e subordinada (linha 18).

Listagem 4.3: Apresentação inicial da fotografia pivô.

```
1
2  fun applyDefaultFitTransformation() {
3
4    photoCenter.set(bmp!!.width.toFloat() / 2, bmp!!.height.toFloat() / 2)
5
6    val sX = ((widthScreen.toFloat() / 2) / bmp!!.width)
7    val sY = (heightScreen.toFloat() / bmp!!.height)
8    scaleXY = Math.min(sX, sY)
9
10   applyTransformation()
11
12 }
13
14 override fun applyTransformation() {
15   super.applyTransformation()
16
17   mappedPhoto?.sync()
18
19 }
```

Relembrando o objetivo da segunda fase, sincronizar o mesmo motivo entre as fotografias pivô e subordinada, para apresentar a fotografia subordinada utiliza-se o mesmo mecanismo que para apresentar a fotografia pivô mas com um ponto diferente a centrar (não as coordenadas do centro da fotografia). Neste caso, as coordenadas a apresentar no centro do *container* da fotografia subordinada, são as coordenadas obtidas através da multiplicação do ponto que se encontra no centro da fotografia pivô com a matriz de homografia. Ao utilizar estas coordenadas "convertidas" (subordinadas) é garantida a sincronização do mesmo motivo para ambas as fotografias, como visível na Figura 4.7.

Listagem 4.4: Apresentação inicial da fotografia subordinada.

```
1 fun sync() {
2
3   this.scaleXY = pivotFrag.getScale()
4
5   val pivotPhotoCenter = pivotFrag.getCurrentPhotoCenter()
```

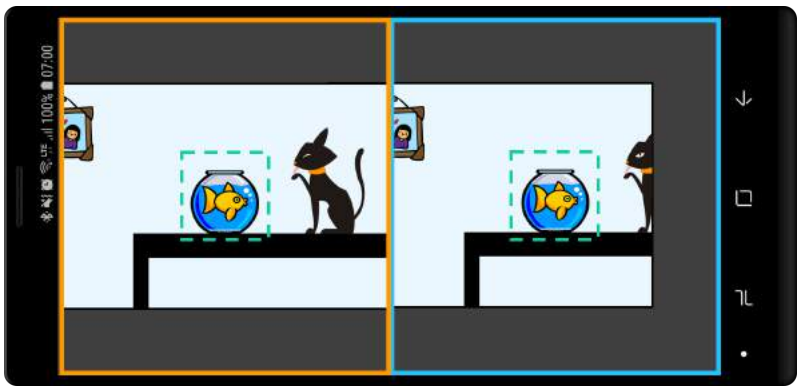


Figura 4.7: Apresentação inicial das fotografias pivô e subordinada, com sincronização no mesmo motivo.

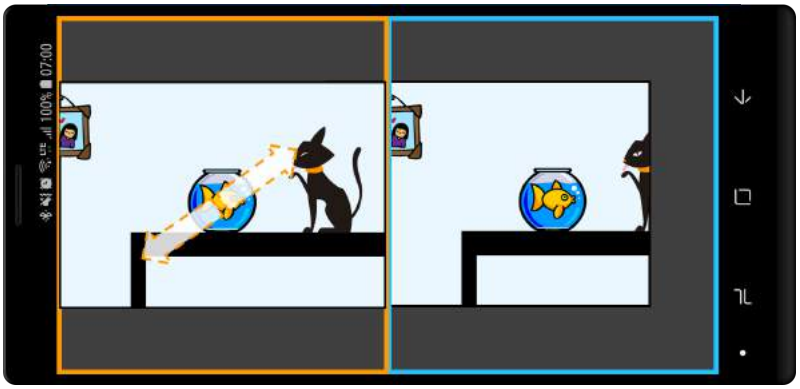
```
6   photoCenter = convertCoordsAinB(pivotPhotoCenter.x, pivotPhotoCenter.y,
        ↪ matrixABValues)
7
8   applyTransformation()
9
10  }
11
12  private fun convertCoordsAinB(coordAX: Float, coordAY: Float, matrix:
        ↪ FloatArray?): PointF {
13
14    val newPointInArray = floatArrayOf(coordAX, coordAY, 1.0f)
15    val newPointAinB = multiplyMatrixByVector(matrix, newPointInArray)
16    val w = newPointAinB[2]
17
18    val coordBX = newPointAinB[0] / w
19    val coordBY = newPointAinB[1] / w
20    return PointF(coordBX, coordBY)
21
22  }
```

Na Listagem 4.4 são apresentados os métodos responsáveis por apresentar a fotografia subordinada sincronizada no mesmo motivo com a fotografia pivô. O método *sync* tem como objetivos atualizar as variáveis de estado da fotografia subordinada (linhas 3 e 5), converter as coordenadas do centro da fotografia pivô em coordenadas da fotografia subordinada (linha 6) e chamar do método *applyTransformation* para aplicar as transformações base sobre a fotografia subordinada (linha 8).

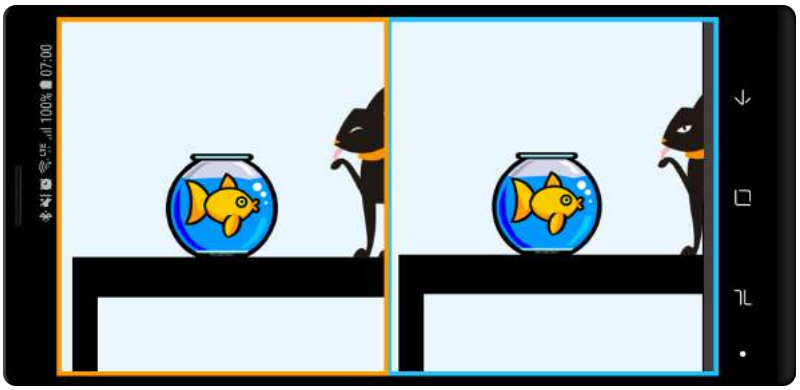
4.6.2 Interação com as Fotografias

As interações disponibilizadas para as fotografias pivô e subordinada são: a Ampliação/-Redução e o Reposicionamento. Neste contexto, as interações mencionadas são relativas a gestos efetuados pelo utilizador sobre cada uma das fotografias. Para que possam ser efetuadas operações quando um gesto é efetuado, cada fotografia tem um *listener* de gestos associado. Os gestos detetáveis pelo *listener* são: o toque único com um dedo, o duplo toque com um dedo, o arrastar de um dedo pelo ecrã, o colocar de dois dedos próximos e afastar, e o colocar de dois dedos afastados e aproximar. As ações associadas a estes gestos são apresentadas de seguida.

4.6.2.1 Ampliação/Redução



(a) Ampliação da área de interesse, na fotografia pivô.



(b) Estado das fotografias após a ampliação sobre a fotografia pivô.

Figura 4.8: Demonstração de ampliação iniciada a partir da fotografia pivô.

Na Figura 4.8 é demonstrada a ampliação sobre as fotografias. Neste caso, a ampliação é iniciada na fotografia pivô através do colocar de dois dedos próximos e consecutivo afastamento. Tanto durante, como após o gesto de ampliação a fotografia subordinada acompanha a escala aplicada sobre a fotografia pivô e mantêm-se sempre sincronizada no mesmo motivo que a fotografia pivô.

Enquanto que a ampliação da fotografia pode ser conseguida através de um duplo toque na área de interesse ou a um colocar dois dedos próximos e afastar, a redução

pode ser obtida através do duplo toque em qualquer área do ecrã (sendo necessário que exista alguma pré-ampliação na imagem) ou então a colocação de dois dedos afastados e a consecutiva aproximação. Sempre que algum destes gestos é efetuado sobre uma das fotografias, é gerado um fator de escala. Por cada fator de escala gerado, é chamado o método *scale* de cada uma das fotografias e o respetivo fator de escala é passado como parâmetro.

Listagem 4.5: Método responsável pela Ampliação/Redução.

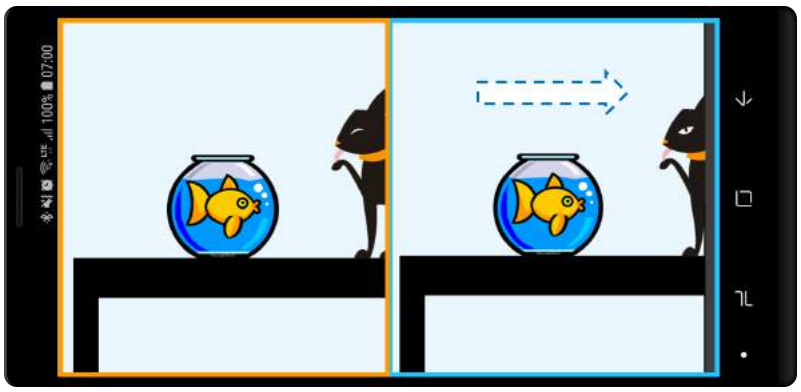
```
1 fun scale(scaleFactor: Float) {
2
3   currentSXY = scaleFactor
4
5   applyTransformation()
6
7 }
```

Na Listagem 4.5 é apresentado o método de ampliação/redução das fotografias pivô e subordinada. Neste caso, o método é bastante simples, uma vez que apenas é preciso atualizar a variável global *currentSXY* e chamar a função *applyTransformation* para aplicar a alteração (o novo fator de escala).

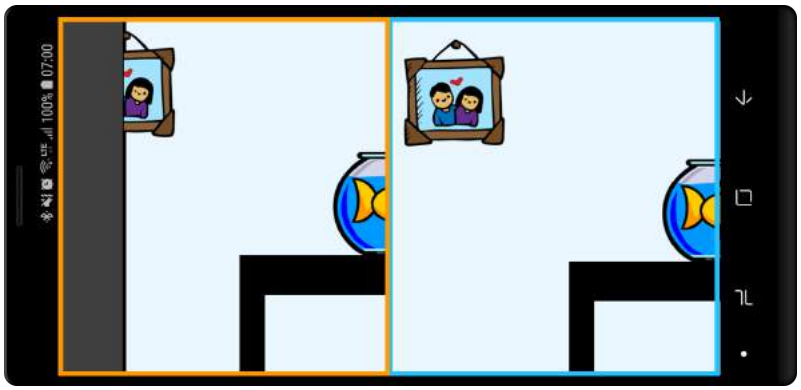
4.6.2.2 Reposicionamento

O reposicionamento difere do método de ampliação/redução, não só pela sua funcionalidade, como também pelo seu modo de operação. Enquanto que na ampliação/redução é despoletado o método de escala para ambas as imagens ao mesmo tempo, no reposicionamento é chamado um método diferente para cada fotografia, consoante o gesto seja efetuado na fotografia pivô ou na subordinada. Na Figura 4.9 é possível ver um exemplo de um reposicionamento da área de interesse, iniciado a partir da fotografia subordinada. Tal como na ampliação/redução, as fotografias encontram-se sempre sincronizadas sobre o mesmo motivo.

O reposicionamento do ponto que está no centro do *container* pode ser obtido através do arrastar do dedo pela tela do ecrã (tanto no *container* da esquerda como no da direita). Sempre que este gesto é efetuado, o *listener* de gestos gera um par de coordenadas (x,y) que identifica a localização atual do dedo no *container* e chama o método *onScroll* da fotografia onde foi iniciado o gesto. O método *onScroll* tem como parâmetros as coordenadas (de ecrã) retornadas pelo *listener* de gestos enquanto o utilizador arrasta o dedo pelo ecrã. Sendo estas coordenadas, coordenadas de ecrã, o primeiro passo da função é converter as coordenadas de ecrã para coordenadas de fotografia, dividindo-as pelo fator de escala atualmente aplicado. De seguida são apresentados separadamente os restantes passos do método *onScroll*, caso o gesto seja iniciado a partir da fotografia pivô (ver Listagem 4.6) e caso o gesto seja iniciado a partir da fotografias subordinada (ver Listagem 4.7).



(a) Reposicionamento da área de interesse, a partir da fotografia subordinada.



(b) Estado das fotografias após o reposicionamento da fotografia subordinada.

Figura 4.9: Demonstração de reposicionamento da área de interesse, iniciado a partir da fotografia subordinada.

Listagem 4.6: Reposicionamento iniciado a partir da fotografia pivô (função da fotografia pivô).

```
1  override fun onScroll(dX: Float, dY: Float) {
2    val dImageX = dX / scaleXY
3    val dImageY = dY / scaleXY
4
5    changeFocus(photoCenter.x + dImageX, photoCenter.y + dImageY)
6  }
7
8  fun changeFocus(x: Float, y: Float) {
9    photoCenter.set(x, y)
10   applyTransformation()
11 }
```

No algoritmo de reposicionamento iniciado a partir da fotografia pivô, visível na Listagem 4.6, os passos seguintes ao cálculo das coordenadas de fotografia (linhas 3 e 4) são:

a soma das coordenadas de fotografia com as coordenadas do ponto que se encontra atualmente no centro da fotografia pivô (linhas 6 e 7); e a chamada do método *changeFocus* que trata de atualizar as variáveis de estado e chamar o método *applyTransformation()*.

Listagem 4.7: Reposicionamento iniciado a partir da fotografia subordinada (função da fotografia subordinada).

```
1  override fun onScroll(dX: Float, dY: Float) {
2    val dImageX = dX / scaleXY
3    val dImageY = dY / scaleXY
4
5    val coordinatesB = floatArrayOf(photoCenter.x + dImageX, photoCenter.y
       ↪ + dImageY, 1F)
6    val pointBinA = multiplyMatrixByVector(matrixBAValues!!, coordinatesB)
7
8    val w = pointBinA[2]
9    val coordX2D = pointBinA[0] / w
10   val coordY2D = pointBinA[1] / w
11
12   pivotFrag?.changeFocus(coordX2D, coordY2D)
13 }
```

Já para o caso do algoritmo de reposicionamento efetuado a partir da fotografia subordinada e visível na Listagem 4.7, o passo seguinte ao cálculo das coordenadas fotografia (linhas 3 e 4) é a soma destas às coordenadas do ponto atualmente focado na fotografia subordinada (linha 5) e conversão destas novas para as coordenadas correspondentes na fotografia pivô (linha 6). Depois de calculadas as coordenadas do ponto a centrar no *container* da fotografia pivô, segue-se a conversão destas para o formato correto 2D (linhas 8 a 10) e, por fim, a execução da função *changeFocus* da fotografia pivô (visível na Listagem 4.6). O método *changeFocus* tratará de atualizar as coordenadas do centro da fotografia pivô e de chamar o método *applyTransformation* também da fotografia pivô, que por sua vez, chamará o método de sincronização no mesmo motivo da fotografia subordinada (método *sync*). Por fim, de forma a ajudar na compreensão de toda a lógica por detrás do reposicionamento iniciado a partir da fotografia subordinada, na Figura 4.10 é apresentado o *workflow* do conjunto de ações e métodos chamados a partir do momento em que é detetado um gesto na fotografia subordinada.

4.7 Bibliotecas Externas

No desenvolvimento da aplicação *photo | uniq* foram utilizadas duas bibliotecas externas às padrão fornecidas pelo próprio Android: o *Glide*, para *caching* de imagens e criação de miniaturas, e o *Retrofit* para implementação da API REST.

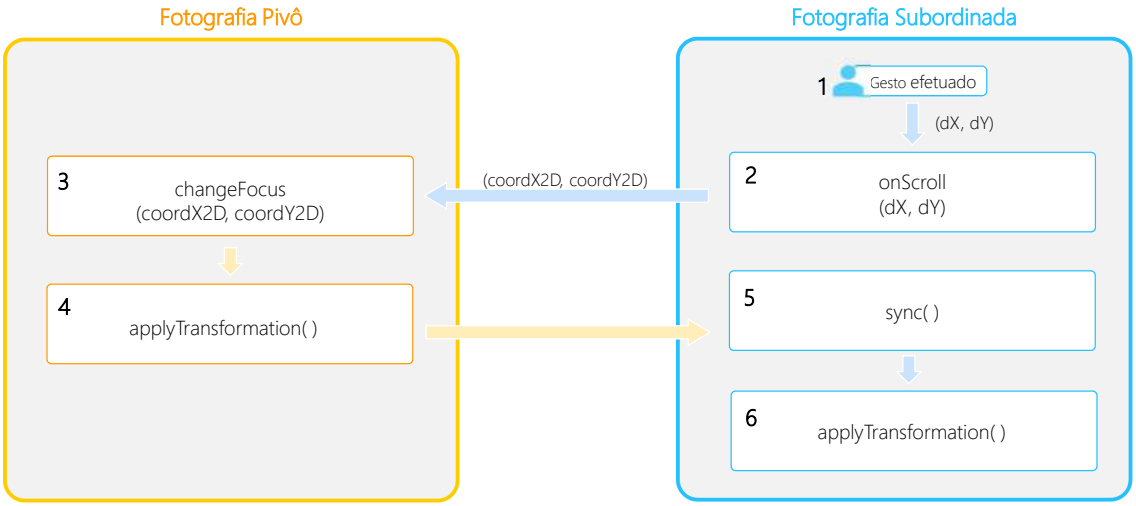


Figura 4.10: *Workflow* de reposicionamento iniciado na fotografia subordinada.

4.7.1 *Glide*

O *Glide* é uma biblioteca de carregamento rápido e eficiente de imagens, vídeos e *gifs* para a plataforma Android. A biblioteca *Glide* tem como principal objetivo garantir que qualquer conjunto de imagens é carregado rápida e suavemente, sem causar distúrbios para o utilizador.

O *Glide* fornece uma API bastante fácil de utilizar, com métodos de carregamento de imagens pré-concebidos, mas permite também a customização de muitas das suas configurações (padrão) para utilizadores avançados. Algumas características/funcionalidades interessantes da biblioteca *Glide* são [46]:

- Carregamento de imagens, podendo as imagens estar armazenadas no dispositivo ou na *cloud*;
- Colocação de *placeholder*, enquanto a imagem não é carregada;
- Transições entre o *placeholder* definido e a imagem a apresentar;
- Definição do formato (resolução) da imagem a ser carregado, por exemplo, tamanho original ou miniatura (*thumbnail*);
- *Caching* de imagens;
- Gestão da memória utilizada em todo o processo de carregamento das imagens;
- Reutilização de recursos (imagens), de modo a diminuir a carga sobre o coletor de lixo (*garbage collector*).

Entre o conjunto de funcionalidades disponíveis na biblioteca, no âmbito da aplicação *photo | uniq*, destacam-se duas das funcionalidades: o *caching* de imagens e a criação de miniaturas. Estas duas funcionalidades são as mais importantes no âmbito da aplicação

photo | uniq, uma vez que esta pode lidar com muitas imagens e com imagens muito grandes (resoluções grandes).

No caso das miniaturas, estas são utilizadas sempre que o utilizador não precise de ver a imagem no seu tamanho/formato original, ou seja, em todas as listagens de fotografias. Já no caso da funcionalidade de *cache* de imagens, a *cache* é utilizada ao longo de toda a aplicação para garantir que o carregamento das imagens é bastante mais rápido. Se não fosse feito *caching* das imagens, as imagens teriam de estar sempre a ser recarregadas das fontes e, consequentemente, seriam provocados atrasos na utilização da aplicação.

Por fim, na Listagem 4.8 é possível ver um exemplo de carregamento de uma imagem com fonte "*url*", para uma vista "*imageview*", dentro de um contexto "*fragment*".

Listagem 4.8: Carregamento de uma imagem utilizando a biblioteca *Glide*.

```
Glide.with(fragment)
   .load(url)
   .into(imageView)
```

4.7.2 *Retrofit*

A *Retrofit* é uma biblioteca que fornece uma implementação de API REST e que tem como funcionalidade base transmitir dados entre uma aplicação *Android* (ou Java) e serviços *REST*. Esta biblioteca facilita bastante todos os pedidos efetuados a um servidor REST, uma vez que fornece uma estrutura robusta para autenticação e criação de pedidos, para além de ainda fornecer conversores para serialização dos dados retornados pelos pedidos[47].

Os *endpoints* e assinaturas dos pedidos são definidos dentro de uma interface (ver Listagem 4.9), sendo utilizadas anotações especiais para codificar os parâmetros e os métodos do protocolo *HTTP* (*GET, POST, PUT*, entre outros).

Listagem 4.9: Interface dos pedidos REST da aplicação *photo | uniq*.

```
interface APIInterface {

  @Headers("Content-Type: application/json")
  @GET("albums")
  fun getAlbums(@Query("user_id") userId: Int): Call<List<Album>>

  @Headers("Content-Type: application/json")
  @POST("album")
  fun postAlbum(@Query("user_id") userId: Int, @Body albumJson:
      ↪ JsonObject): Call<Int>

  @Headers("Content-Type: application/json")
```

```
12   @GET("album")
13   fun getAlbum(@Query("user_id") userId: Int, @Query("album_id") albumId:
         ↪ Int): Call<Album>
14
15   (...)
16 }
```

Focando apenas num dos métodos, por exemplo o de retorno de todos os álbuns de um determinado utilizador (linhas 3 a 5 da Listagem 4.9), é possível identificar duas anotações:

- *@Headers*, que permite enviar informações extras no pedido através dos cabeçalhos;
- *@GET*, que define o método do protocolo *HTTP* e respetivo *endpoint*.

O passo seguinte às anotações é a definição da função e parâmetros necessários para o pedido. Neste caso, o único parâmetro necessário é o identificador único do utilizador. Por fim, temos a definição do tipo de retorno do pedido, neste caso, uma lista de todos os álbuns do utilizador. É possível definir uma lista de álbuns como retorno graças aos conversores fornecidos pelo *Retrofit*.

Depois de todos os métodos serem definidos na interface, segue-se a implementação do código encarregue de efetuar os pedidos ao servidor, para além da implementação do que deve acontecer quando algum destes métodos é efetuado (com ou sem sucesso).

Listagem 4.10: Classe dedicada à interação com o servidor.

```
1  const val NOT_FOUND = -1
2  const val SERVERURL = "http://192.111.1.11:5000/api/"
3
4  class BackOffice {
5
6    private var apiInterface: APIInterface = getClient().create(
         ↪ APIInterface::class.java)
7
8    fun getClient(): Retrofit {
9      val client = OkHttpClient.Builder()
10       .connectTimeout(30, TimeUnit.SECONDS)
11       /*.addInterceptor { chain ->
12         val token = preferences.getString("TOKEN", "")
13
14         val originalRequest = chain.request()
15         val builder = originalRequest.newBuilder()
16         .header("Authorization", "Bearer " + token!!)
17         val newRequest = builder.build()
```

```
18          chain.proceed(newRequest)
19        }*/
20        .build()
21
22
23      return Retrofit.Builder()
24        .baseUrl(SERVERURL)
25        .addConverterFactory(GsonConverterFactory.create())
26        .client(client)
27        .build()
28    }
29
30    fun getAlbums(activity: Activity, userID: Int, listener: Listener<Any>)
          ↪ {
31
32      apiInterface.getAlbums(userID).enqueue(object : Callback<List<Album
            ↪ >>() {
33
34        override fun onResponse(call: Call<List<Album>>, response: retrofit2.
              ↪ Response<List<Album>>) {
35
36          if (response.isSuccessful) {
37            val albums = response.body!!
38            (...)
39          } else
40            serverError(call, response, listener)
41
42        }
43
44        override fun onFailure(call: Call<List<Album>>, t: Throwable) {
45          clientError(t, listener)
46        }
47
48      })
49    }
50
51    (...)
52
53  }
```

Na Listagem 4.10 é apresentada a classe que lida com todas as interações com o servidor.

Nas linhas 5 a 20 é possível ver um exemplo de como criar um cliente HTTP utilizando o *Retrofit*, incluindo a sua própria configuração. Neste caso, o cliente HTTP criado dará *timeout* num pedido se o tempo de conexão ao servidor for superior a 30 segundos (linha 10). Nas linhas 11 a 19 é possível ver um exemplo de como adicionar autenticação, por exemplo, *Bearer Authentication*[12], a todos os pedidos. Já no caso das configurações do *Retrofit*, este fará pedidos ao *endPoint* identificado por *SERVERURL* e utilizará um conversor *Gson* para as respostas dos pedidos do servidor. Por fim, nas linhas 30 a 49, encontra-se um exemplo de implementação de uma chamada ao servidor (*getAlbums*), onde é possível definir o que deve acontecer caso o pedido seja efetuado com sucesso (linha 37), caso falhe no lado do servidor (linha 40) ou caso falhe no lado do cliente (linha 45).

[12] *Bearer Authentication* é um tipo de autenticação HTTP que envolve *tokens* de segurança para serem enviados nos pedidos ao servidor. Estes *tokens* têm de ser requisitados ao servidor periodicamente, uma vez que têm um tempo de expiração para se tornarem inválidos [48].

Capítulo 5

Validação do Sistema

Uma vez que algumas funcionalidades essenciais da aplicação *photo | uniq* envolvem interação com um servidor que ainda não está operacional, não considerámos viável lançar uma primeira versão *alpha* da aplicação para efetuar os testes de usabilidade e experiência de utilização com utilizadores externos. No entanto, ao longo do desenvolvimento, para além das maquetas previamente pensadas e concebidas, foram sempre efetuados testes de desempenho e de recursos consumidos de modo a melhorar continuamente a aplicação *photo | uniq*.

Assim, este capítulo tem como objetivos validar funcionalmente o protótipo desenvolvido, em particular avaliar a escalabilidade da aplicação com o aumento do número de fotografias, avaliar os recursos consumidos em função do tamanho das imagens e avaliar o desempenho da aplicação no carregamento das fotografias.

Para validar a performance da aplicação *photo | uniq* foram efetuados testes executando a aplicação com galerias grandes/pequenas — Escalabilidade — ou com imagens de grandes/pequenas resoluções — Perfil de Consumo de Memória e Desempenho — para que, após uma análise dos dados recolhidos, fosse possível identificar quais os pontos a melhorar.

No decorrer dos testes que envolviam a medição do tempo foram recolhidas 5 medidas para cada um dos casos de uso, sendo depois descartados os valores mais alto e mais baixo deste conjunto e fazendo-se a média para os restantes 3 valores. Além disso, foram utilizados dois dispositivos móveis: um Wiko Highway Star 4G e um Samsung Galaxy S8, um tecnologicamente mais antigo e outro mais recente, respetivamente. Na Tabela 5.1 encontram-se as principais especificações de cada um dos dispositivos.

Nota Nas próximas secções serão apresentados gráficos cujos dados/tabelas estão no Apêndice B. Para cada um dos gráficos, a respetiva tabela aparece referenciada na legenda.

Tabela 5.1: Tabela comparativa de especificações entre o Wiko Highway Star 4G e o Samsung Galaxy S8 (retirado de [49, 50]).

	Wiko Highway Star 4G	**Samsung Galaxy S8**
Data	Abril de 2015	Abril de 2017
Tipo de Ecrã	Touchscreen capacitivo AMOLED, 16 milhões de cores	Touchscreen capacitivo super AMOLED, 16 milhões de cores
Resolução de Ecrã	1280x720 HD (~294 ppi)	2960x1440 Quad HD+ (~570 ppi)
Sistema Operativo	Android 5.1 (Lollipop)	Android 9 (Pie)
CPU	Octa-core 1.5 GHz	Octa-core 2.3 GHz & 1.7 GHz
GPU	ULP GeForce	Mali-G71 MP20
Memória ROM (GB)	16	64
Memória RAM (GB)	2	4

5.1 Escalabilidade

Para avaliar a escalabilidade da aplicação *photo | uniq* medimos o tempo de abertura da aplicação para os seguintes casos de uso:

- Primeira utilização da aplicação, que depende do número de fotografias existentes na galeria do telemóvel;
- Restantes utilizações da aplicação, que depende do número de alterações — fotografias novas e apagadas — realizadas na galeria do dispositivo desde a última vez que se utilizou a aplicação *photo | uniq*.

Tal como mencionado na Secção 4.1, na página 58, quando a aplicação inicia é executado o método responsável pela sincronização entre a base de dados do telemóvel e a base de dados da aplicação *photo | uniq*, bem como pelo preenchimento da informação volátil[1]. Muito sucintamente, este método de sincronização inicial executa os seguintes passos:

1. Obter o ID (da fotografia) com valor mais alto na base de dados da aplicação;
2. Iterar pelo conjunto de fotografias do dispositivo, criando a lista de todos os IDs das fotografias e a lista de fotografias novas (desde a última utilização da aplicação);
3. Atualizar da base de dados da aplicação, tendo como base as listas geradas no passo anterior;

[1] O repositório de informação volátil tem como função manter em memória todos os dados necessários para o (rápido) funcionamento da aplicação.

4. Obter a lista das fotografias da base de dados da aplicação;

5. Preencher a informação volátil com base na lista obtida no passo 4.

Nas próximas secções serão apresentados os gráficos referentes a cada um dos tipos de testes — primeira e restantes utilizações da aplicação — e será feita uma análise sobre esses mesmos dados.

5.1.1 Primeira Utilização da Aplicação

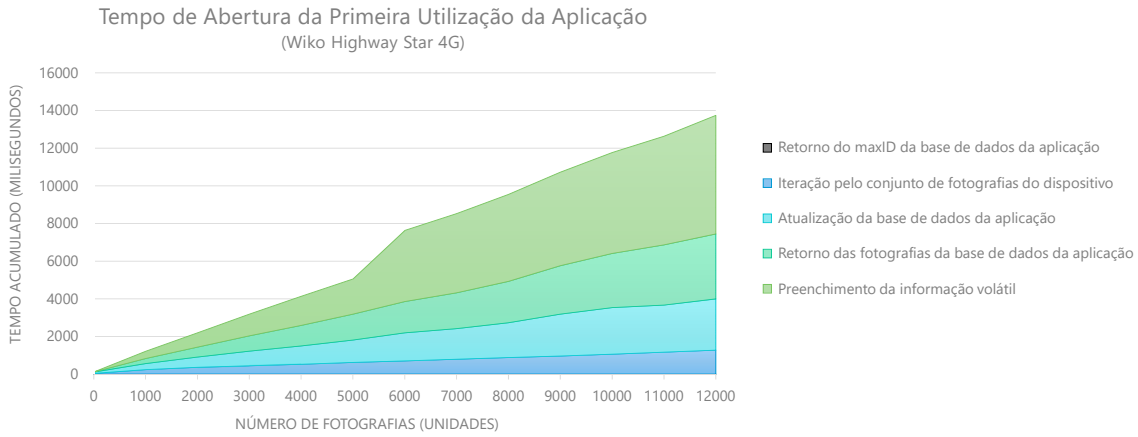
Com a base de dados vazia, variando o número de fotografias que a aplicação tinha de iterar e inserir na sua base de dados, mediu-se os tempos de execução dos passos 1 a 5 do método de sincronização inicial (ver gráficos das Figuras 5.1 e 5.2).

Nos gráficos das Figuras 5.1a e 5.1b é possível observar que à medida que o número de fotografias aumenta, o tempo de execução de cada um dos passos também aumenta linearmente[2]. Assim, estando perante um comportamento linear para ambos os dispositivos, é possível concluir que o padrão de comportamento é independente do *hardware*.

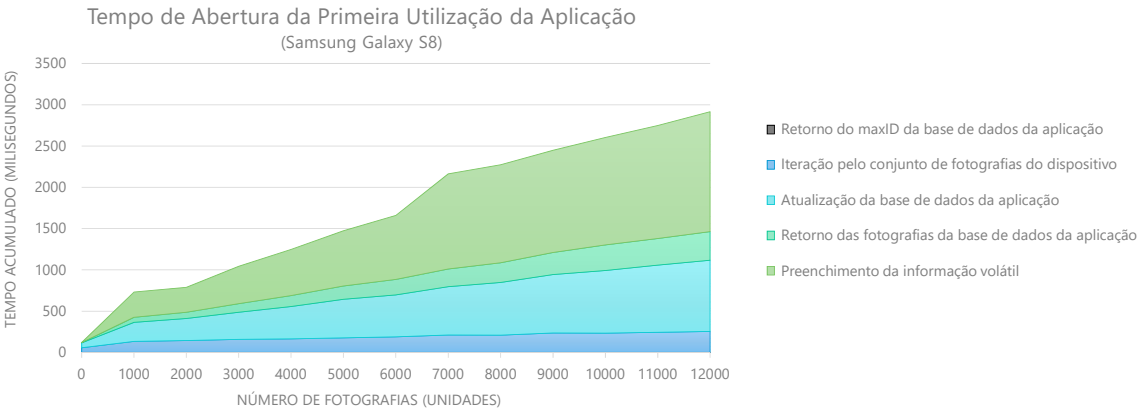
Para ambos os dispositivos, o tempo necessário para identificar a fotografia na base de dados da aplicação que tem o ID mais elevado é quase nulo, o que é natural pois neste caso de estudo a base de dados da aplicação está sempre vazia. Considerando agora as restantes operações sobre a base de dados (atualização e retorno das fotografias), verifica-se que a base de dados foi otimizada entre as versões dos dispositivos, uma vez que a área referente às leituras sobre a base de dados é bastante menor para o Samsung. Por fim, é possível observar que o preenchimento da informação volátil é o passo mais dispendioso para ambos os dispositivos, uma vez que é proporcional ao número de fotografias existentes na base de dados da aplicação.

Na Figura 5.2 é apresentado o tempo total para a execução do método de sincronização inicial para cada um dos dispositivos. Neste gráfico é possível ver que o Samsung apresenta tempos bastante menores que o Wiko. No intervalo $[0, 5000]$ de número de fotografias, o Samsung nunca chega a demorar mais de 2 segundos para executar o método, enquanto que o Wiko ultrapassa os dois segundos logo nas 2000 fotografias. No intervalo $[5000, 6000]$ de número de fotografias, o Wiko apresenta uma subida temporal mais acentuada, passando de 5 segundos para aproximadamente 8 segundos, enquanto que o Samsung se mantém linear. Por fim, para o restante intervalo ($[6000, 12000[$) de número de fotografias, é possível verificar que a diferença do tempo de computação entre os dois dispositivos se torna cada vez mais acentuada, chegando quase aos 14 segundos no Wiko para 12000 fotografias e apenas aos 3 segundos nas mesmas circunstâncias para o Samsung. Posto isto, comprova-se que quanto mais recente tecnologicamente for o dispositivo, mais rápida será a execução do método.

[2]Ter em atenção que o eixo vertical dos gráficos presentes nas Figuras 5.1a e 5.1b não se encontra na mesma escala de valores.



(a) Wiko Highway Star 4G - Tabela A.1.



(b) Samsung Galaxy S8 - Tabela A.2.

Figura 5.1: Tempo de abertura da primeira utilização da aplicação.

5.1.2 Segunda Utilização e Seguintes

No gráfico da Figura 5.3 são apresentados os resultados da avaliação de qual a operação que requer mais tempo de execução sobre a base de dados: a inserção de novas fotografias ou a remoção de fotografias. Para efetuar esta avaliação, utilizámos somente o Samsung e tínhamos 5000 fotografias na base de dados da aplicação. Obtidos os resultados, verificou-se que a inserção de 2500 fotografias sobre essas 5000 demorava aproximadamente 210 milissegundos e a remoção de 2500 das 5000 fotografias demorava somente 50 milissegundos. Assim, validou-se que a inserção de novas fotografias demorava bastante mais tempo (aproximadamente 76%) do que a remoção de fotografias, o que faz sentido segundo os tempos medidos, nas secções anteriores, para cada um dos passos da sincronização das bases de dados e inicialização da informação volátil.

Na Figura 5.4 é apresentado um gráfico para o tempo de abertura da aplicação em função do número de alterações efetuadas sobre a Galeria Local. Para cada iteração do teste, a Galeria Local era inicializada com 5000 fotografias[3] e depois eram efetuadas alterações sobre a Galeria Local. Neste teste, executado somente para o Samsung, metade

[3]O tempo de inserção das 5000 fotografias na Galeria Local não foi contabilizado para o teste.

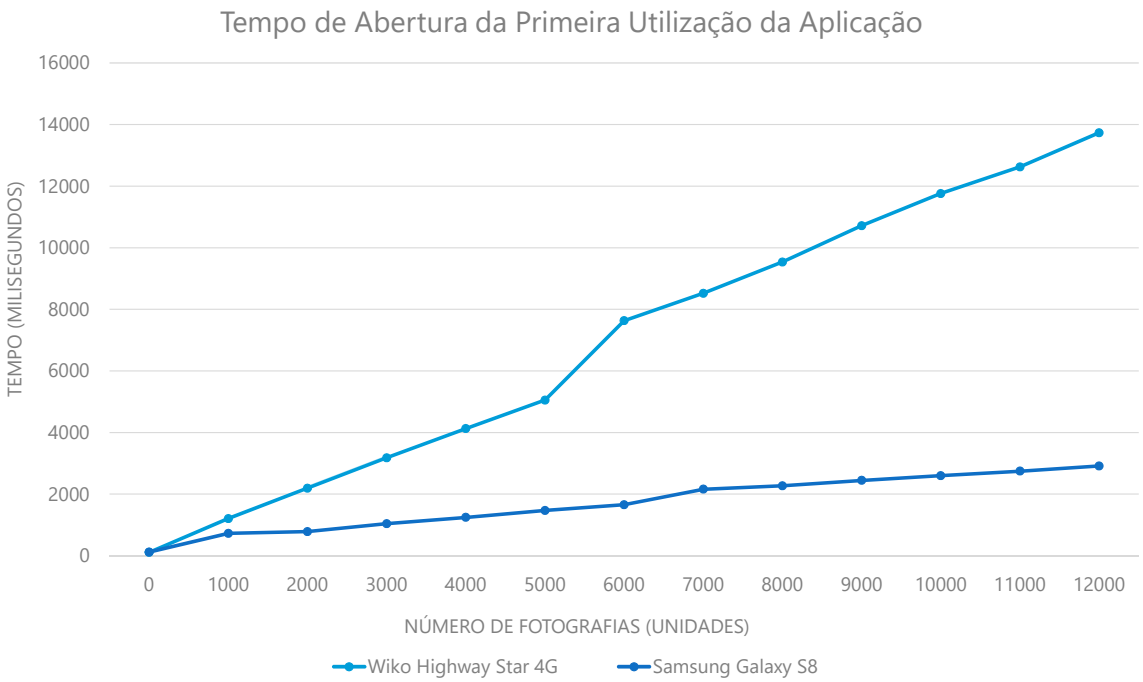


Figura 5.2: Tempo de abertura da primeira utilização da aplicação para ambos os dispositivos - Tabela A.3.

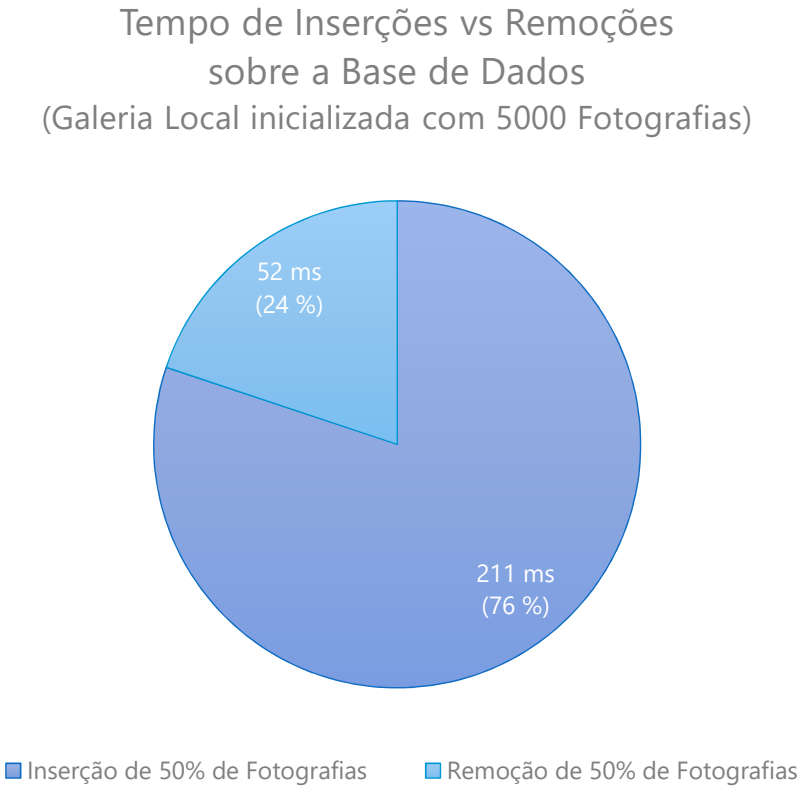


Figura 5.3: Tempo de inserção de fotografias vs remoção de fotografias, no Samsung Galaxy S8 - Tabela A.4.

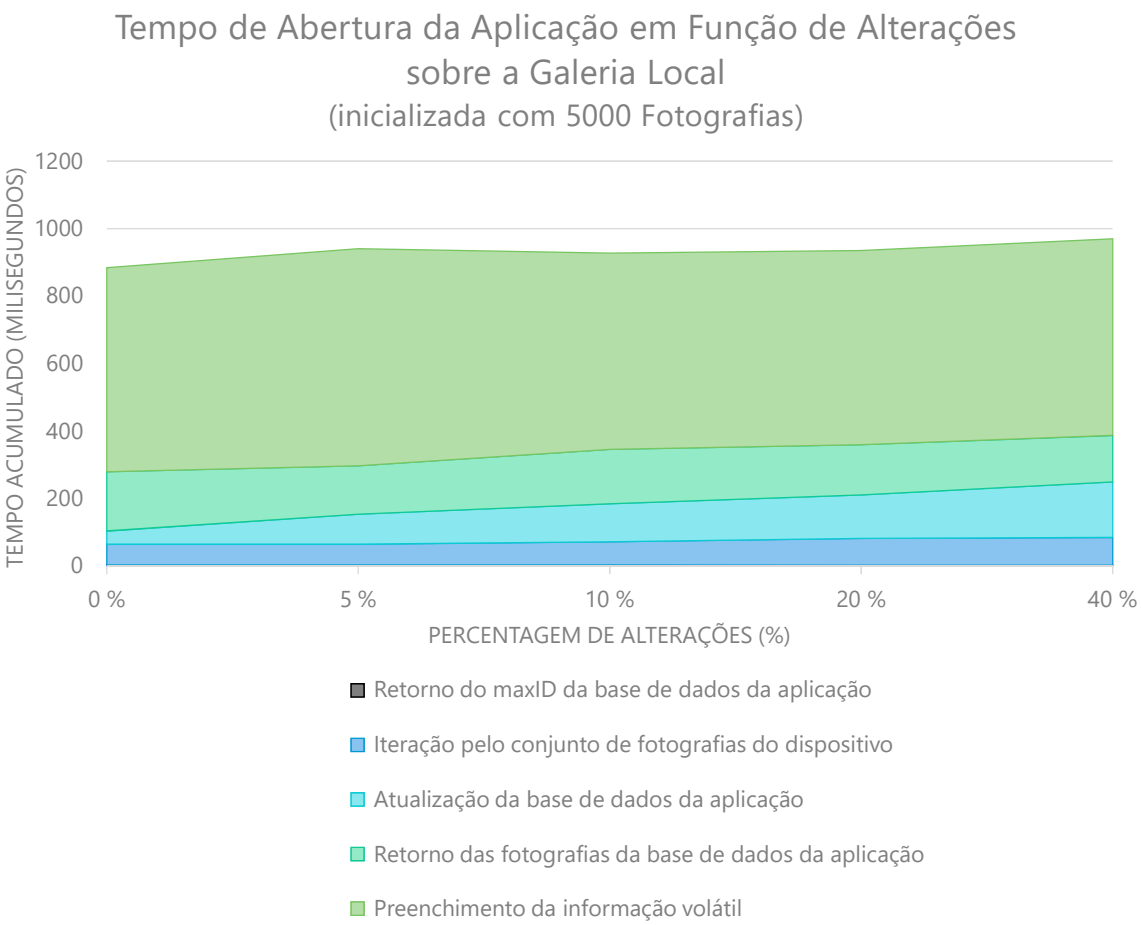


Figura 5.4: Tempo de abertura da aplicação em função de alterações sobre a Galeria Local, no Samsung Galaxy S8 - Tabela A.5.

das alterações introduzidas na base de dados da Galeria Local deviam-se à inserção de fotografias novas e a outra metade à remoção de fotografias. Por exemplo, o teste efetuado para 10% de alterações, consistiu em inserir 250 fotografias novas e remover 250 fotografias do conjunto inicial de 5000 fotografias. No final de cada iteração, a Galeria Local mantinha-se com um total de 5000 fotografias.

O gráfico da Figura 5.4 mostra que, independentemente do número de alterações efetuadas, o tempo total de abertura da aplicação se mantém constante, dentro do intervalo $[800, 1000]$ milissegundos. Cada um dos passos do método de sincronização inicial mantém-se também constante ao longo do intervalo $[0, 40]$ de alterações.

Focando-nos nos 40% de alterações, onde foram inseridas 1000 fotografias e removidas outras 1000, o tempo total de abertura da aplicação foi de aproximadamente 1 segundo. Tendo em conta os dados obtidos para a avaliação da operação mais pesada sobre a base de dados (ver Figura 5.3), é-nos possível dizer que a inserção das 1000 fotografias dos 40% de alterações foi de aproximadamente 760 milissegundos (76% do tempo total), o que corrobora os resultados obtidos para a inserção de 1000 fotografias na primeira utilização da aplicação — aproximadamente 800 milissegundos (ver Figura 5.1b).

Assim, é possível concluir que a percentagem de alterações efetuadas sobre um determinado conjunto de fotografias não tem grande impacto, uma vez que quando comparadas 40% de alterações (quase metade do valor total) com 0% de alterações não existe nenhum aumento significativo no tempo total de abertura da aplicação.

5.2 Perfil de Consumo de Memória

Numa aplicação móvel, ao lidar com um número elevado de fotografias, entre as quais fotografias que podem ter resoluções grandes, é necessário ter bastante atenção à memória utilizada. Caso contrário, incorremos no risco de esgotar a memória disponível (OOM).

Um OOM acontece quando a aplicação tenta alocar um novo objeto na memória e a memória já está no limite de espaço disponível para armazenamento de dados[4]. Normalmente, quando a aplicação tenta carregar demasiados *bitmaps* ou *bitmaps* demasiado grandes para memória sem que exista qualquer libertação do espaço ocupado, pode ser lançada a exceção de OOM. A abordagem mais comum para resolver este problema é fazer uma continua gestão de todos os recursos consumidos, de forma a libertar os recursos em memória quando estes deixem de ser necessários/utilizados. Esta abordagem, apesar de aparentemente simples, é, na verdade, bastante complexa, dada a dificuldade inerente a uma gestão de recursos correta e eficiente. Por esse motivo, a melhor solução é utilizar uma biblioteca que implemente essa gestão, minimizando a possibilidade de erros associados.

Embora não diretamente relacionado com a memória, outro erro comum em aplicações que lidam com *bitmaps* de grandes dimensões e, em particular, aplicações que efetuem cálculos com *bitmaps* grandes, é o *Application Not Responding* (ANR). Um ANR acontece quando a aplicação fica mais de 5 segundos sem responder a um *input* do utilizador (por exemplo, um toque no ecrã)[5]. A ocorrência de um ANR não tem uma causa específica, dado que pode ocorrer devido a vários fatores. No entanto, existem alguns padrões para tentar diagnosticar a causa do problema [51], como por exemplo:

- A aplicação estar a efetuar operações lentas de *Input/Output* na *thread* principal[6];
- A aplicação estar a efetuar cálculos pesados na *thread* principal;
- A *thread* principal estar à espera de uma operação pesada que está a executar numa outra *thread*.

[4] Cada aplicação tem um limite de memória que pode solicitar ao Sistema Operativo. Este limite difere de dispositivo para dispositivo.

[5] Sempre que um ANR ocorre, o sistema operativo do dispositivo lança uma mensagem ao utilizador a perguntar se este pretende aguardar que a aplicação volte a responder ou se prefere terminar por completo a execução da aplicação.

[6] A *thread* principal é a responsável por todas as atualizações na interface do utilizador e pelo processamento de todos os *inputs* do utilizador.

A principal abordagem para evitar um ANR é executar os cálculos pesados em *threads* de segundo plano, fornecendo *listeners* para serem chamados na *thread* principal quando o processamento estiver concluído.

Recapitulando o objetivo desta secção, a avaliação do consumo de memória, a aplicação *photo | uniq* tem dois grandes momentos de maior desafio para a gestão da memória:

1. Apresentação de uma Lista de Fotografias (por exemplo, na vista de Galeria Local);
2. Apresentação de Fotografias em *Fullscreen* (por exemplo, na vista de Pré-Visualização de Fotografias).

Para cada um destes casos, é ainda necessário garantir que a aplicação funciona corretamente e sem demasiadas demoras, independentemente da resolução das fotografias.

5.2.1 Apresentação de uma Lista de Fotografias

Quando a aplicação *photo | uniq* inicia, após a sincronização das bases de dados e inicialização da informação volátil, segue-se o carregamento das fotografias existentes na base de dados da aplicação para a lista de fotografias da vista Galeria Local. Neste caso, foram testadas duas formas de carregamento para as fotografias:

1. Carregamento manual, sem gerar miniaturas;
2. Carregamento utilizando a biblioteca *Glide*.

Figura 5.5: Consumo de memória durante a apresentação da Galeria Local, efetuando carregamento manual e terminando com um OOM (captura de ecrã do *Android Profiler*).

Para avaliar estas formas de carregamento de fotografias, utilizámos uma ferramenta disponibilizada pelo *Android Studio*, o *Android Profiler*. O *Android Profiler* é uma ferramenta que permite avaliar o desempenho de uma aplicação, fornecendo mecanismos que

permite identificar onde a aplicação está a utilizar ineficientemente recursos, como o CPU, a memória, os gráficos, a rede e a bateria do dispositivo[52].

Quando a aplicação faz o carregamento manual das fotografias — sem gerar miniaturas — a memória utilizada vai aumentando gradualmente até ao ponto de disparar um OOM (ver Figura 5.5). A Figura 5.5 apresenta uma captura de ecrã do *Android Profiler*, onde é visível uma linha de tempo detalhada do uso de memória da aplicação. Na linha de tempo, mais propriamente poucos segundos após o minuto 1:30 (visível na zona inferior da captura de ecrã), a aplicação tem uma subida acentuada de memória alocada, chegando a um total de 1.5Gb por volta do minuto 1:37. Esta subida foi consequência de um *scroll* sobre a lista de fotografias da Galeria Local (assinalado com um circulo cor de rosa no topo da captura de ecrã). Nos segundos após o pico de memória alocada, vê-se uma descida drástica da memória, sendo esta descida consequência de um OOM. Neste caso, o OOM foi resultado de uma continua alocação de fotografias com resolução grande (> 3Mb cada) em memória, seguido de um gesto de *scroll* que despoletou a necessidade de alocação de mais objetos (fotografias) em memória, de modo a poder apresentar as novas fotografias no ecrã. Para solucionar este problema utilizámos uma ferramenta de carregamento de imagens mais sofisticada, o *Glide* (apresentado na Secção 4.7.1). Com esta biblioteca já não nos deparamos com o problema de OOM, visto a biblioteca fazer toda a gestão de recursos, criação automática de miniaturas (que por sua vez diminuem bastante a memória utilizada) e, ainda, disponibilizando *caching* de imagens.

Na Figura 5.6 é possível ver uma nova captura de ecrã, nas mesmas circunstâncias que a captura anterior, mas utilizando a biblioteca *Glide* e a *cache* por esta disponibilizada. Nesta nova linha de tempo, é possível ver que a memória utilizada nem chega a 200Mb e, portanto, está bastante longe de OOM. Além da resolução do OOM, outra grande vantagem na utilização desta biblioteca é a velocidade de carregamento das imagens (ver Secção 5.3), uma vez que é bastante mais eficiente e rápida que o carregamento manual.

5.2.2 Apresentação de Fotografias em *Fullscreen*

Sempre que um utilizador clica numa fotografia da vista Galeria Local é redirecionado para o ecrã de Pré-Visualização de Fotografias. Quando este ecrã abre, a fotografia é carregada na sua totalidade (*fullscreen*) para o *container*. Nesta vista, as imagens têm de ser carregadas na sua totalidade para o *container*, visto o propósito da vista ser permitir ao utilizador a visualização total e em detalhe da fotografia, habilitando gestos de *zoom* e *panning* sobre a fotografia. Tal como para a apresentação de uma lista de fotografias, utilizámos o *Android Profiler* para medir o consumo de memória enquanto a vista de pré-visualização é utilizada.

Na Figura 5.7 é apresentado o consumo de memória durante a utilização da vista de Pré-Visualização de Fotografias. Nesta nova linha de tempo, as fotografias foram carregadas de forma manual e o teste consistiu em alternar a visualização *fullscreen* para

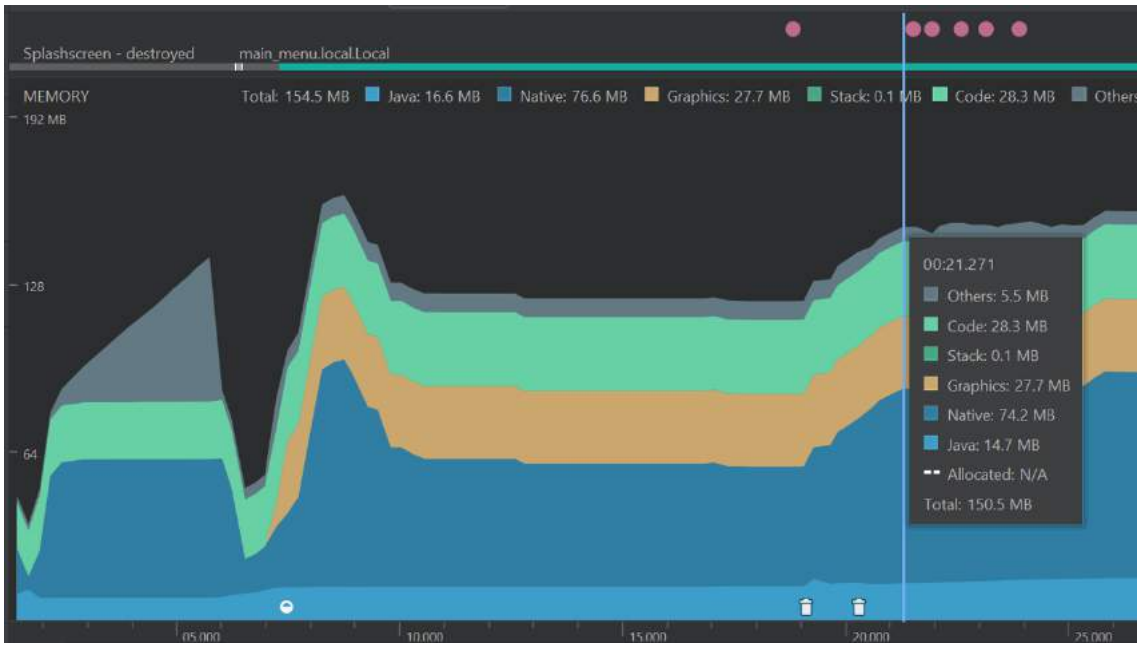


Figura 5.6: Consumo de memória durante a apresentação da Galeria Local, utilizando a biblioteca *Glide* (captura de ecrã do *Android Profiler*).

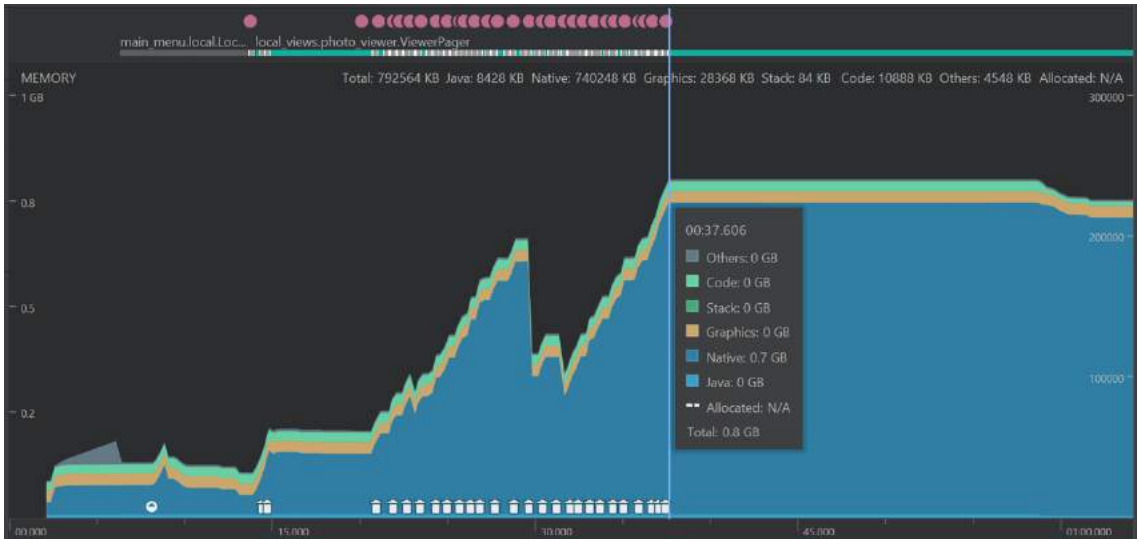


Figura 5.7: Consumo de memória durante a utilização da vista de Pré-Visualização de Fotografias, efetuando carregamento manual (captura de ecrã do *Android Profiler*).

um conjunto de fotografias. No intervalo [0,30] segundos, a memória alocada vai aumentando em função dos gestos efetuados (alternar a visualização *fullscreen*). Por volta dos 30 segundos é visível uma descida da memória alocada, proveniente da libertação de objetos (fotografias) pelo *garbage collector*[7]. No intervalo [33,37] segundos, a memória alocada volta a subir, ainda coincidente com gestos efetuados sobre a vista. No restante intervalo ([38,60[segundos), a memória mantém-se constante nos 0.8Gb, sendo este valor constante consequência de um ANR. Neste caso, o ANR teve como provável causa a execução consecutiva do código de carregamento manual das fotografias, um algoritmo bastante *naïve* que se torna pesado, que acaba por sobrecarregar o dispositivo e bloquear a aplicação[8].

Figura 5.8: Consumo de memória durante a utilização da vista de Pré-Visualização de Fotografias, utilizando a biblioteca *Glide* (captura de ecrã do *Android Profiler*).

Na Figura 5.8 é apresentado o consumo de memória durante a utilização da vista de Pré-Visualização de Fotografias, utilizando a biblioteca *Glide* para efetuar o carregamento de fotografias e gestão de recursos utilizados. Tal como para o carregamento manual, este teste consistiu em alternar a visualização *fullscreen* para um conjunto de fotografias. No intervalo [0, 64] segundos é possível ver que, na generalidade, a memória alocada vai aumentando, também em consequência dos gestos. Após o minuto 1.07, momento em que a aplicação já ultrapassou os 1.1Gb de memória alocada, a aplicação volta a sofrer um ANR, ficando bloqueada no último gesto efetuado. Contrariamente ao esperado, utilizando a biblioteca *Glide* também foi possível bloquear a aplicação, significando isto que a causa do ANR pode não estar diretamente relacionada com a gestão de recursos. No entanto, é também possível ver que o número de gestos necessários para lançar um ANR tem de ser bastante mais frequente num curto intervalo de tempo e durante bastante mais tempo na sua totalidade quando em relação com o carregamento manual. Este não é o

[7] O *garbage collector* consiste em recuperar memória inutilizada pelo programa/aplicação.
[8] Para apurar a causa em concreto do ANR, terão de ser efetuados outro tipo de testes exaustivos.

cenário de utilização ideal, visto existir algo que faz com que a aplicação sofra ANRs, mas, dadas as circunstâncias necessárias para que esse erro ocorra, podemos dizer que é "erro aceitável" para já[9], uma vez que a probabilidade de serem executados tantos gestos, num espaço curto de tempo, é bastante baixa. Portanto, assim se conclui que a utilização da biblioteca *Glide* continua a ser vantajosa sobre o carregamento manual.

5.3 Desempenho

Outro caso interessante de estudo para a validação da aplicação *photo | uniq* é o desempenho da mesma nos casos de uso da perfilagem do consumo de memória. Neste caso, os testes efetuados mediam o tempo de carregamento das miniaturas de fotografias para a lista da Galeria Local e o tempo de carregamento das fotografias em modo *fullscreen* para a vista de Pré-Visualização de Fotografias. Para ambos os casos de uso, testámos os seguintes tipos de carregamentos:

1. Carregamento manual, sem gerar miniaturas e sem utilizar qualquer tipo de *caching* de imagens;
2. Carregamento utilizando a biblioteca *Glide*, com a *cache* desligada;
3. Carregamento utilizando a biblioteca *Glide*, com a *cache* ligada.

Carregamento de Miniaturas de Fotografias para a Lista da Galeria Local

No gráfico da Figura 5.9, é apresentado o fator de aceleração para o carregamento de miniaturas de fotografia para a vista de Galeria Local, utilizando a biblioteca *Glide*. De forma geral verifica-se, a utilização da bibilioteca *Glide* é bastante benéfica, uma vez que ultrapassa a *baseline* (carregamento manual) na generalidade dos casos de uso: *glide* com e sem *cache*; fotografias com resoluções médias e grandes. Caso optássemos por não fazer *caching* de fotografias teríamos, na mesma, benefícios ao utilizar a biblioteca, uma vez que para imagens pequenas (< 1Mb) seria quase 5 vezes mais rápido que um carregamento manual e, para imagens maiores (> 3Mb) seria 6.5 vezes mais rápido. Já no caso da utilização da *cache*, é possível ver que o carregamento de imagens se torna quase 70 vezes mais rápido para imagens com tamanho superior a 3Mb, o que dadas as novas tecnologias, tende para ser o caso com maior probabilidade de acontecer. Mesmo no caso de imagens com resoluções mais pequenas, é possível fazer a um carregamento 17 vezes mais rápido que manualmente. Estes valores devem-se ao facto do carregamento manual ser bastante *naïve*, não gerando sequer uma miniatura para as fotografias (carregando-as completamente para a vista), nem tendo qualquer tipo de *cache* para as imagens já carregadas anteriormente. Assim, conclui-se que o carregamento manual não poderia ser

[9] Como trabalho futuro, deveremos tentar apurar as causas do ANR para que depois possamos corrigir o erro.

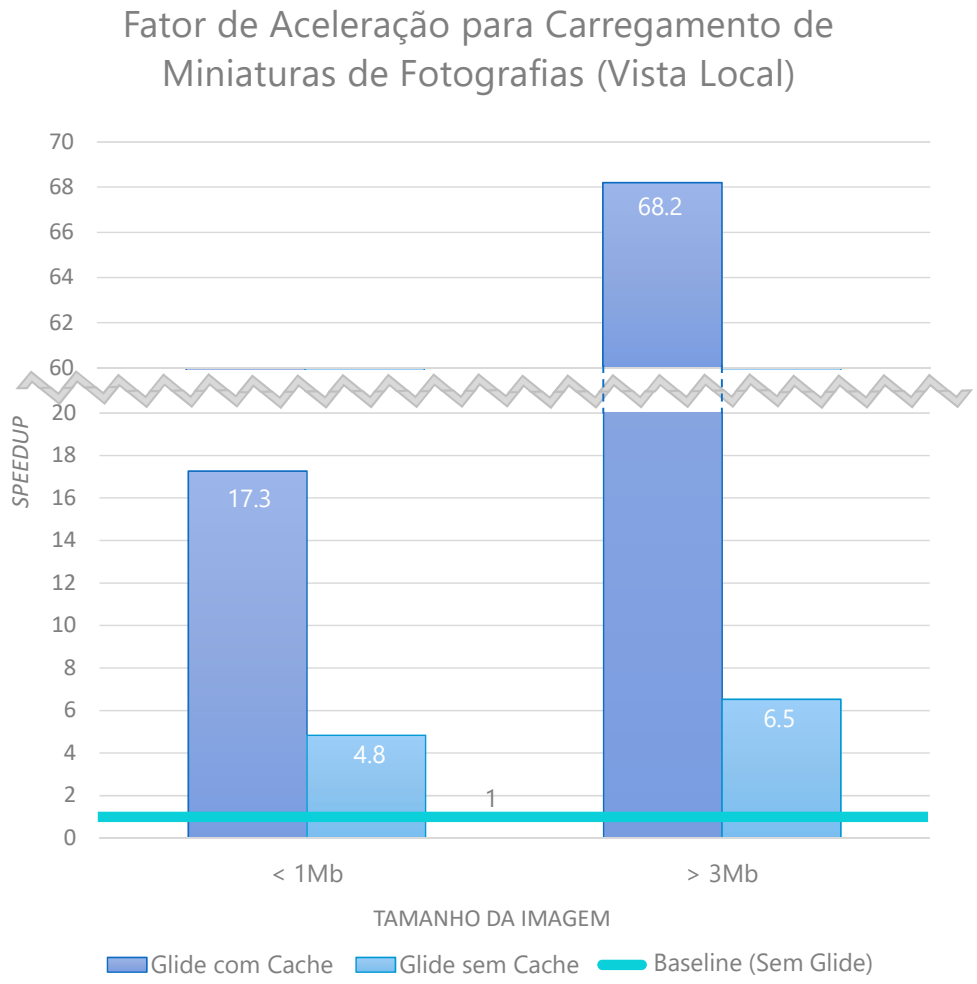


Figura 5.9: Quociente de carregamento de uma miniatura de fotografia para a vista de Galeria Local, no Samsung Galaxy S8 - Tabela A.6.

considerado como uma alternativa viável, pois, para além da salvaguarda dos OOMs, no pior caso, a utilização do *Glide* já só gasta 1/4 do tempo total.

Carregamento *Fullscreen* de Fotografias para a vista de Pré-Visualização

No gráfico da Figura 5.10, é apresentado o fator de aceleração para o carregamento de uma fotografia em *fullscreen* para a vista de Pré-Visualização de Fotografias. Contrariamente ao teste anterior, onde apenas era necessário carregar uma miniatura da imagem para a lista, nesta vista de pré-visualização, se a imagem não fosse carregada na sua totalidade para o *container*, o gesto de *zoom in* poderia levar a que o utilizador visualizasse (desnecessariamente) uma versão desfocada da imagem. Posta esta exigência de carregamento total da fotografia para o *container*, este seria considerado o cenário mais perigoso para o OOM, visto ser necessário lidar com a imagem na sua resolução original. Ao termos optado pela utilização da biblioteca *Glide*, o perigo foi reduzido ou mesmo eliminado, uma vez que a biblioteca trata de libertar todos os *bitmaps* quando deixam de ser necessários, prevenindo os OOMs.

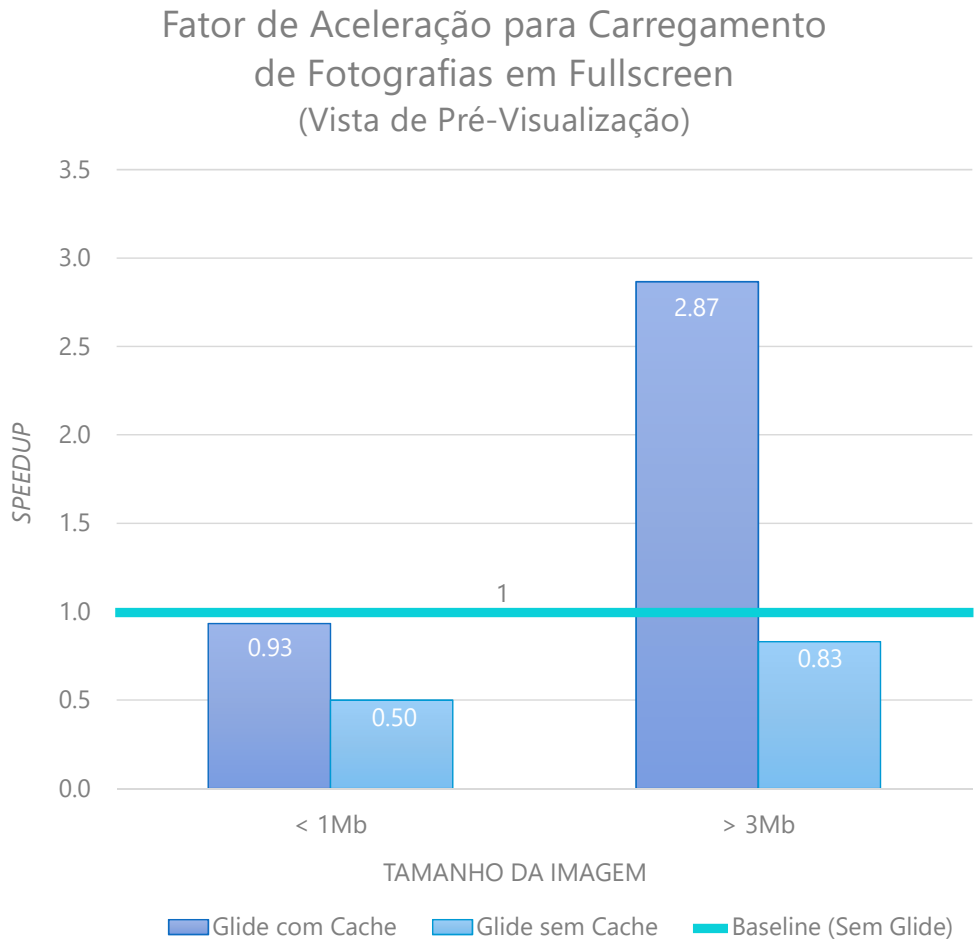


Figura 5.10: Fator de aceleração para o carregamento de fotografias em *fullscreen* na vista de Pré-Visualização de Fotografias, no Samsung Galaxy S8 - Tabela A.7.

Tal como para o carregamento de fotografias para a lista da vista de Galeria Local, no gráfico da Figura 5.10 efetuámos o fator de aceleração entre o carregamento manual (*baseline*) e o carregamento com a biblioteca *Glide* (com e sem *cache*), ambos para imagens com tamanho inferior a 1Mb e superior a 3Mb. Neste gráfico, o *speedup* já não é tão acentuado visto termos que lidar com fotografias na sua resolução original. Ainda assim, apenas o carregamento com *Glide* sem *cache* de uma imagem pequena é que se torna 2 vezes mais lento. Para imagens pequenas, utilizando *Glide* com *cache*, e para imagens grandes também utilizando Glide, mas sem *cache*, o tempo de carregamento fica bastante próximo do carregamento manual. Por fim, para imagens grandes, utilizar a biblioteca *Glide* com *cache* de imagens, torna-se quase 3 vezes mais rápido. Assim, é possível concluir, novamente, que a utilização da biblioteca *Glide* com *cache* se torna uma mais valia pois é onde se obtêm melhores resultados e a prevenção de OOM.

Capítulo 6

Conclusão e Trabalho Futuro

6.1 Conclusão

Nesta dissertação apresentou-se uma proposta de arquitetura composta por três módulos (servidor, plataformas disponíveis e gestores de fotografias), projetada e estudada para que possam ser suportadas várias plataformas e gestores de fotografias, e implementou-se um protótipo de uma aplicação Android como plataforma de interação com o utilizador na resolução do problema.

Durante a conceção da arquitetura, apesar de não terem sido implementados todos os componentes, esta foi pensada para suportar múltiplos dispositivos e muitos dos gestores de fotografias externos existentes. Outra característica importante desta arquitetura é o facto do modelo de interação permitir exportar para um servidor externo muitas das operações pesadas, para além do facto do armazenamento de todo o estado das instâncias de utilização de cada utilizador, permitindo que entre instalações da aplicação nada se perca.

Já relativamente à conceção da interface, são visíveis as semelhanças entre os ecrãs da maqueta e o produto final implementado. Tudo isto demonstra que houve um trabalho de estudo e estruturação de toda a aplicação, de modo a garantir que no momento da implementação muitos dos detalhes importantes das funcionalidades estavam já previamente pensados/planeados. No caso da implementação da interface, a estrutura apresentada para a organização/acesso aos dados teve um papel importante na performance da aplicação, uma vez que muitos dos dados tinham de ser acedidos rápida e frequentemente. O repetitivo acesso à base de dados para retorno dos dados não era viável por ser bastante moroso.

Por fim, resta mencionar a importância da utilização de bibliotecas auxiliares. Estas bibliotecas, apesar de não serem fundamentais, facilitam bastante o trabalho, podendo todo

o tempo e custo de uma implementação de raiz das funcionalidades por estas fornecidas ser aproveitado para outras tarefas.

6.2 Trabalho Futuro

Como melhorias para o futuro, um dos primeiros passos poderia ser o afinar de todo o processo de interação com o servidor, uma vez que o servidor ainda não se encontra finalizado. O período de espera pelo término do servidor, poderia ser também utilizado para detetar a fonte do problema dos ANRs e desenvolver as funcionalidades assinaladas que não foram implementadas.

De seguida poderíamos estudar das alterações necessárias para suportar a sincronização multi-dispositivos, nomeadamente para endereçar quais as melhores estratégias para resolução de conflitos. Estudada e implementada a sincronização multi-dispositivos, penso que o próximo passo poderá ser a validação de qual a melhor forma para integrar os gestores de fotografias já existentes no mercado, para além da avaliação de qual o melhor local para fazer a integração (se do lado da aplicação ou se diretamente no servidor).

Feitas todas estas melhorias, o último passo seria a avaliação/apreciação da interface e experiência de utilização, podendo ser utilizadas técnicas e metodologias *standard*.

Bibliografia

[1] *Top Devices of 2017 on Flickr | Flickr Blog*. URL: http://blog.flickr.net/en/2017/12/07/top-devices-of-2017/ (acedido em 22/01/2018).

[2] *Musei Capitolini*. URL: http://www.museicapitolini.org/en/ (acedido em 01/03/2019).

[3] A. Alves. "Automatic Best Photo Selection". Tese de mestrado. Caparica: Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2015.

[4] C. Machado. "Suporte Aplicational de um Workflow para Selecção Assistida de Fotografias". Tese de mestrado. Caparica: Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2017.

[5] *7 Reasons Why Your Business Needs a Mobile App | AllBusiness.com*. URL: https://www.allbusiness.com/7-reasons-business-needs-mobile-app-19179-1.html (acedido em 29/01/2018).

[6] *Projetar | Android Developers*. URL: https://developer.android.com/design/index.html (acedido em 14/01/2018).

[7] *Themes - Overview - iOS Human Interface Guidelines*. URL: https://developer.apple.com/ios/human-interface-guidelines/overview/themes/ (acedido em 14/01/2018).

[8] Archiveddocs. *Design library for Windows Phone*. en-us. URL: https://docs.microsoft.com/en-us/previous-versions/windows/apps/hh202915(v%3dvs.105) (acedido em 14/01/2018).

[9] *An Overview of Different Stages of Mobile App Development Lifecycle*. Inglês. Jun. de 2017. URL: https://www.mindinventory.com/blog/an-overview-of-different-stages-of-mobile-app-development-lifecycle/ (acedido em 17/01/2018).

[10] *8 Stages of Effective Mobile App Development*. en-US. Nov. de 2016. URL: https://www.dogmasystems.com/8-stages-effective-mobile-app-development/ (acedido em 17/01/2018).

[11] *What is HCI (human-computer interaction)? - Definition from WhatIs.com*. en. URL: http://searchsoftwarequality.techtarget.com/definition/HCI-human-computer-interaction (acedido em 17/01/2018).

[12] J. Nielsen. *Usability Engineering*. 1st. Morgan Kaufmann, 1993, pp. 36 –32. ISBN: 0125184069,9780125184069.

[13] J. J. Garrett. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. 2010, p. 240. ISBN: 9780321683687. URL: http://books.google.com/books?hl=en{\&}lr={\&}id=9QC6r5OzCpUC{\&}pgis=1.

[14] *What is User Experience (UX) Design? | Interaction Design Foundation*. URL: https://www.interaction-design.org/literature/topics/ux-design (acedido em 18/01/2018).

[15] *About Peter Morville*. URL: http://semanticstudios.com/about/ (acedido em 18/01/2018).

[16] *User Experience Design*. URL: http://semanticstudios.com/user_experience_design/ (acedido em 18/01/2018).

[17] *Usability: A part of the User Experience | Interaction Design Foundation*. URL: https://www.interaction-design.org/literature/article/usability-a-part-of-the-user-experience (acedido em 18/01/2018).

[18] *56e960696b7cf.jpg (1500×500)*. URL: https://public-media.interaction-design.org/images/ux-daily/56e960696b7cf.jpg (acedido em 24/01/2018).

[19] *Introduction to the Mobile Software Development Lifecycle - Xamarin*. URL: https://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_sdlc/ (acedido em 14/01/2018).

[20] *Compatibilidade com várias telas | Android Developers*. URL: https://developer.android.com/guide/practices/screens{_}support.html (acedido em 19/01/2018).

[21] A. Ionescu. "Resource Management in Mobile Cloud Computing". Em: *Informatica Economică* 19.1 (2015). DOI: 10.12948/issn14531305/19.1.2015.05. URL: http://revistaie.ase.ro/content/73/05-Ionescu.pdf.

[22] H. Höpfner, S. Wendland e E. Mansour. "DATA CACHING ON MOBILE DEVICES The Experimental MyMIDP Caching Framework". Em: (). URL: https://pdfs.semanticscholar.org/3719/c93d8fc5efb4dcdb033af3fa8bd22f6e0370.pdf.

[23] *Google Play*. URL: https://play.google.com/store (acedido em 19/01/2018).

[24] *iTunes - Browse the top free apps on the App Store - Apple*. URL: https://www.apple.com/itunes/charts/free-apps/ (acedido em 19/01/2018).

[25] *Aplicações Windows Phone - Loja Microsoft*. URL: https://www.microsoft.com/pt-pt/store/apps/windows-phone?rtc=1 (acedido em 19/01/2018).

[26] *Remo Duplicate Photos Remover na App Store*. URL: https://itunes.apple.com/pt/app/remo-duplicate-photos-remover/id1066797785?mt=8 (acedido em 03/01/2018).

[27] *GetSpace: Delete Duplicate Photo in Device Storage na App Store*. URL: https://itunes.apple.com/pt/app/getspace-delete-duplicate-photo-in-device-storage/id1055204786?mt=8 (acedido em 03/01/2018).

[28] *Dr. Clean na App Store*. URL: https://itunes.apple.com/pt/app/dr-clean/id1156773866?mt=8 (acedido em 03/01/2018).

[29] *PhotoSweeper*. URL: https://itunes.apple.com/pt/app/photosweeper/id463362050?mt=12 (acedido em 12/02/2018).

[30] *Best Photos na App Store*. URL: https://itunes.apple.com/pt/app/best-photos/id1168605247?mt=8 (acedido em 03/01/2018).

[31] *X-Dups - Remove duplicate and similar photos*. URL: https://itunes.apple.com/us/app/x-dups-remove-duplicate-and-similar-photos/id1123212211?mt=8 (acedido em 02/01/2018).

[32] *Duplicate Photos Fixer on the App Store*. URL: https://itunes.apple.com/us/app/duplicate-photos-fixer/id966475595?mt=8 (acedido em 02/01/2018).

[33] *Remove master for camera roll na App Store*. URL: https://itunes.apple.com/pt/app/remove-master-for-camera-roll/id792628362?mt=8 (acedido em 03/01/2018).

[34] *Duplicate Photos Remover*. URL: https://play.google.com/store/apps/details?id=com.tweaking.duplicatephotofixer (acedido em 03/01/2018).

[35] *Phone Cleaner - Free up Storage on your phone na App Store*. URL: https://itunes.apple.com/pt/app/phone-cleaner-free-up-storage-on-your-phone/id1092614777?mt=8 (acedido em 03/01/2018).

[36] *OpenCV library*. URL: https://opencv.org/ (acedido em 31/01/2018).

[37] *The Open Incubator Model: Entrepreneurship, Open Innovation, and Economic ... - Ilan Bijaoui - Google Livros*. URL: https://books.google.pt/books?id=BlWxCgAAQBAJ{\&}pg=PT66{\&}lpg=PT66{\&}dq=smarter+mobile+devices+that+are+more+aware+of+their+owner{\%}27s+preferences+and+location{\&}source=bl{\&}ots=e7fPi9JkwN{\&}sig=Ctjzdo5LOSks4EG2iRlPoBo1OL0{\&}hl=pt-PT{\&}sa=X{\&}ved=2ahUKEwiU7-TpnuvfAhUKExQKHW (acedido em 26/01/2019).

[38] *Industry Leaders Announce Open Platform for Mobile Devices | Open Handset Alliance*. URL: http://www.openhandsetalliance.com/press{_}110507.html (acedido em 26/01/2019).

[39] *The history of Android OS: its name, origin and more*. URL: https://www.androidauthority.com/history-android-os-name-789433/ (acedido em 26/01/2019).

[40] *Mobile Operating System Market Share Worldwide | StatCounter Global Stats*. URL: http://gs.statcounter.com/os-market-share/mobile/worldwide (acedido em 26/01/2019).

[41] *Conheça o Android Studio*. URL: https://developer.android.com/studio/intro/ (acedido em 20/01/2019).

[42] *App startup time*. URL: https://developer.android.com/topic/performance/vitals/launch-time (acedido em 18/03/2019).

[43] *Android - Facebook Login*. URL: https://developers.facebook.com/docs/facebook-login/android/ (acedido em 21/03/2019).

[44] *Autenticar usando o Login do Google no Android | Firebase*. URL: https://firebase.google.com/docs/auth/android/google-signin (acedido em 21/03/2019).

[45] *Facebook Login for Android - Quickstart*. URL: https://developers.facebook.com/docs/facebook-login/android (acedido em 18/03/2019).

[46] *About Glide*. URL: https://bumptech.github.io/glide/ (acedido em 01/03/2019).

[47] *Retrofit*. URL: https://square.github.io/retrofit/ (acedido em 01/03/2019).

[48] *Bearer Authentication*. URL: https://swagger.io/docs/specification/authentication/bearer-authentication/ (acedido em 09/06/2019).

[49] *Galaxy S8 & S8+ Specs | Especificações Técnicas | Samsung Portugal*. URL: https://www.samsung.com/pt/smartphones/galaxy-s8/spec-plus/ (acedido em 19/05/2019).

[50] *Wiko Mobile - HIGHWAY STAR*. URL: https://world.wikomobile.com/m543-highway-star (acedido em 19/05/2019).

[51] *ANRs*. URL: https://developer.android.com/topic/performance/vitals/anr (acedido em 15/05/2019).

[52] *Profile your app performance*. URL: https://developer.android.com/studio/profile (acedido em 19/05/2019).

Apêndice A

Apêndice A : Dados recolhidos para a Validação

Este apêndice apresenta todos os dados recolhidos para a validação da aplicação *photo | uniq*.

Tabela A.1: Tempo de abertura da primeira utilização da aplicação no Wiko Highway Star 4G.

Número de Fotografias	Obtenção do maxID da base de dados da aplicação	Iteração pelo conjunto de fotografias do dispositivo	Atualização da base de dados da aplicação	Obtenção das fotografias da base de dados da aplicação	Preenchimento da informação volátil
0	1.333	36.333	71.333	2.333	0.000
1000	1.333	233.000	323.000	266.333	383.667
2000	1.333	360.000	543.000	531.667	759.000
3000	1.333	445.000	768.667	815.667	1149.667
4000	1.333	524.667	966.000	1097.667	1540.333
5000	1.333	617.667	1191.333	1372.333	1871.333
6000	1.333	702.000	1493.333	1658.000	3779.333
7000	1.333	791.333	1622.000	1905.667	4201.333
8000	1.333	882.000	1847.000	2195.333	4615.333
9000	1.333	961.000	2218.000	2573.333	4967.000
10000	1.333	1059.667	2470.667	2874.333	5358.667
11000	1.333	1168.667	2497.333	3192.333	5769.333
12000	1.333	1273.000	2717.667	3453.333	6290.000

Tabela A.2: Tempo de abertura da primeira utilização da aplicação no Samsung Galaxy S8.

Número de Fotografias	Obtenção do maxID da base de dados da aplicação	Iteração pelo conjunto de fotografias do dispositivo	Atualização da base de dados da aplicação	Obtenção das fotografias da base de dados da aplicação	Preenchimento da informação volátil
0	2.000	55.667	59.667	3.333	0.000
1000	2.333	132.000	231.000	59.667	307.333
2000	2.000	141.667	267.000	76.333	302.667
3000	2.000	156.667	328.333	103.667	454.000
4000	2.000	162.000	393.000	132.667	558.667
5000	2.000	175.667	466.000	161.000	669.667
6000	2.000	187.000	507.667	188.667	774.000
7000	2.000	209.333	586.333	213.000	1153.333
8000	2.000	208.000	637.333	239.333	1186.667
9000	2.000	234.333	707.667	267.667	1238.000
10000	2.000	231.667	760.000	308.333	1301.333
11000	2.000	243.000	813.667	320.667	1370.000
12000	2.000	252.667	863.000	346.333	1451.667

Tabela A.3: Tempo de abertura da primeira utilização da aplicação para ambos os dispositivos.

Número de Fotografias	**Tempo Total no Wiko Highway Star 4G**	**Tempo Total no Samsung Galaxy S8**
0	111.333	120.667
1000	1207.333	732.333
2000	2195.000	789.667
3000	3180.333	1044.667
4000	4130.000	1248.333
5000	5054.000	1474.333
6000	7634.000	1659.333
7000	8521.667	2164.000
8000	9541.000	2273.333
9000	10720.667	2449.667
10000	11764.667	2603.333
11000	12629.000	2749.333
12000	13735.333	2915.667

Tabela A.4: Tempo de inserção de fotografias vs remoção de fotografias, no Samsung Galaxy S8.

Ação	Tempo
Inserção de 50% de Fotografias	211.000
Remoção de 50% de Fotografias	52.333

Tabela A.5: Tempo de abertura da aplicação em função de alterações sobre a "Galeria Local", no Samsung Galaxy S8.

Percentagem de Alterações	Obtenção do maxID da base de dados da aplicação	Iteração pelo conjunto de fotografias do dispositivo	Atualização da base de dados da aplicação	Obtenção das fotografias da base de dados da aplicação	Preenchimento da informação volátil
0	2.333	62.000	39.000	175.667	605.000
5	2.333	62.000	88.667	144.000	643.000
10	2.667	68.333	113.333	161.000	582.000
20	2.000	79.000	129.333	148.667	575.667
40	2.000	82.333	164.667	137.667	583.000

Tabela A.6: Fator de aceleração para o carregamento de miniaturas de fotografias para a lista da vista de Galeria Local, no Samsung Galaxy S8.

Tamanho da Fotografia	Glide com Cache	Glide sem Cache
<1Mb	17.265	4.837
>3Mb	68.197	6.533

Tabela A.7: Fator de aceleração para o carregamento de fotografias em *fullscreen* na vista de Pré-Visualização de Fotografias, no Samsung Galaxy S8.

Tamanho da Fotografia	Glide com Cache	Glide sem Cache
<1Mb	0.933	0.500
>3Mb	2.866	0.831

Apêndice B

Apêndice B : Lista completa de melhorias e *bugs* encontrados

Este apêndice apresenta a lista completa de *bugs* encontrados até ao momento na aplicação *photo | uniq*.

Title	T	P	Status	Votes	Assignee	Created
#47: Upload Fotos			NEW		Mara Felismino	2019-01-19
#34: Remoção de Fotografias da Galeria Nativa			NEW		Mara Felismino	2018-09-18
#45: Uploading Service (MODO PAUSA)			NEW		Mara Felismino	2018-11-24
#43: Eliminar photos que estavam em upload			NEW		Mara Felismino	2018-09-25
#36: Lentidão ao Selecionar todas as Fotografias			NEW		Mara Felismino	2018-09-18
#27: Esconder (TODAS) as fotografias que estão em processo de envio (pausado ou não) faz com que o botão da nuvem desapareça			NEW		Mara Felismino	2018-07-28
#26: Pause > Remove > Play quando se está a fazer upload de fotos ainda não está 100% funcional			NEW		Mara Felismino	2018-07-28
#46: Vista de Comparação de 2 imagens: Pontos de teste desenhados sobre o Bitmap			NEW		Mara Felismino	2019-01-05
#8: Guardar o estado de seleção anterior			OPEN		Mara Felismino	2018-07-05
#42: Vistas de Ordenação do Servidor não têm scroll sincronizado			NEW		Mara Felismino	2018-09-18
#28: Colocar a label "all" no botão de selecionar todas as fotos na toolbar			NEW		Mara Felismino	2018-07-28
#9: Reposicionamento da Vista			NEW		Mara Felismino	2018-07-05

Figura B.1: Lista completa de melhorias a efetuar e de *bugs* encontrados.