UNIVERSIDADE FEDERAL DE SANTA CATARINA

Análise do desenvolvimento de aplicativos mobile nativos e multiplataforma

Juliana Silva Pinheiro

Florianópolis
2020

Juliana Silva Pinheiro

Análise do desenvolvimento de aplicativos mobile nativos e multiplataforma

Trabalho de conclusão de curso submetido como parte dos requisitos para obtenção do título de Bacharel, do curso de Ciências da Computação da Universidade Federal de Santa Catarina.

Orientador: Raul Sidnei Wazlawick

Florianópolis, Dezembro de 2020

Juliana Silva Pinheiro

Análise do desenvolvimento de aplicativos mobile nativos e multiplataforma

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de Bacharel em Ciências da Computação, e aprovado em sua forma final pelo Curso de Ciências da Computação da Universidade Federal de Santa Catarina.

Prof. Dr. Raul Sidnei Wazlawick
Orientador
Universidade Federal de Santa Catarina

Profa. Dra. Patrícia Vilain
Membro da Banca
Universidade Federal de Santa Catarina

Renan Luiz Arceno
Membro da Banca

Florianópolis, Novembro de 2020

You deserve to be here. You deserve to exist.

You deserve to take up space in this world of men.

Mackenzi Lee

Resumo

Atualmente, os sistemas operacionais Android e iOS dominam o mercado de dispositivos móveis. Como ambos possuem linguagem de programação nativa diferentes, desenvolver aplicações para os dois sistemas necessitaria de possivelmente o dobro de esforço e recursos humanos, e o dobro de código. Para evitar o retrabalho, surgem os *frameworks* e ferramentas de desenvolvimento multiplataforma. Com o auxílio destes, é possível desenvolver aplicações em uma única linguagem e utilizar o mesmo código para ambos os sistemas. Este trabalho teve como objetivo comparar o desenvolvimento nativo com o desenvolvimento multiplataforma utilizando o *framework* Flutter. Foram desenvolvidas e comparadas três versões de um aplicativo (nativas e multiplataforma), por critérios de performance, acesso a funcionalidades nativas, documentação e reuso de código. Apesar das versões nativas demonstrarem algumas vantagens de performance, o *framework* obteve bons resultados de tempo de resposta, além de possibilitar acesso as funcionalidades nativas, ter boa documentação e alto reuso de código.

Palavras-chave: desenvolvimento de aplicativos móveis, desenvolvimento multiplataforma, framework de desenvolvimento de aplicativos, Flutter.

Abstract

Nowadays, Android and iOS dominate the market of mobile operating systems. Since both have different native programming languages, developing applications for both systems could require twice the amount of effort and human resources, and twice the amount of code. To avoid that, multi-platform frameworks and development tools were created. With their help, it is possible to develop applications in a single language and use the same code base for both systems. This work aimed to compare native development with multi-platform development using the Flutter framework. Three versions of a mobile application (native and multi-platform) were developed and compared, based on performance criteria, access to native functionality, documentation and code reuse. Although the native versions demonstrate some performance advantages, the framework obtained good response times results, in addition to allowing access to native functionalities, having good documentation and high code reuse.

Keywords: mobile application development, multi-platform development, cross-platform development, mobile development framework, Flutter.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API *Application Programming Interface*. 25

BD *Banco de Dados*. 50

DSL *Domain-Specific Language*. 27

IDE *Integrated Development Environment*. 25

LLVM *Low Level Virtual Machine*. 43

MDD *Model Driven Development*. 27

NDK *Native Development Kit*. 43

SDK *Software Development Kit*. 21, 25

UML *Unified Modeling Language*. 27

Sumário

1 Introdução

O mercado de dispositivos móveis está em constante ascensão, com 3,2 bilhões de usuários mundialmente no final de 2019 [Statista 2020], e mais de 80% do tempo gasto nos dispositivos são com aplicativos, segundo pesquisa realizada pela Comscore 2018. Como resultado, é possível ver o crescimento também no mercado de aplicativos: o número de downloads de aplicativos móveis alcançou 204 bilhões no final de 2019 [Statista 2020].

Atualmente os sistemas operacionais Android[1] e iOS[2] detêm juntos 98,82% do mercado de dispositivos móveis (celulares e tablets) [StatCounter 2019], sendo assim o principal foco do desenvolvimento de aplicativos [SlashData 2019]. Entretanto, esses sistemas possuem cada um suas particularidades, além de linguagens de programação nativas e *Software Development Kits* (SDKs) diferentes, o que demanda que o desenvolvimento de aplicações para ambos os sistemas, necessário para alcançar a maior parte do público, seja feito separadamente. Isto é, se faz necessário desenvolver duas versões do mesmo aplicativo, em linguagens diferentes, duplicando a quantidade de esforço, custo e/ou tempo para o desenvolvimento. Além do desenvolvimento inicial, a manutenção de dois projetos de forma a mantê-los iguais (versionamento, mesmas funcionalidades novas, entre outros) também necessita de mais esforço comparado a apenas um projeto.

Como solução para essa questão surgiram *frameworks* e ferramentas de desenvolvimento multiplataforma, também referido como híbrido. Essas ferramentas tem como objetivo utilizar uma única base de código para todas as plataformas alvo, diminuindo o esforço necessário para desenvolver os aplicativos. Para alcançar esse objetivo, as ferramentas implementam métodos diferentes, explicitados no capítulo 3 deste trabalho.

Segundo o relatório *Developer Economics State of The Developer Nation 17th Edition* [SlashData 2019], numa pesquisa com 1.189 desenvolvedores *mobile*, 40% dos desenvolvedores profissionais responderam ter utilizado algum *framework* multiplataforma nos últimos 12 meses. Dentre os desenvolvedores que utilizaram alguma ferramenta multiplataforma, aproximadamente 1 em 4 disseram utilizar React Native[3]. Com pelo menos 10% das resposta cada, foram citados também Xamarin[4], Flutter[5] e Ionic[6], mostrando um cenário relativamente segmentado.

O surgimento de novas ferramentas, porém, naturalmente leva a comparações entre

[1] <https://www.android.com/>
[2] <https://www.apple.com/ios/>
[3] <https://facebook.github.io/react-native/>
[4] <https://dotnet.microsoft.com/apps/xamarin>
[5] <https://flutter.dev/>
[6] <https://ionicframework.com/>

elas, e entre o desenvolvimento multiplataforma e nativo. A Airbnb[7], empresa que fornece serviço online (*web* e *mobile*) de anúncio e reserva de acomodações e hospedagem a mais de 150 milhões de usuários [iPropertyManagement.com 2019], adotou o desenvolvimento multiplataforma em 2016, utilizando React Native, e reportou sua experiência com o *framework*. Após dois anos de desenvolvimento multiplataforma, a empresa se viu obrigada a optar por voltar ao desenvolvimento nativo, tanto por questões técnicas do React Native quanto por questões organizacionais que surgiram com o desenvolvimento multiplataforma. A equipe citou pontos positivos: 95%-100% de código compartilhado entre as plataformas, performance, uso de React[8], qualidade das animações, entre outros. Entretanto, a imaturidade do *framework*, dificuldade em atualizá-lo, em refatorar código e em adicionar acessibilidade ao aplicativo, o tamanho do aplicativo e o tempo de inicialização e renderização foram apenas alguns dos problemas encontrados durante os dois anos de uso [Peal 2018].

Em contraste, o Nubank[9], maior *fintech* da América Latina [Gagne 2019], cujos produtos (conta digital, cartão de crédito, entre outros) são totalmente controlados por meio do seu aplicativo, recentemente iniciou a adoção do Flutter, *framework* de desenvolvimento multiplataforma da Google, integrando-o à já existente base de código em React Native, Kotlin e Swift [Freire e Andrade 2019].

Flutter foi lançado recentemente, em 2017, e pode ser uma boa alternativa aos *frameworks* e ferramentas multiplataforma existentes. Neste trabalho será comparado o desenvolvimento multiplataforma utilizando esse *framework* com o desenvolvimento nativo, após ter sido identificada a falta de estudos publicados na revisão bibliográfica apresentada no capítulo 3.

1.1 Objetivos

O objetivo geral é comparar o método de desenvolvimento nativo com o desenvolvimento multiplataforma, implementando uma aplicação com versões nativas para iOS e Android e versão em Flutter.

Os objetivos específicos são (1) identificar os métodos de desenvolvimento para plataformas móveis (celulares e tablets), e seus desafios ou dificuldades, (2) definir e implementar uma aplicação para plataformas móveis com versões nativas para iOS e Android, e versão multiplataforma utilizando Flutter, (3) comparar as versões do aplicativo desenvolvidas pelas critérios de consumo de CPU, (4) consumo de memória, (5) tamanho da aplicação, (6) tempo de resposta, (7) acesso a funcionalidades nativas, (8) qualidade da documentação e (9) reuso de código.

[7] <https://www.airbnb.com/>
[8] <https://reactjs.org/>
[9] <https://nubank.com.br/>

2 Método de Pesquisa

Este trabalho seguiu as seguintes etapas:

1. Revisão bibliográfica com o objetivo de identificar os métodos de desenvolvimento de aplicativos multiplataforma e *frameworks* ou ferramentas nessa área ainda não abordados ou pouco abordados, e trabalhos correlatos que realizaram comparações entre os diferentes métodos de desenvolvimento.
2. Definição da aplicação a ser implementada para comparação do desenvolvimento nativo com o desenvolvimento multiplataforma.
3. Desenvolvimento de três versões do aplicação: nativa para iOS, nativa para Android e multiplataforma utilizando Flutter.
4. Comparação das versões por critérios de consumo de CPU, consumo de memória, tamanho da aplicação, tempo de resposta, acesso a funcionalidades nativas, qualidade da documentação e reuso de código.
5. Elaboração do relatório com os resultados encontrados.

3 Fundamentação Teórica

Neste capítulo são apresentados conceitos relevantes no desenvolvimento móvel, incluindo os métodos disponíveis, segundo a literatura lida.

3.1 iOS

iOS é um sistema operacional móvel criado e desenvolvido pela Apple exclusivo para dispositivos touchscreen da empresa. No mês de Novembro de 2019, a Apple Store, loja oficial do iOS, disponibilizava mais de 3 milhões de aplicativos [Statista 2019]. A linguagem de programação nativa do sistema é o Swift, desenvolvido também pela Apple para gradualmente substituir o uso de Objective-C, linguagem originalmente utilizada pela plataforma [Apple 2014]. Para o desenvolvimento de aplicativos, a Apple fornece o kit de desenvolvimento iOS SDK e a IDE Xcode, ambos exclusivos para o sistema operacional macOS, também da empresa.

3.2 Android

Android é um sistema operacional móvel de código aberto, focado em dispositivos touchscreen, como smartphones e tablets, que atualmente domina o mercado de sistemas operacionais [StatCounter 2019]. É desenvolvido pelo grupo Open Handset Alliance, cujo maior contribuidor é a Google. Durante o terceiro trimestre de 2019, a Google Play Store, loja do sistema operacional, disponibilizava mais de 2 milhões de aplicativos [Statista 2019]. O kit de desenvolvimento para Android, Android SDK, possibilita o uso das linguagens Java e Kotlin.

3.3 Desenvolvimento nativo

Aplicativos nativos são aqueles desenvolvidos utilizando a linguagem de programação específica do sistema operacional alvo [Smutný 2012]. Como citado anteriormente para os sistemas iOS e Android, cada sistema operacional móvel possui linguagens nativas, SDKs, APIs, IDEs e lojas diferentes, como pode ser visto na tabela 1. Essa característica cria a necessidade de desenvolver um mesmo aplicativo múltiplas vezes, uma vez para cada plataforma desejada, o que aumenta o custo, tempo e esforço de desenvolvimento para que um aplicativo novo seja disponibilizado em todas as lojas. Em contrapartida, aplicativos nativos têm melhor performance e total acesso a todas as funcionalidades do dispositivo, além de oferecer a interface de usuário nativa do sistema [El-Kassas et al. 2017].

	Linguagem de programação	SDK	IDE	Loja de aplicativos
iOS	Swift	iOS SDK	Xcode	App Store
Android	Java/Kotlin	Android SDK	Android Studio	Google Play Store

Tabela 1 – Principais diferenças no desenvolvimento para sistemas iOS e Android

3.4 Desenvolvimento multiplataforma

Segundo El-Kassas et al. 2017, o conceito das soluções multiplataforma, referenciadas tanto como *cross-platform* quanto como *multi-platform* em inglês, é desenvolver um aplicativo uma vez e executá-lo em qualquer sistema operacional móvel. Inicialmente, o desenvolvimento de aplicativos multiplataforma era impraticável devido as diferenças de linguagem e SDK, como citado na seção anterior. Para contornar esse problema, vários métodos e ferramentas de desenvolvimento foram criados, todos sob o conceito de desenvolvimento multiplataforma. Importante ressaltar que alguns trabalhos e publicações fora da academia se referem ao desenvolvimento multiplataforma como desenvolvimento híbrido, porém, nesse trabalho optou-se por utilizar a taxonomia sugerida por Biørn-Hansen, Grønli e Ghinea 2018. Os autores categorizam as ferramentas em quatro métodos: híbrido, interpretado, *cross-compiled* e orientado a modelos.

3.4.1 Híbrido

O método híbrido utiliza tecnologias web, incluindo HTML, CSS e JavaScript na implementação da interface de usuário e da lógica do aplicativo. Isso é possível pois as ferramentas que implementam esse método criam um projeto nativo, instanciam um *browser* por meio de um componente *WebView* e executam o código da aplicação no *browser*. O acesso a funcionalidades nativas (sistema de notificações, GPS, câmera, sistema de arquivos, entre outras) é limitado e feito por meio de uma camada de abstração [Latif et al. 2016] (geralmente referida como *plugin* [1] [2]) oferecidas pelo *framework* escolhido. Sendo assim, os aplicativos consistem da junção do projeto nativo da plataforma alvo com o código base comum às plataformas. As ferramentas mais populares são PhoneGap, Ionic Framework e OnsenUI [Latif et al. 2017].

3.4.2 Interpretado

O método interpretado permite o uso de linguagens genéricas como JavaScript por meio de interpretadores nos dispositivos. As ferramentas diferem do método híbrido por não necessitarem do componente *WebView*, e conseguem acessar as APIs nativas (e, consequentemente, as funcionalidades nativas, como notificações, câmera, entre outras) por

[1] <http://docs.phonegap.com/plugin-apis/>
[2] <https://ionicframework.com/docs/native/community>

meio de uma camada de abstração. Os *frameworks* mais populares são React Native [Biørn-Hansen et al. 2019], Appcelerator Titanium e NativeScript [Delia et al. 2019].

3.4.3 Cross-compiled

Nesse método, o código produzido na linguagem de programação escolhida pelo *framework* é compilado para código binário executável nativo de cada plataforma alvo, sem necessidade de camadas intermediárias como os métodos anteriores. O acesso às funcionalidades nativas nesse caso é feito por meio do SDK do *framework*. Por ser compilado para código binário nativo, a maior vantagem desse método sobre os anteriores é a performance [Latif et al. 2016]. O principal *framework* que implementa esse método é o Xamarin.

3.4.4 Orientado a modelos

O método orientado a modelos, *Model Driven Development* (MDD), se assemelha ao *cross-compiled* por usar geração automática de código nativo, porém partindo não de linguagens de programação mas de modelos da interface de usuário e da lógica de negócio descritos em UML ou outras linguagens *Domain-Specific Language* (DSL). Segundo Biørn-Hansen, Grønli e Ghinea 2018, uma das filosofias do MDD, não exclusivo ao desenvolvimento de aplicativos, é permitir que indivíduos que não possuem conhecimento técnico de desenvolvimento de *software* mas que são especialistas na lógica de negócio da aplicação a ser desenvolvida possam modelar o sistema.

Umuhoza e Brambilla 2016 lista quatro soluções comerciais orientadas a modelo, Mendix App Platform, IBM Rational Rhapsody, WebRatio Mobile Platform e Appian Mobile. Porém, uma pesquisa realizada por Biørn-Hansen et al. 2019 sobre as ferramentas de desenvolvimento multiplataforma na indústria não obteve nenhuma resposta citando o uso de MDD, o que pode indicar uma baixa adesão ao método pela comunidade de desenvolvimento.

4 Revisão Bibliográfica

Foi realizada uma revisão da literatura com o objetivo de identificar aspectos, questões ou desafios da área de desenvolvimento de aplicativos móveis, focado na comparação entre desenvolvimento nativo e desenvolvimento multiplataforma, que ainda não foram abordados, ou não foram abordados a contento.

O motor de busca escolhido foi o Scopus[1] e a *string* de busca utilizada foi "((("mobile application") OR ("mobile development")) AND (hybrid OR multi-platform OR cross-platform OR native))". A primeira parte foca no desenvolvimento de aplicativos móveis, e a segunda limita a artigos que comentem o método de desenvolvimento, entre nativo e multiplataforma ou híbrido. A busca também foi limitada a artigos publicados desde 2015.

A primeira etapa consistiu em analisar os 868 artigos recuperados pela busca, realizando a leitura dos títulos. Os artigos de interesse foram selecionados, diminuindo o escopo para 79 artigos. Na segunda etapa foram lidos os resumos de todos os artigos para realizar uma nova exclusão, levando a 41 artigos.

A etapa seguinte consistiu em ler os artigos na íntegra, resultando em uma seleção final de 33 artigos. Dos 8 artigos descartados, dois foram por indisponibilidade de acesso e o restante por não ser relacionado a revisão.

A partir da leitura dos artigos, foi identificada uma sugestão de trabalho futuro na revisão da literatura (em inglês, *survey*) realizada por Biørn-Hansen, Grønli e Ghinea 2018: o estudo da ferramenta Flutter como *framework* de desenvolvimento multiplataforma. A data de publicação recente do artigo, janeiro de 2019, e a extensa revisão realizada pelos autores eram boas indicações de que a ferramenta de fato ainda não fora abordada a contento, porém, como garantia, foram levantadas todas as ferramentas de desenvolvimento multiplataforma mencionadas pelos artigos lidos, com a limitação de artigos publicados a partir de 2017, ano de lançamento do *framework*.

Appcelerator Titanium	MoSync	Sencha Touch
Apache Cordova	NativeScript	Xamarin
Ionic Framework	PhoneGap	Xamarin Forms
jQuery Mobile	React Native	

Tabela 2 – Ferramentas analisadas em artigos que realizaram a implementação de alguma aplicação

No levantamento as ferramentas foram separadas em duas categorias: (1) ferramen-

[1] <https://www.scopus.com/>

tas analisadas em artigos de comparação entre desenvolvimento nativo e desenvolvimento multiplataforma, que realizaram a implementação de alguma aplicação; e (2) ferramentas citadas apenas em revisões da literatura, revisões e outros artigos que realizaram estudo da ferramenta sem implementação. As ferramentas citadas na primeira categoria são apresentadas na tabela 2, enquanto as ferramentas analisadas pela segunda categoria estão apresentadas na tabela 3.

Flutter está na segunda categoria, tendo sido citado apenas no *survey* mencionado anteriormente, como sugestão de trabalho futuro, e por Santos et al. 2019 como *framework* que deveria ter sido considerado no questionário apresentado pelo artigo. Após esse levantamento, e considerando o lançamento recente do *framework*, foi definido como objetivo deste trabalho o estudo da ferramenta, e a comparação do seu uso no desenvolvimento multiplataforma com o desenvolvimento nativo de aplicativos.

Adobe AIR	EvoThings	Mobia	RhoMobile
Alpha	Firebase	Mobl	RoboVM
Angular	**Flutter**	MobML	Sencil.js
AppGyver	Fusetools	Modeler	Smartface Cloud
Appian	Glimmer.js	MonoCross	Svelte Framework
Applause	Intel App Framework	Moon.js	Tabris.js
Apportable	Intel XDK	MOPPET	Touchstone.js
Automobile	Jasonette	MoSync	Trigger.io
AXIOM	Kony	NSB/AppStudio	Unity
Bootsrap	LuaView	OnsenUI	Unreal
Capacitor	MAML	Polymer	viperHTML
Cocoon	Marmalade	Preact	Vue.js
Cocos	MD2	Qt Mobile	WebRatio
Codename One	mdsl	Quasar Framework	Weex
Corona	Mendix	RAD Studio	XIS-Mobile
Crosslight	Meteor	React.js	Xmob
DragonRAD	Mithril	Reapp	Xojo Mobile
Ember.js	MobDSL	Rhodes	Zuix

Tabela 3 – Ferramentas citadas em *surveys*, *reviews* e artigos que não se encaixam na categoria anterior

Para comparar os dois modos de desenvolvimento, foram levantados os critérios mais utilizados pelos artigos para comparação de *frameworks*, bem como problemas e desafios encontrados no desenvolvimento multiplataforma. Para isso foi feita uma segunda seleção dos artigos, retirando revisões da literatura, taxonomias e artigos que não mencionavam diretamente alguma dificuldade apresentada por um *framework* ou algum critério de análise, levando a 23 artigos. Foram encontrados mais de 40 itens, e selecionados os cinco mais citados, como consta na tabela 4.

Artigo	Performance	*User experience*	Acesso a funcionalidades nativas	Documentação	Reuso de código
Brito et al. 2019	×	×			
Biørn-Hansen, Grønli e Ghinea 2019	×	×			
Meirelles et al. 2019	×			×	×
Biørn-Hansen et al. 2019	×	×			
Delia et al. 2019	×	×	×	×	×
Pinto e Coutinho 2018			×		
Brito, Gomes e Bernardino 2018	×			×	×
Jia, Ebone e Tan 2018	×				
M. S. Ferreira et al. 2018	×			×	
Ahmad et al. 2018		×			×
Delia et al. 2017	×	×	×		
Que, Guo e Zhu 2016	×				
Ciman e Gaggi 2016	×				

Artigo	**Performance**	***User experience***	**Acesso a funcionalidades nativas**	**Documentação**	**Reuso de código**
Vilcek e Jakopec 2017				×	
McKay 2017					
Martinez e Lecomte 2017				×	×
Botella, Escribano e Peñalver 2016			×		
Ahti, Hyrynsalmi e Nevalainen 2016	×	×			
Willocx, Vossaert e Naessens 2016	×				×
Ptitsyn e Radko 2016	×		×	×	
Willocx, Vossaert e Naessens 2015	×	×			
Delia et al. 2015		×	×		×
Lim 2015			×		
Total	15	9	7	7	7

Tabela 4 – Problemas encontrados nos artigos analisados

O parâmetro para escolha dos itens foram aqueles citados em pelo menos 1/4 dos artigos selecionados. Um dos artigos, McKay 2017, não menciona nenhum dos itens, por focar especificamente em acessibilidade, que, apesar de ser uma questão importante, não foi tão citada quanto as presentes na tabela 4.

4.1 Critérios de comparação

4.1.1 Performance

Performance é citada de diversas maneiras nos artigos, tanto como um conceito único, quanto pelos seus indicativos, sendo os mais citados: uso de memória, tamanho da aplicação, uso de CPU e tempo de inicialização. Cada um é abordado a seguir.

4.1.1.1 Consumo de memória

Consumo de memória se refere à quantidade de memória RAM alocada pela aplicação, sendo crucial em dispositivos de baixo custo, pois aplicativos se tornam lentos se não há memória suficiente, o que afeta a usabilidade dos mesmos [Ahti, Hyrynsalmi e Nevalainen 2016] [Biørn-Hansen, Grønli e Ghinea 2019].

A quantidade de memória alocada varia conforme o ciclo de vida do aplicativo, e é interessante medi-la quando o aplicativo está em *background*, quando geralmente a aplicação necessita de menos RAM. Willocx, Vossaert e Naessens 2015 observaram que Android aloca mais memória para um aplicativo comparado ao iOS, porém também libera consideravelmente mais memória ao mover um aplicativo para *background*. Os autores constataram também que enquanto há diferença de alocação de memória entre aplicativos nativos e multiplataforma quando comparados utilizando dispositivos Android de baixo e alto nível, o iOS apresenta comportamento semelhante entre os aplicativos, independente do dispositivo.

Já comparando entre métodos de desenvolvimento multiplataforma diferentes, Willocx, Vossaert e Naessens 2016 verificaram que implementações do método *cross-compiled* se comportam semelhante a implementação nativa, tendo um leve crescimento no uso de memória em relação à nativa. Já aplicações que utilizam *frameworks* Javascript alocam muito mais memória, observado também por Que, Guo e Zhu 2016.

4.1.1.2 Tamanho da aplicação

O tamanho da aplicação instalada é um fator importante no desenvolvimento móvel, visto que celulares tem memória persistente menor que computadores, especialmente dispositivos de baixo custo com recursos limitados. Assim, é desejável que aplicativos utilizem o menor espaço possível em disco [Ahti, Hyrynsalmi e Nevalainen 2016]. Willocx,

Vossaert e Naessens 2015, Willocx, Vossaert e Naessens 2016 e Jia, Ebone e Tan 2018 mensuraram o tamanho de aplicativos nativos e multiplataforma e observaram que aplicações nativas ocupam menos espaço em disco comparada a aplicações multiplataforma utilizando *frameworks* variados - PhoneGap, Xamarin, Xamarin Forms, Famo.us, IAF, Ionic, jQuery Mobile, Mgwt, Sencha Touch 2, Adobe AIR, NeoMAD, Appcelerator Titanium e Apache Cordova.

Willocx, Vossaert e Naessens 2015 apontam também a importância do tamanho em disco do instalador da aplicação, em formato APK para Android e .ipa para iOS, visto que um instalador compacto é crucial para usuários que baixam aplicações por conexão de internet móvel. Que, Guo e Zhu 2016 observaram que o tamanho do instalador também tem impacto no tempo de instalação - quanto maior, mais demorada a instalação do aplicativo.

4.1.1.3 Consumo de CPU

Consumo de CPU é a porcentagem da capacidade total de CPU de um dispositivo utilizada por uma aplicação durante um período de tempo. Segundo Willocx, Vossaert e Naessens 2015, aplicações com alto uso de CPU podem impactar negativamente outros processos sendo executados no dispositivo. Os autores consideraram importante também mensurar o uso de CPU na inicialização de aplicações multiplataforma, que podem apresentar algum *overhead* adicional pela tecnologia utilizada pelo *framework*.

Os resultados apresentados nos artigos de Willocx, Vossaert e Naessens 2015, Willocx, Vossaert e Naessens 2016, Que, Guo e Zhu 2016 indicam um maior consumo de CPU por aplicações multiplataformas comparadas às nativas, utilizando os *frameworks*: Framo.us, Intel App Framework, Ionic, jQuery Mobile, Mgwt, Sencha Touch 2, Adobe AIR, NeoMAD, Xamarin, PhoneGap e Titanium. Entretanto, Biørn-Hansen, Grønli e Ghinea 2019 encontraram resultados divergentes dos citados, num estudo focado em animações nos aplicativos. Nesse caso, as ferramentas Ionic e Xamarin tiveram consumo de CPU inferior as implementações nativas, em iOS e em Android, respectivamente.

4.1.1.4 Tempo de inicialização

Tempo de inicialização é uma importante medida de performance e também de experiência do usuário. Segundo Ahti, Hyrynsalmi e Nevalainen 2016, como aplicações móveis muitas vezes são utilizadas por um curto período de tempo antes de serem terminadas ou passadas para segundo plano, é essencial que iniciem rapidamente.

Ptitsyn e Radko 2016 citam que aplicações implementadas com a ferramenta Titanium, do método interpretado, tem um tempo de inicialização maior que o nativo, pois exige processamento de código adicional antes de executar a aplicação. M. S. Ferreira et al. 2018 confirma esse fato com experimentos com Titanium, PhoneGap e Sencha Touch.

Titanium teve resultado inferior em ambos os sistemas operacionais, enquanto Sencha Touch e PhoneGap apresentaram resultados melhores que a aplicação nativa em Android.

4.1.2 User experience

Experiência do usuário, mais comumente referida pelo seu termo em inglês *user experience* ou ainda pela sigla UX, é o segundo critério mais citado pelos artigos lidos, considerado por Ahti, Hyrynsalmi e Nevalainen 2016 o aspecto mais importante de uma aplicação. Questionários realizadas por Ahmad et al. 2018 e Biørn-Hansen et al. 2019, com 34 e 101 desenvolvedores participantes, respectivamente, indicam que *user experience* é um dos maiores desafios no desenvolvimento móvel, e mais crítico no desenvolvimento multiplataforma em comparação ao nativo. Foi, inclusive, um dos motivos que levou o Facebook a abandonar, em 2012, o desenvolvimento híbrido dos seus aplicativos, retornando, naquela época, ao desenvolvimento nativo (importante ressaltar que o *framework* React Native, do método interpretado, foi criado pela empresa 3 anos depois [Facebook 2012] [Facebook 2012]).

Já é um consenso a importância desse aspecto das aplicações móveis, porém, não há uma definição única do conceito e do melhor modo de mensurá-lo. Delia et al. 2019 define UX como um conjunto de fatores relacionados à satisfação do usuário ao utilizar o produto, alguns deles sendo tempo de resposta, design da interface de usuário, usabilidade e similaridade com os padrões do sistema operacional. Ao comparar implementações nativas e multiplataforma, o autor realizou uma pesquisa com participantes utilizando os aplicativos, por considerar *user experience* um aspecto subjetivo. Ahti, Hyrynsalmi e Nevalainen 2016 consideram difícil caracterizar em termos de medidas quantitativas, e em seu artigo focam na velocidade de resposta de uma aplicação aos gestos do usuário e em que tipo de visão holística a aplicação dá ao usuário. Similarmente, Willocx, Vossaert e Naessens 2015 consideraram como métricas de UX em seus experimentos diferentes tempos de resposta (navegação entre páginas, iniciar, pausar e retomar o aplicativo), além do uso de CPU, uso de memória e uso de bateria.

Por fim, outro consenso entre os artigos que apresentaram experimentos foi a excelência dos aplicativos nativos no quesito *user experience*, contra resultados variados nas aplicações multiplataforma, dependendo muito do *framework* escolhido.

4.1.3 Acesso a funcionalidades nativas

Segundo Delia et al. 2019, o acesso a funcionalidades nativas, ou ainda, funcionalidades específicas do dispositivo, consiste na disponibilidade de acesso a câmeras, sensores, notificações, calendários, entre outros recursos do dispositivo. Como visto por Lim 2015 e Delia et al. 2015, a abordagem híbrida necessita de *plugins* externos para obter esse

acesso, e o primeiro autor ainda observa que a camada de *browser engine* (mecanismo de renderização de um navegador *web*) do *framework* gera *overhead*, isto é, aumenta o tempo de acesso às funcionalidades.

Botella, Escribano e Peñalver 2016 levantam também a importância de comparar a compatibilidade de componentes entre plataformas, isto é, um componente, disponibilizado pelo *framework* ou como um *plugin*, para acessar um recurso do dispositivo pode funcionar em apenas uma das plataformas. A partir de experimentos, o autor cita o Ionic como tendo bons resultados nesse critério, observado também por Delia et al. 2019.

4.1.4 Documentação

Vilcek e Jakopec 2017, Brito, Gomes e Bernardino 2018 e Delia et al. 2019 consideram uma boa documentação do *framework* importante pois diminui a curva de aprendizado e torna o desenvolvimento mais fácil e viável. A pesquisa realizada por Meirelles et al. 2019 com estudantes e profissionais da área de desenvolvimento também indica a qualidade da documentação como um critério decisivo para a escolha de uma ferramenta.

Um exemplo citado de *framework* bem documentado foi o React Native, por Martinez e Lecomte 2017 e por Brito, Gomes e Bernardino 2018. Porém, é importante ressaltar que nenhum dos artigos definiu um método empírico de mensurar a qualidade ou quantidade de documentação, tendo sido feito de forma subjetiva.

4.1.5 Reuso de código

Como exposto na seção de fundamentação teórica, no método de desenvolvimento nativo não existe compartilhamento, ou reuso, de código entre as plataformas-alvo das aplicações móveis. Isso foi possível com a criação dos *frameworks* multiplataforma, que permitem reutilizar parcial ou totalmente a base de código de um aplicativo.

Segundo Martinez e Lecomte 2017 e Delia et al. 2019, o compartilhamento de código entre plataformas reduz consideravelmente os custos de desenvolvimento e manutenção. Delia et al. 2019 também ressaltam a importância da porcentagem de reuso que uma ferramenta consegue alcançar, por variar muito de método para método. Willocx, Vossaert e Naessens 2016 citam que alguns *frameworks* que utilizam Javascript apresentam 100% de reuso de código, com a desvantagem de não ter acesso a todas as funcionalidades nativas. Ele segue comentando que outras ferramentas, como Titanium[2], oferecem uma interface gráfica uniforme entre plataformas mas ao mesmo tempo permitem uso de código específico para acesso de funcionalidades de cada plataforma, enquanto o Xamarin[3] implementa interface gráfica diferenciada para cada plataforma, compartilhando o resto do código. A

[2] <https://www.appcelerator.com/titanium/titanium-sdk/4/>
[3] <https://dotnet.microsoft.com/apps/xamarin>

porcentagem de reuso de código no Xamarin varia muito conforme observado entre os artigos, indo de 50% de compartilhamento [Delia et al. 2015] até 93% [Martinez e Lecomte 2017] .

Brito, Gomes e Bernardino 2018 citam também outro tipo de reuso de código, entre aplicações diferentes, por meio de reuso de componentes gráficos, permitido, por exemplo, pelo React Native. O Flutter, a ser abordado nesse trabalho, também permite o reuso de componentes entre aplicações.

5 Planejamento do experimento

Neste capítulo são apresentadas as definições referentes ao experimento realizado para comparar o desenvolvimento nativo com o uso do *framework* Flutter.

5.1 Definição do aplicativo

A fim de comparar o desenvolvimento nativo de aplicativos com o desenvolvimento multiplataforma, foi definido um aplicativo a ser implementado em três versões: (1) nativa para Android, (2) nativa para iOS, e (3) multiplataforma (para Android e iOS), utilizando Flutter. O aplicativo foi criado para ser de simples implementação, descrito por uma lista de requisitos funcionais.

R1 O usuário pode registrar uma tarefa, informando uma breve descrição da tarefa, e, opcionalmente, uma data de lembrete.

R2 O sistema deve mostrar uma lista com todas as tarefas criadas, ordenadas por realizadas e não realizadas.

R3 O usuário pode informar se uma tarefa foi realizada.

R4 O sistema deve notificar o usuário na data de lembrete, caso tenha sido definida, sobre uma tarefa ainda não realizada.

R5 O usuário pode excluir uma tarefa da lista de tarefas.

R6 O usuário pode registrar uma anotação, inserindo texto e/ou imagens, tiradas pela câmera do dispositivo ou selecionadas na galeria de fotos do mesmo.

R7 O sistema deve mostrar uma lista de todas as anotações criadas, ordenadas por data de criação.

R8 O usuário pode excluir uma anotação da lista.

R9 O sistema deve garantir persistência de todos os dados cadastrados, isto é, deve manter os dados salvos após entrar em segundo plano ou ser fechado.

5.2 Definição das métricas

Para comparar as três versões do aplicativo, foram definidas métricas relacionadas aos critérios encontrados no capítulo 4.

Métrica 1.1: consumo de memória RAM ao abrir o aplicativo, em megabytes.

Métrica 1.2: consumo de memória RAM pelo aplicativo em segundo plano, em megabytes.

Métrica 1.3: consumo máximo de memória RAM durante o uso do aplicativo,em megabytes.

Métrica 2.1: espaço em disco utilizado pelo aplicativo, em megabytes.

Métrica 2.2: espaço em disco utilizado pelo instalador do aplicativo, em megabytes.

Métrica 3.1: consumo de CPU ao iniciar o aplicativo, em porcentagem sobre a capacidade total do dispositivo.

Métrica 3.2: consumo máximo de CPU durante o uso do aplicativo, em porcentagem sobre a capacidade total do dispositivo

Métrica 4.1: tempo de inicialização do aplicativo, em milissegundos.

Métrica 4.2: tempo para retomar o aplicativo, quando em segundo plano, em milissegundos.

Métrica 4.3: tempo de navegação entre páginas do aplicativo, em milissegundos.

Métrica 5: acesso às funcionalidades nativas necessárias (câmera, sistema de notificações e sistema de arquivos).

Métrica 6: reuso do código, em porcentagem de código escrito utilizado para ambas as plataformas sobre o total (não considera código gerado).

Métrica 7: qualidade da documentação, conforme a disponibilidade de três elementos:

E1 Trechos de código, curtos e comentados, para compreender uma funcionalidade básica de um componente da API;

E2 Tutoriais com passo-a-passo para compreender como desenvolver uma funcionalidade com vários componentes da API;

E3 Manual de referência baixo nível documentando todos os componentes da API.

O critério performance foi dividido em 4 conjuntos de métricas diferentes, relacionadas a consumo de RAM, consumo de CPU, tamanho da aplicação em disco, e tempo de inicialização, como visto no capítulo anterior. Devido a falta de consenso quanto ao critério de *user experience* e ao escopo do trabalho, esse critério foi relacionado ao tempo de resposta, tempo de inicialização e tempo de retomada, como sugerido por Delia et al. 2019. Estão relacionados também consumo de memória, que pode tornar a a aplicação lenta, e consumo de CPU, que pode impactar negativamente outros processos executados

no dispositivos, ambos piorando a experiência do usuário [Ahti, Hyrynsalmi e Nevalainen 2016] [Willocx, Vossaert e Naessens 2015].

Como comentado na seção 4.1.5, nenhum dos artigos de comparação de *frameworks* utilizou um método para mensurar a qualidade da documentação destes. Buscou-se na literatura, um modo de avaliar a qualidade de documentação de APIs. Numa revisão da literatura, Cummaudo, Vasa e Grundy 2019 levantaram os três elementos mais citados necessários para uma documentação de API de qualidade, definidos em **E1**, **E2** e **E3**.

Critério de comparação	**Métricas**
Performance	1.1, 1.2, 1.3, 2.1, 2.2, 3.1, 3.2, 4.1, 4.2, 4.3
User experience	1.1, 1.2, 1.3, 3.1, 3.2, 4.1, 4.2, 4.3
Acesso a funcionalidades nativas	5
Reuso do código	6
Documentação	7

Tabela 5 – Relação entre critérios de comparação e métricas

5.3 Dispositivos utilizados

Para obter os resultados, os aplicativos desenvolvidos foram instalados em dois dispositivos *smartphone*, com sistemas operacionais diferentes. Em cada dispositivo foi instalada a versão nativa apropriada do aplicativo, e a versão em Flutter foi instalada em ambos.

Dispositivo	iOS	Android
Modelo	iPhone 8	Galaxy A10s
Sistema operacional	iOS 13	Android 10
Memória RAM	2GB	2GB
CPU	2.39 GHz hexa-core 64-bit	1.8GHz hecta-core
Armazenamento	64GB	32GB

Tabela 6 – Dispositivos utilizados para análises de performance

5.4 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para obter os resultados referentes as métricas 1 a 4, como pode ser visto na tabela 7.

Métrica	**Ferramentas**
1.1, 1.2 e 1.3	Android Profiler e Xcode
2.1 e 2.2	Visível no dispositivo
3.1 e 3.2	Android Profiler e Xcode
4.1, 4.2 e 4.3	Android Profiler, Flutter Driver e Xcode UI Testing

Tabela 7 – Relação entre métricas obtidas e ferramentas utilizadas

6 Desenvolvimento

Neste capitulo é apresentado o desenvolvimento das três versões do aplicativo proposto no capítulo anterior.

6.1 Flutter

Flutter é um *toolkit* de código aberto para desenvolvimento de aplicativos multiplataforma, incluindo os sistemas Android e iOS, focado no desenvolvimento de interface de usuário. Tem como proposta possibilitar a criação de aplicativos visualmente agradáveis, diminuindo o tempo, o custo e a complexidade do desenvolvimento [Flutter 2019].

A ferramenta se destaca de outras no mercado de desenvolvimento multiplataforma de aplicativo por não usar *WebView* nem componentes nativos do sistema operacional alvo, sendo classificada no método *cross-compiled* por Biørn-Hansen, Grønli e Ghinea 2018. Para renderizar a interface, Flutter utiliza a *engine* de renderização 2D Skia[1], escrita em C/C++. Apesar de não utilizar os componentes nativos, o *toolkit* disponibiliza uma biblioteca de *widgets* - componentes gráficos - que implementa a linguagem de design padrão dos dispositivos Android, Material Design[2], e uma biblioteca que implementa a linguagem de design do iOS, referida pelo Flutter como Cupertino[3]. Ambas as bibliotecas são atualizadas, por contribuidores da Google e pela comunidade, conforme novos componentes nativos são lançados nos sistemas operacionais alvo.

Aplicativos Flutter são construídos a partir dos já citados *widgets*, componente básico de interface gráfica. A renderização da interface utiliza três conceitos diferentes: (1) o *widget*, que consiste na declaração das propriedades da interface gráfica do componente; (2) o elemento, que mantém o lugar do componente na hierarquia de interface; e (3) o objeto de renderização, responsável por de fato renderizar o componente na tela, a partir das informações de tamanho, cor e posicionamento. Quando a propriedade de um *widget* é modificada - por exemplo, modificar a cor de um componente de texto - o *widget* é destruído e substituído por um novo, mas o elemento e o objeto de renderização associados a ele são apenas atualizados com a propriedade nova, minimizando a quantidade de processamento necessária para atualizar a interface [Flutter 2019].

Para ser executado tanto em dispositivos iOS quanto Android, o código da *engine* em C/C++ é compilado com NDK ou LLVM, enquanto o código Dart tanto da aplicação quanto do SDK são compilados *ahead-of-time* para código de máquina da plataforma alvo,

[1] <https://skia.org/>
[2] <https://material.io/>
[3] <https://flutter.dev/docs/development/ui/widgets/cupertino>

que é então incluído num projeto nativo para formar o instalador do aplicativo, .ipa para iOS e APK para Android.

6.2 Versões desenvolvidas

Versão do aplicativo	A1	A2	A3
SDK	Android SDK	Flutter SDK 1.22	iOS SDK
Plataforma	Android	Android, iOS	iOS
Linguagem	Kotlin	Dart	Swift 5.0
IDE	Android Studio	VS Code	Xcode

Tabela 8 – Versões do aplicativo desenvolvidas

Buscou-se tomar decisões de arquitetura similares em todas as versões. Para garantir o requisito **R9**, de persistência de dados, todos os dados inseridos na aplicação pelo usuário são salvos em banco de dados relacional local, em SQLite. As SDKs nativas possuem bibliotecas para implementação de banco de dados, sendo elas Core Data para iOS e Room para Android. A versão em Flutter necessitou de um *plugin* externo, sqflite, desenvolvido e mantido pela comunidade. Importante relembrar que o requisito é apenas persistência local, e o aplicativo não necessita de acesso a Internet para funcionar.

Para cumprir os requisitos foram implementadas três telas: lista de tarefas (requisitos **R1**, **R3** e **R5**), lista de anotações (requisito **R7**), e uma tela para visualização e edição de anotações (requisitos **R6** e **R7**), além de uma *dialog* para criação de tarefas (requisito **R1**). Tal como a arquitetura, buscou-se implementar as versões visualmente equivalentes, como pode ser visto na figura 1.

Os requisitos **R4** e **R6** necessitam que o aplicativo acesse o sistema de notificações, a câmera e o sistema de arquivos do dispositivo. Ambos os sistemas operacionais exigem que a aplicação solicite ao usuário permissão para acessar essas funcionalidades, além da definição prévia nas configurações de cada projeto sobre quais serão utilizadas. Os SDKs nativos oferecem as APIs necessárias para realizar a solicitação e acesso. Já em Flutter, novamente são disponibilizados pela comunidade *plugins* a serem adicionados ao projeto, que abstraem o acesso as APIs nativas de cada recurso. Assim como no caso nativo, também é necessário definir nos arquivos de configuração, um para cada plataforma, quais funcionalidades serão utilizadas.

Por meio dos *plugins*, foi possível acessar todos os recursos necessários na versão multiplataforma, com pouquíssimo código específico para cada uma (a ser comentado no capítulo de resultados).

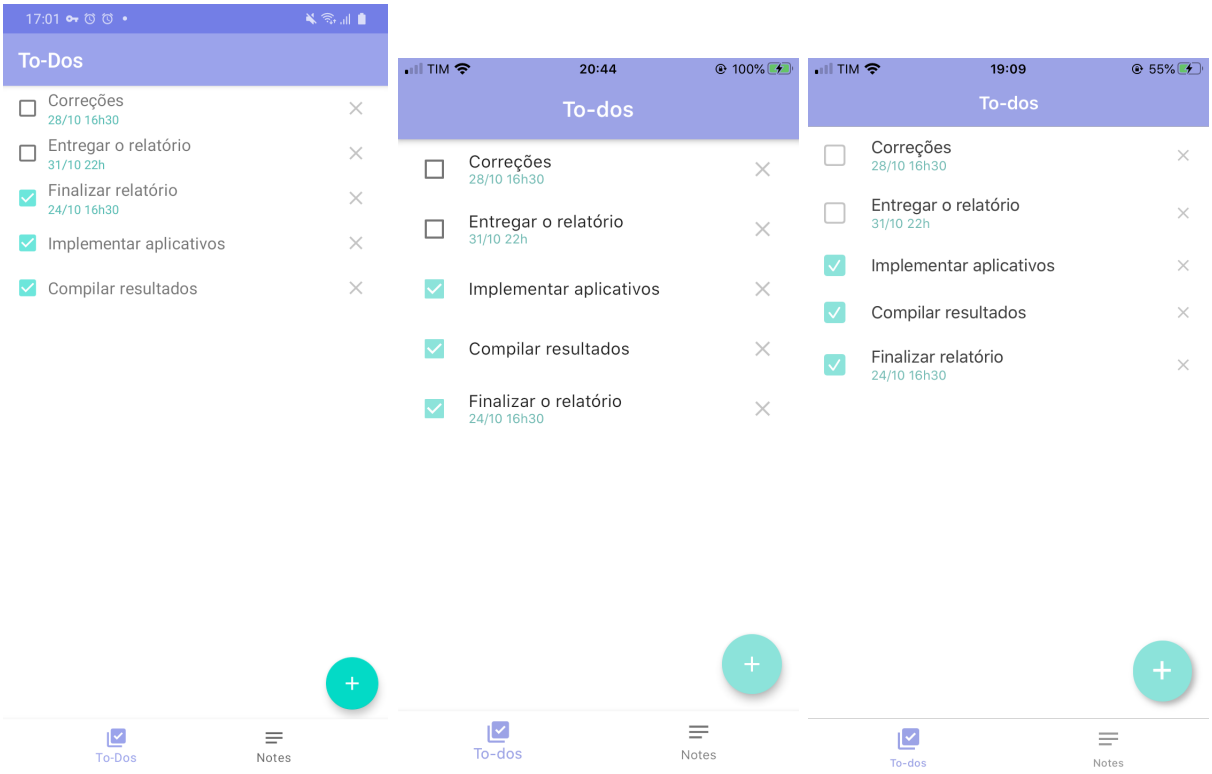

Figura 1 – *Screenshots* da tela inicial da versão A1 no dispositivo Android, e das versões A2 e A3 no dispositivo iOS. Fonte: própria (2020).

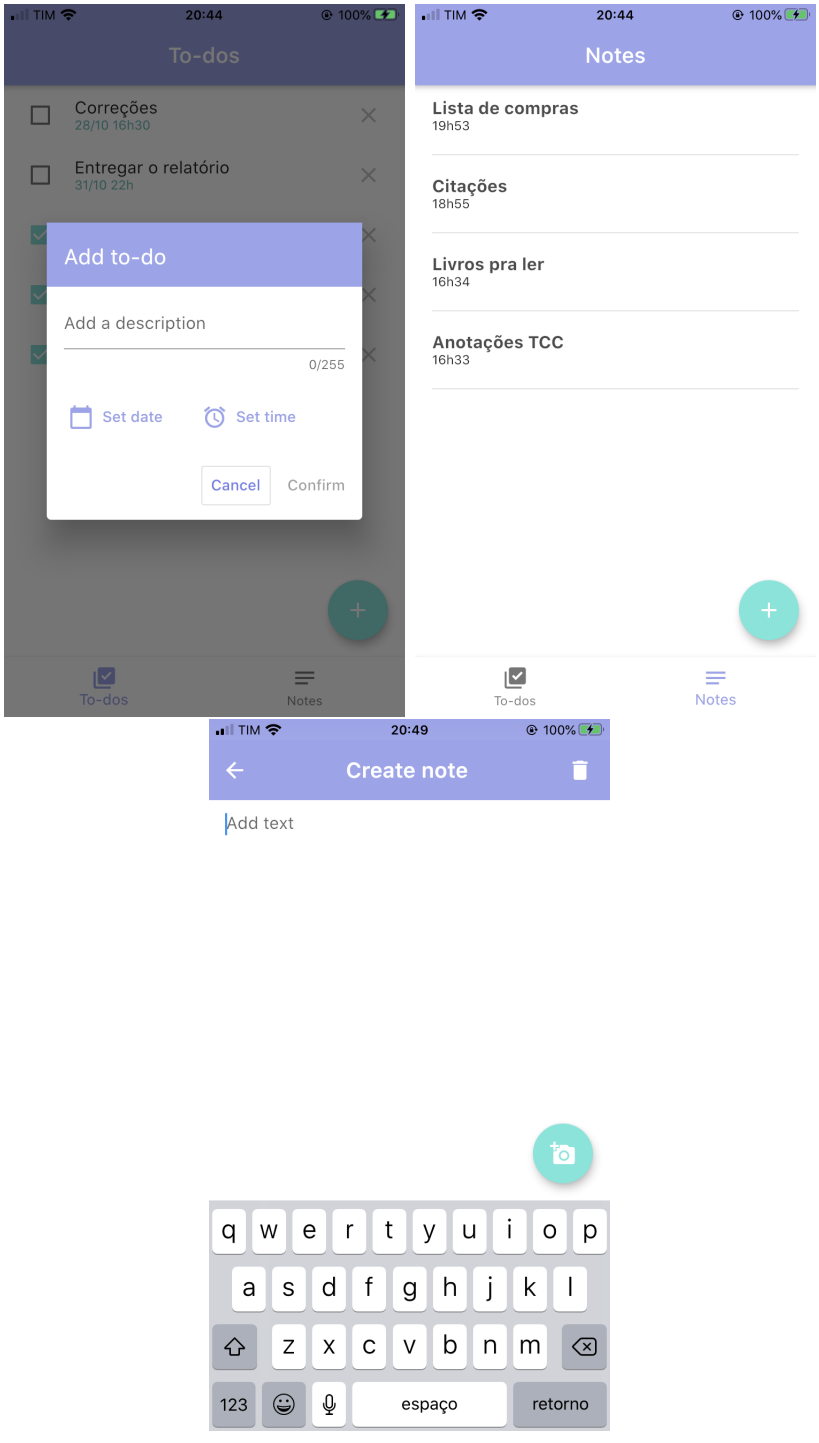

Figura 2 – *Screenshots* da *dialog* para criação de tarefas, tela de anotações e tela de edição, da versão A2 no dispositivo iOS. Fonte: própria (2020).

7 Resultados

Neste capítulo são apresentados os resultados obtidos a partir do experimento definido no capítulo 5.

7.1 Consumo de memória RAM

Métrica 1.1

A primeira métrica se refere ao consumo de memória RAM ao iniciar o aplicativo. No dispositivo Android, o consumo foi similar entre as versões nativa e multiplataforma, enquanto no dispositivo iOS, a versão nativa obteve notável vantagem ao utilizar apenas 32% da memória consumida pela aplicação em Flutter.

	Android	iOS
Nativo	89,6	22,7
Flutter	93,2	69,4

Tabela 9 – Métrica 1.1: consumo de memória RAM ao abrir o aplicativo, em megabytes.

Métrica 1.2

Como apontado por Willocx, Vossaert e Naessens 2015, é interessante observar a diferença de alocação de memória para o aplicativo quando é movido para segundo plano, ou *background*. Foi possível observar o comportamento apontado pelos autores: ambas as versões nativa e multiplataforma consumiram mais memória no dispositivo Android em primeiro plano, porém liberaram mais memória ao serem movidas para segundo plano, comparadas as versões instaladas no dispositivo iOS.

	Android	iOS
Nativo	46,5	22,3
Flutter	75,9	67

Tabela 10 – Métrica 1.2: consumo de memória RAM pelo aplicativo em segundo plano, em megabytes.

Métrica 1.3

Uma última métrica de consumo de RAM foi obtida realizando diversas ações no

aplicativo. Em cada versão e dispositivo, as mesmas ações foram executadas, na mesma ordem, e foi obtido o maior valor de consumo de memória. As ações foram: *scroll* pela lista de tarefas da tela inicial, criar uma tarefa nova, inserir data de lembrete na tarefa, abrir a aba de anotações e realizar *scroll* pela lista, criar anotação nova, obter foto pela câmera do dispositivo e adicionar na anotação, voltar para a lista de anotações e reabrir a anotação com foto.

	Android	iOS
Nativo	290,8	155,9
Flutter	250,3	148

Tabela 11 – Métrica 1.3: consumo máximo de memória RAM durante o uso do aplicativo, em megabytes.

Em todas as versões e dispositivos, o pico no consumo de memória ocorreu ao executar a ação de adicionar a foto na anotação. É possível notar que, apesar de nas outras métricas a versão multiplataforma ter obtido desvantagem em relação a versão nativa no dispositivo iOS, nesse caso ambas obtiveram resultados muito próximos, com Flutter mostrando uma leve vantagem.

7.2 Espaço em disco

Métrica 2.1

Como visto na revisão bibliográfica, é esperado que versões multiplataforma ocupem mais espaço em disco do que versões nativas. Os resultados encontrados variaram drasticamente entre plataformas. Para Android, a versão multiplataforma, teve 2,65 vezes o tamanho da versão nativa, porém mantendo um tamanho ainda razoável, de 26,12MB.

	Android	iOS
Nativo	9,95	1,2
Flutter	26,12	254,9

Tabela 12 – Métrica 2.1: espaço em disco utilizado pelo aplicativo, em megabytes.

Já na plataforma iOS, a diferença entre os tamanhos foi excepcional. Enquanto a versão nativa ocupa ínfimos 1,2MB, a versão multiplataforma chegou ao tamanho de 254,9MB. Devido a grande disparidade entre os valores, buscou-se entender a causa. O Flutter SDK disponibiliza uma ferramenta de linha de comando que informa um relatório

do espaço utilizado por cada componente da aplicação. A partir do uso dessa ferramenta foi possível observar que 87,74% do tamanho da aplicação final consiste no próprio *framework*.

Para ambas as plataformas, foram instaladas as *builds* versão *release* dos aplicativos. Porém, é possível que ao publicar o aplicativo nas respectivas lojas e realizar a instalação a partir delas, o tamanho final varie. Devido ao custo de se publicar um aplicativo na App Store, foram analisados apenas os tamanhos das *builds* geradas localmente.

Métrica 2.2

Também foram obtidos os tamanhos dos instaladores, como sugerido por Willocx, Vossaert e Naessens 2016 e Que, Guo e Zhu 2016. Entretanto, não é possível gerar instaladores .ipa, utilizados pelos dispositivos iOS, sem uma conta no Apple Developer Program[1]. Novamente devido ao custo monetário para obter a conta, não foi possível obter valores referentes às versões nativa e multiplataforma para iOS.

	Android	iOS
Nativo	3,7	N/A
Flutter	8,4	N/A

Tabela 13 – Métrica 2.2: espaço em disco utilizado pelo instalador, em megabytes.

Apesar do tamanho do instalador da versão em Flutter ser o dobro da versão nativa, ainda é um valor aceitável, considerando que a média de velocidade de download das redes de internet móvel no Brasil era 19,67MB/s em 2018 [OpenSignal 2018].

7.3 Consumo de CPU

Métrica 3.1

O consumo de CPU foi medido ao iniciar os aplicativos. Nesse critério, as versões nativas novamente mostraram resultados superiores ao *framework*, semelhante ao encontrado por Willocx, Vossaert e Naessens 2016 e Que, Guo e Zhu 2016 ao comparar o desenvolvimento nativo com outras ferramentas. Os comportamentos das versões nativas foram similares, assim como os comportamentos da versão Flutter em cada dispositivo.

Métrica 3.2

Esta métrica é similar a métrica 1.3, consistindo no consumo máximo de um recurso durante o uso do aplicativo, dessa vez de CPU. As mesmas ações foram realizadas em todas as versões, e foi obtido o maior valor de consumo de CPU. As versões mantiveram consumo relativamente baixo no dispositivo Android, com os picos ocorrendo durante

[1] <https://developer.apple.com/programs/>

	Android	iOS
Nativo	23%	22%
Flutter	28%	31%

Tabela 14 – Métrica 3.1: consumo de CPU ao iniciar o aplicativo, em porcentagem sobre a capacidade total do dispositivo.

a navegação entre páginas e ao realizar *scroll* na lista de anotações, na versão nativa e Flutter respectivamente. Já no dispositivo iOS, o consumo foi consideravelmente mais elevado, e ocorreu ao realizar a ação de adicionar foto, em ambas as versões.

	Android	iOS
Nativo	32%	72%
Flutter	41%	86%

Tabela 15 – Métrica 3.2: consumo máximo de CPU durante o uso do aplicativo, em porcentagem sobre a capacidade total do dispositivo.

7.4 Tempo de resposta

Métrica 4.1

A métrica 4.1 consiste no tempo de inicialização do aplicativo. Foi coletado o tempo de inicialização logo após instalar o aplicativo, sem dados salvos no banco de dados. Como a tela inicial é uma lista de tarefas, carregadas do banco de dados da aplicação, para simular uma situação de uso mais próxima da real cada versão do aplicativo foi populada com a mesma quantidade de tarefas, e novamente foram obtidos os tempos de inicialização.

	BD vazio		BD populado	
	Android	iOS	Android	iOS
Nativo	1.800	1.110	1.832	1.239
Flutter	1.060	1.254	1.765	1.412

Tabela 16 – Métrica 4.1: tempo de inicialização do aplicativo, em milissegundos.

Os tempos obtidos foram de grandezas iguais, comparando entre versão Flutter e nativa. O *framework* até mesmo obteve vantagem sobre a versão Android. Porém teve um salto maior que a versão nativa ao carregar dados do banco para a lista, o que pode indicar que consultas ao banco sejam mais lentas na versão multiplataforma. Os

resultados no dispositivo iOS foram diferentes. A versão nativa manteve uma vantagem de aproximadamente 12% sobre a multiplataforma em ambos os cenários de teste.

Métrica 4.2

A próxima métrica está relacionada ao estado de segundo plano pelo qual as aplicações podem passar durante seu ciclo de vida no dispositivo, isto é, um aplicativo pode ser suspendido durante o uso, mantendo alguns dados na memória, e retomado posteriormente por opção do usuário. Os testes foram realizados nos dispositivos com as bases de dados populadas.

	Android	iOS
Nativo	823	997
Flutter	651	647

Tabela 17 – Métrica 4.2: tempo para retomar o aplicativo, quando em segundo plano, em milissegundos.

Nesse ponto, o *framework* obteve resultados melhores em ambos os dispositivos, demonstrando maior vantagem na plataforma iOS, ao utilizar apenas 64% do tempo gasto pela aplicação nativa para retomar o aplicativo.

Métrica 4.3

Essa métrica se refere ao tempo para o aplicativo navegar de uma tela para outra. Para os testes, esse cálculo foi feito ao abrir a tela de criar anotação. Em ambas as plataformas o *framework* obteve vantagem, em especial no dispositivo iOS.

	Android	iOS
Nativo	283,77	999
Flutter	163,82	52,17

Tabela 18 – Métrica 4.3: tempo de navegação entre páginas do aplicativo, em milissegundos.

7.5 Acesso a funcionalidades nativas

Métrica 5

Como comentado no capítulo de desenvolvimento, os requisitos da aplicação requerem acesso ao sistema de notificações, câmera e sistema de arquivos do dispositivo, disponibilizado por APIs do SDK de cada plataforma, no desenvolvimento nativo.

Plugin	Funcionalidade	Plataformas compatíveis
Image Picker[2]	Câmera e galeria de fotos	Android, iOS
Path Provider[3]	Sistema de arquivos	Android, iOS
Flutter Local Notifications[4]	Sistema de notificações	Android, iOS

Tabela 19 – *Plugins* utilizados para acesso a funcionalidades nativas

Já no desenvolvimento multiplataforma, foi possível acessar todos os recursos por meio de *plugins* adicionados ao projeto Flutter, que abstraem a camada de comunicação com as APIs nativas, similar ao comportamento relatado por Lim 2015 em outros *frameworks*. Todos os *plugins* utilizados eram compatíveis com ambas as plataformas, necessitando de pouco código específico para cada uma, e a aplicação conseguiu satisfazer os requisitos da mesma forma que as nativas.

7.6 Reuso de código

Métrica 6

Quanto a métrica de reuso de código, as versões nativas, por definição, não possibilitam compartilhamento do código entre as plataformas. Já o *framework* Flutter permite escrever aplicações inteiras uma vez e utilizá-las em ambas as plataformas. A versão desenvolvida tem apenas 2,6% de código específico para uma plataforma, do total de código escrito (não foi considerado para o percentual os arquivos de código gerado pela *framework* nem bibliotecas), e essas linhas de código foram necessárias apenas para configurar o acesso ao sistema de notificação dos dispositivos.

Esse resultado pode ser comparado ao encontrado por Willocx, Vossaert e Naessens 2016, ao citarem que *frameworks* que apresentam 100% de reuso de código têm como desvantagem a ausência de acesso a funcionalidades do dispositivo. De fato, caso a aplicação desenvolvida não necessitasse de acesso ao sistema de notificações, teria o código totalmente compartilhado entre plataformas. Porém, ressalta-se aqui que os *plugins* para Flutter abstraem muito da camada de comunicação com as APIs nativas, diminuindo ou até eliminando a necessidade de código específico em alguns caso, como o do *plugin* para acesso a câmera do dispositivo.

7.7 Qualidade da documentação

Métrica 7

Foi realizada a comparação da documentação oferecida pela Google para desenvolvimento Android, a documentação para desenvolvimento iOS disponibilizada pela Apple,

[2] <https://pub.dev/packages/image_picker>
[3] <https://pub.dev/packages/path_provider>
[4] <https://pub.dev/packages/flutter_local_notifications>

e a documentação do *framework* Flutter, também disponibilizada pela Google, utilizando três elementos reapresentados nesta seção para praticidade:

E1 Trechos de código, curtos e comentados, para compreender uma funcionalidade básica de um componente da API;

E2 Tutoriais com passo-a-passo para compreender como desenvolver uma funcionalidade com vários componentes da API;

E3 Manual de referência de baixo nível documentando todos os componentes da API.

A Google disponibiliza para desenvolvimento nativo Android, além da documentação das classes e métodos das APIs, tutoriais para uso das bibliotecas nativas e trechos de código de exemplo para todos os componentes utilizados no desenvolvimento deste trabalho.

Já no caso da plataforma iOS, a documentação apresentava poucos tutoriais, que não abrangiam a maioria das funcionalidades da nova biblioteca gráfica nativa, SwiftUI, além de não apresentar nenhum trecho de código para os componentes individualmente. É uma documentação muito mais crua, no sentido de apresentar apenas a definição de cada componente e seus métodos.

	Android nativo	iOS nativo	Flutter
E1	✓		✓*
E2	✓		✓*
E3	✓	✓	✓

Tabela 20 – Elementos presentes em cada documentação Legenda: * = varia entre *plugins*.

Flutter apresentou uma documentação mais completa. Todos os *widgets* têm seus métodos e atributos documentados, além de uma breve descrição do componente, e na maioria dos casos trechos de código de exemplo. São disponibilizados tutoriais oficiais demonstrando o uso de diversas funcionalidades (animações, listas, persistência, navegação, rede, entre outros).

Porém, o *framework* depende do uso de *plugins* externos para acessar funcionalidades do dispositivo, e essas ferramentas apresentam documentações separadas, de responsabilidade de cada autor, o que leva a alta variação na qualidade e disponibilidade. Para todos os *plugins*, a documentação oferecida apresentava o elemento E3, e apresentava pelo menos um dos outros dois elementos: trecho de código de exemplo ou tutorial passo-a-passo.

8 Conclusão

Com o crescimento do mercado de aplicativos para dispositivos móveis, surgem novas ferramentas para facilitar o desenvolvimento para múltiplas plataformas móveis. Este trabalho buscou comparar uma dessas ferramentas, o *framework* Flutter, com o desenvolvimento nativo. Foram levantados os critérios mais citados na revisão da literatura, e desenvolvidas três versões de um aplicativo, que foram, então, comparadas seguindo cada um desses critérios.

A performance das versões nativas, iOS e Android, se mostrou superior em algumas métricas, principalmente em tamanho da aplicação. Porém, a versão multiplataforma demonstrou vantagem nos tempos de resposta: inicialização mais rápida no dispositivo Android, e tempo para retomar a aplicação menor e navegação entre páginas mais rápida em ambos os dispositivos. Além disso, a aplicação em Flutter conseguiu acesso a todas as funcionalidades nativas necessárias (câmera, sistema de notificações e sistema de arquivos), com pouco código específico: 97,4% do código escrito era destinado a ambas as plataformas. A documentação do *framework* também se mostrou satisfatória, em especial comparada a documentação disponibilizada pela Apple.

O *framework* Flutter se mostrou como uma alternativa relevante no desenvolvimento *mobile*, pois além da vantagem intrínseca do desenvolvimento multiplataforma - uma única aplicação compatível com ambas as plataformas - conseguiu se equiparar e superar o desenvolvimento nativo em algumas métricas de performance, sem perda de funcionalidades nativas.

8.1 Trabalhos futuros

Como recomendação de trabalho futuro, sugere-se: a comparação do Flutter com React Native, outro *framework* de desenvolvimento multiplataforma, mais utilizado segundo relatório da SlashData 2019; expandir os critérios de comparação, visto que vários outros foram levantados na literatura lida, porém com menos citações; desenvolvimento e avaliação de aplicações mais complexas utilizando Flutter, que tenham como requisito, por exemplo, conexão com a Internet.

Referências

AHMAD, A. et al. An empirical study of investigating mobile applications development challenges. *IEEE Access*, PP, p. 1–1, 03 2018. Citado 2 vezes nas páginas 31 e 35.

AHTI, V.; HYRYNSALMI, S.; NEVALAINEN, O. An evaluation framework for cross-platform mobile app development tools: A case analysis of adobe phonegap framework. In: . [S.l.: s.n.], 2016. Citado 5 vezes nas páginas 32, 33, 34, 35 e 41.

APPLE. *Introduction to Swift - Apple WWDC 2014*. 2014. Acesso em 12/11/2019. Disponível em: <https://www.youtube.com/watch?v=A0C6L4XmrZM>. Citado na página 25.

BIØRN-HANSEN, A.; GRØNLI, T.-M.; GHINEA, G. A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 5, p. 108:1–108:34, nov. 2018. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/3241739>. Citado 4 vezes nas páginas 26, 27, 29 e 43.

BIØRN-HANSEN, A.; GRØNLI, T.-M.; GHINEA, G. Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance. *Sensors*, v. 19, p. 2081, 05 2019. Citado 3 vezes nas páginas 31, 33 e 34.

BIØRN-HANSEN, A. et al. An empirical study of cross-platform mobile development in industry. *Wireless Communications and Mobile Computing*, v. 2019, p. 1–12, 01 2019. Citado 3 vezes nas páginas 27, 31 e 35.

BOTELLA, F.; ESCRIBANO, P.; PEñALVER, A. Selecting the best mobile framework for developing web and hybrid mobile apps. In: . [S.l.: s.n.], 2016. p. 1–4. Citado 2 vezes nas páginas 32 e 36.

BRITO, H.; GOMES, A.; BERNARDINO, J. Javascript in mobile applications: React native vs ionic vs nativescript vs native development. In: . [S.l.: s.n.], 2018. p. 1–6. Citado 3 vezes nas páginas 31, 36 e 37.

BRITO, H. et al. Mobile development in swift, java and react native: an experimental evaluation in audioguides. In: *2019 14th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.: s.n.], 2019. p. 1–6. ISSN 2166-0727. Citado na página 31.

CIMAN, M.; GAGGI, O. An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive and Mobile Computing*, v. 39, 10 2016. Citado na página 31.

COMSCORE. *Global Digital Future in Focus 2018*. [S.l.], 2018. Citado na página 21.

CUMMAUDO, A.; VASA, R.; GRUNDY, J. What should i document? a preliminary systematic mapping study into api documentation knowledge. 07 2019. Citado na página 41.

DELIA, L. et al. Approaches to mobile application development: Comparative performance analysis. In: . [S.l.: s.n.], 2017. p. 652–659. Citado na página 31.

DELIA, L. et al. Multi-platform mobile application development analysis. In: . [S.l.: s.n.], 2015. p. 181–186. Citado 3 vezes nas páginas 32, 35 e 37.

DELIA, L. et al. Development approaches for mobile applications: Comparative analysis of features. In: ARAI, K.; KAPOOR, S.; BHATIA, R. (Ed.). *Intelligent Computing*. Cham: Springer International Publishing, 2019. p. 470–484. ISBN 978-3-030-01177-2. Citado 5 vezes nas páginas 27, 31, 35, 36 e 40.

EL-KASSAS, W. S. et al. Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal*, v. 8, n. 2, p. 163–190, june 2017. ISSN 1084-6654. Disponível em: <https://doi.org/10.1016/j.asej.2015.08.004>. Citado 2 vezes nas páginas 25 e 26.

FACEBOOK. *Under the Hood: Rebuilding Facebook for Android*. 2012. Acesso em 12/09/2020. Disponível em: <https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-android/10151189598933920/>. Citado na página 35.

FACEBOOK. *Under the hood: Rebuilding Facebook for iOS*. 2012. Acesso em 12/09/2020. Disponível em: <https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-ios/10151036091753920/>. Citado na página 35.

FLUTTER. *FAQ - Flutter*. 2019. Acesso em 12/11/2019. Disponível em: <https://flutter.dev/docs/resources/faq>. Citado na página 43.

FLUTTER. *How Flutter renders Widgets*. 2019. Acesso em 13/09/2020. Disponível em: <https://www.youtube.com/watch?v=996ZgFRENMs>. Citado na página 43.

FREIRE, A.; ANDRADE, V. *Why we think Flutter will help us scale mobile development at Nubank*. 2019. Acesso em 20/11/2019. Disponível em: <https://medium.com/building-nubank/https-medium-com-freire-why-nubank-chose-flutter-61b80b568772>. Citado na página 22.

GAGNE, Y. *Banking in Brazil is hard. Here's how Nubank is changing that*. 2019. Acesso em 20/11/2019. Disponível em: <https://www.fastcompany.com/90299054/nubank-most-innovative-companies-2019>. Citado na página 22.

IPROPERTYMANAGEMENT.COM. *Airbnb Statistics*. 2019. Acesso em 20/11/2019. Disponível em: <https://ipropertymanagement.com/airbnb-statistics>. Citado na página 22.

JIA, X.; EBONE, A.; TAN, Y. A performance evaluation of cross-platform mobile application development approaches. In: . [S.l.: s.n.], 2018. p. 92–93. ISBN 978-1-4503-5712-8. Citado 2 vezes nas páginas 31 e 34.

LATIF, M. et al. Cross platform approach for mobile application development: A survey. In: *2016 International Conference on Information Technology for Organizations Development (IT4OD)*. [S.l.: s.n.], 2016. p. 1–5. Citado 2 vezes nas páginas 26 e 27.

LATIF, M. et al. Review of mobile cross platform and research orientations. In: *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*. [S.l.: s.n.], 2017. p. 1–4. Citado na página 26.

LIM, S.-H. Experimental comparison of hybrid and native applications for mobile systems. *International Journal of Multimedia and Ubiquitous Engineering*, v. 10, p. 1–12, 03 2015. Citado 3 vezes nas páginas 32, 35 e 52.

M. S. Ferreira, C. et al. An evaluation of cross-platform frameworks for multimedia mobile applications development. *IEEE Latin America Transactions*, v. 16, n. 4, p. 1206–1212, April 2018. ISSN 1548-0992. Citado 2 vezes nas páginas 31 e 34.

MARTINEZ, M.; LECOMTE, S. Towards the quality improvement of cross-platform mobile applications. In: . [S.l.: s.n.], 2017. p. 184–188. Citado 3 vezes nas páginas 32, 36 e 37.

MCKAY, M. Accessibility challenges of hybrid mobile applications. In: . [S.l.: s.n.], 2017. p. 193–208. ISBN 978-3-319-58705-9. Citado 2 vezes nas páginas 32 e 33.

MEIRELLES, P. et al. A students' perspective of native and cross-platform approaches for mobile application development. In: _____. [S.l.: s.n.], 2019. p. 586–601. ISBN 978-3-030-24307-4. Citado 2 vezes nas páginas 31 e 36.

OPENSIGNAL. *The State of LTE(February 2018)*. 2018. Acesso em 19/10/2020. Disponível em: <https://www.opensignal.com/reports/2018/02/state-of-lte>. Citado na página 49.

PEAL, G. *Sunsetting React Native*. 2018. Acesso em 20/11/2019. Disponível em: <https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a>. Citado na página 22.

PINTO, C.; COUTINHO, C. From native to cross-platform hybrid development. In: . [S.l.: s.n.], 2018. p. 669–676. Citado na página 31.

PTITSYN, P.; RADKO, D. Analysis of cross-platform technologies for mobile applications development. v. 11, p. 11300–11307, 10 2016. Citado 2 vezes nas páginas 32 e 34.

QUE, P.; GUO, X.; ZHU, M. A comprehensive comparison between hybrid and native app paradigms. In: . [S.l.: s.n.], 2016. p. 611–614. Citado 4 vezes nas páginas 31, 33, 34 e 49.

SANTOS, D. S. dos et al. Recommendation system for cross-platform mobile development framework. In: *Proceedings of the XV Brazilian Symposium on Information Systems*. New York, NY, USA: ACM, 2019. (SBSI'19), p. 69:1–69:8. ISBN 978-1-4503-7237-4. Disponível em: <http://doi.acm.org/10.1145/3330204.3330279>. Citado na página 30.

SLASHDATA. *Developer Economics Community*. 2019. Acesso em 19/11/2019. Disponível em: <https://www.developereconomics.com/resources/graphs/>. Citado na página 21.

SLASHDATA. *Developer Economics State of The Developer Nation 17th Edition*. [S.l.], 2019. Citado 2 vezes nas páginas 21 e 55.

SMUTNý, P. Mobile development tools and cross-platform solutions. In: *Proceedings of the 2012 13th International Carpathian Control Conference, ICCC 2012*. [S.l.: s.n.], 2012. Citado na página 25.

STATCOUNTER. *Mobile Operating System Market Share Worldwide - October 2019*. 2019. Acesso em 19/11/2019. Disponível em: <http://gs.statcounter.com/os-market-share/mobile-tablet/worldwide>. Citado na página 21.

STATCOUNTER. *Operating System Market Share Worldwide*. 2019. Acesso em 12/11/2019. Disponível em: <https://gs.statcounter.com/os-market-share>. Citado na página 25.

STATISTA. *Number of available apps at Google Play from 2nd quarter 2015 to 3rd quarter 2019*. 2019. Acesso em 12/11/2019. Disponível em: <https://www.statista.com/statistics/289418/number-of-available-apps-in-the-google-play-store-quarter/>. Citado na página 25.

STATISTA. *Number of available apps in the Apple App Store from 2008 to 2019*. 2019. Acesso em 12/11/2019. Disponível em: <https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/>. Citado na página 25.

STATISTA. *Number of mobile app downloads worldwide from 2016 to 2019 (in billions)*. 2020. Acesso em 31/10/2020. Disponível em: <https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/>. Citado na página 21.

STATISTA. *Number of smartphone users worldwide from 2016 to 2021 (in billions)*. 2020. Acesso em 31/10/2020. Disponível em: <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>. Citado na página 21.

UMUHOZA, E.; BRAMBILLA, M. Model driven development approaches for mobile applications: A survey. In: YOUNAS, M. et al. (Ed.). *Mobile Web and Intelligent Information Systems*. Cham: Springer International Publishing, 2016. p. 93–107. ISBN 978-3-319-44215-0. Citado na página 27.

VILCEK, T.; JAKOPEC, T. Comparative analysis of tools for development of native and hybrid mobile applications. In: . [S.l.: s.n.], 2017. p. 1516–1521. Citado 2 vezes nas páginas 32 e 36.

WILLOCX, M.; VOSSAERT, J.; NAESSENS, V. A quantitative assessment of performance in mobile app development tools. In: . [S.l.: s.n.], 2015. p. 454–461. Citado 6 vezes nas páginas 32, 33, 34, 35, 41 e 47.

WILLOCX, M.; VOSSAERT, J.; NAESSENS, V. Comparing performance parameters of mobile app development strategies. In: . [S.l.: s.n.], 2016. p. 38–47. Citado 6 vezes nas páginas 32, 33, 34, 36, 49 e 52.

Glossário

build versão compilada de um *software*. 49

fintech é uma empresa que oferece serviços financeiros, com o diferencial de utilizar novas tecnologias e inovações nos seus processos para aumentar eficiência e tornar os serviços mais acessíveis ao público. 22

framework é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica. Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação. Ao contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação. 21

plugin extensão de um programa de *software*. 26

release por tradução direta, lançamento. No contexto de um aplicativo, a versão *release* se refere a *build* com código binário otimizado, gerada sem os dados necessários para *debugging*. 49

toolkit conjunto de elementos básicos de interface de usuário gráfica. 43

Apêndices

APÊNDICE A – Artigo

Análise do desenvolvimento de aplicativos mobile nativos e multiplataforma

Juliana Silva Pinheiro[1], **Raul Sidnei Wazlawick**[2]

[1] Departamento de Informática e Estatística
Universidade Federal de Santa Catarina – Florianópolis, SC – Brazil

juliana.pinheiro@grad.ufsc.br, raul.wazlawick@ufsc.br

Abstract. *Nowadays, Android and iOS dominate the market of mobile operating systems. Since both have different native programming languages, developing applications for both systems could require twice the amount of effort and human resources, and twice the amount of code. To avoid that, multi-platform frameworks and development tools were created. For this work, three versions of a mobile application (native and multi-platform) were developed and compared. Although the native versions demonstrate some performance advantages, the framework obtained good response times results, in addition to allowing access to native functionalities, having good documentation and high code reuse.*

Resumo. *Atualmente, os sistemas operacionais Android e iOS dominam o mercado de dispositivos móveis. Como ambos possuem linguagem de programação nativa diferentes, desenvolver aplicações para os dois sistemas necessitaria de possivelmente o dobro de esforço e recursos humanos, e o dobro de código. Para evitar o retrabalho, surgem os frameworks e ferramentas de desenvolvimento multiplataforma. Neste trabalho foram desenvolvidas e comparadas três versões de um aplicativo (nativas e multiplataforma). Apesar das versões nativas demonstrarem algumas vantagens de performance, o framework obteve bons resultados de tempo de resposta, além de possibilitar acesso as funcionalidades nativas, ter boa documentação e alto reuso de código.*

1. Introdução

Atualmente os sistemas operacionais Android[1] e iOS[2] detêm juntos 98,82% do mercado de dispositivos móveis (celulares e tablets) [StatCounter 2019], sendo assim o principal foco do desenvolvimento de aplicativos[SlashData 2019a]. Entretanto, esses sistemas possuem cada um suas particularidades, além de linguagens de programação nativas e SDKs diferentes, o que demanda que o desenvolvimento de aplicações para ambos os sistemas, necessário para alcançar a maior parte do público, seja feito separadamente. Isto é, se faz necessário desenvolver duas versões do mesmo aplicativo, em linguagens diferentes, duplicando a quantidade de esforço, custo e/ou tempo para o desenvolvimento. Além do desenvolvimento inicial, a manutenção de dois projetos de forma a mantê-los iguais (versionamento, mesmas funcionalidades novas, entre outros) também necessita de mais esforço comparado a apenas um projeto.

[1] https://www.android.com/
[2] https://www.apple.com/ios/

Como solução para essa questão surgiram *framework* e ferramentas de desenvolvimento multiplataforma, também referido como híbrido. Essas ferramentas tem como objetivo utilizar uma única base de código para todas as plataformas alvo, diminuindo o esforço necessário para desenvolver os aplicativos. Para alcançar esse objetivo, as ferramentas implementam métodos diferentes, explicitados no capítulo 3 deste trabalho.

Segundo o relatório *Developer Economics State of The Developer Nation 17th Edition* [SlashData 2019b], numa pesquisa com 1.189 desenvolvedores *mobile*, 40% dos desenvolvedores profissionais responderam ter utilizado algum *framework* multiplataforma nos últimos 12 meses. Dentre os desenvolvedores que utilizaram alguma ferramenta multiplataforma, aproximadamente 1 em 4 disseram utilizar React Native[3]. Com pelo menos 10% das resposta cada, foram citados também Xamarin[4], Flutter[5] e Ionic[6], mostrando um cenário relativamente segmentado.

Flutter foi lançado recentemente, em 2017, e pode ser uma boa alternativa aos *frameworks* e ferramentas multiplataforma existentes. Neste trabalho será comparado o desenvolvimento multiplataforma utilizando esse *framework* com o desenvolvimento nativo, após ter sido identificada a falta de estudos publicados na revisão bibliográfica apresentada no capítulo 3.

1.1. Objetivos

O objetivo geral é comparar o método de desenvolvimento nativo com o desenvolvimento multiplataforma, implementando uma aplicação com versões nativas para iOS e Android e versão em Flutter.

Os objetivos específicos são (1) identificar os métodos de desenvolvimento para plataformas móveis (celulares e tablets), e seus desafios ou dificuldades, (2) definir e implementar uma aplicação para plataformas móveis com versões nativas para iOS e Android, e versão multiplataforma utilizando Flutter, (3) comparar as versões do aplicativo desenvolvidas pelas critérios de consumo de CPU, (4) consumo de memória, (5) tamanho da aplicação, (6) tempo de resposta, (7) acesso a funcionalidades nativas, (8) qualidade da documentação e (9) reuso de código.

2. Método de Pesquisa

Este trabalho seguiu as seguintes etapas:

1. Revisão bibliográfica com o objetivo de identificar os métodos de desenvolvimento de aplicativos multiplataforma e *frameworks* ou ferramentas nessa área ainda não abordados ou pouco abordados, e trabalhos correlatos que realizaram comparações entre os diferentes métodos de desenvolvimento.
2. Definição da aplicação a ser implementada para comparação do desenvolvimento nativo com o desenvolvimento multiplataforma.
3. Desenvolvimento de três versões do aplicação: nativa para iOS, nativa para Android e multiplataforma utilizando Flutter.

[3] `https://facebook.github.io/react-native/`
[4] `https://dotnet.microsoft.com/apps/xamarin`
[5] `https://flutter.dev/`
[6] `https://ionicframework.com/`

4. Comparação das versões por critérios de consumo de CPU, consumo de memória, tamanho da aplicação, tempo de resposta, acesso a funcionalidades nativas, qualidade da documentação e reuso de código.
5. Elaboração do relatório com os resultados encontrados.

3. Fundamentação Teórica

Nesta seção são apresentados conceitos relevantes no desenvolvimento móvel, incluindo os métodos disponíveis, segundo a literatura lida.

3.1. Desenvolvimento nativo

Aplicativos nativos são aqueles desenvolvidos utilizando a linguagem de programação específica do sistema operacional alvo [Smutný 2012]. Como citado anteriormente para os sistemas iOS e Android, cada sistema operacional móvel possui linguagens nativas, SDKs, APIs, IDEs e lojas diferentes. Essa característica cria a necessidade de desenvolver um mesmo aplicativo múltiplas vezes, uma vez para cada plataforma desejada, o que aumenta o custo, tempo e esforço de desenvolvimento para que um aplicativo novo seja disponibilizado em todas as lojas. Em contrapartida, aplicativos nativos têm melhor performance e total acesso a todas as funcionalidades do dispositivo, além de oferecer a interface de usuário nativa do sistema [El-Kassas et al. 2017].

3.2. Desenvolvimento multiplataforma

Segundo [El-Kassas et al. 2017], o conceito das soluções multiplataforma, referenciadas tanto como *cross-platform* quanto como *multi-platform* em inglês, é desenvolver um aplicativo uma vez e executá-lo em qualquer sistema operacional móvel. Inicialmente, o desenvolvimento de aplicativos multiplataforma era impraticável devido as diferenças de linguagem e SDK, como citado na seção anterior. Para contornar esse problema, vários métodos e ferramentas de desenvolvimento foram criados, todos sob o conceito de desenvolvimento multiplataforma. Importante ressaltar que alguns trabalhos e publicações fora da academia se referem ao desenvolvimento multiplataforma como desenvolvimento híbrido, porém, nesse trabalho optou-se por utilizar a taxonomia sugerida por [Biørn-Hansen et al. 2018]. Os autores categorizam as ferramentas em quatro métodos: híbrido, interpretado, *cross-compiled* e orientado a modelos.

3.2.1. Híbrido

O método híbrido utiliza tecnologias web, incluindo HTML, CSS e JavaScript na implementação da interface de usuário e da lógica do aplicativo. Isso é possível pois as ferramentas que implementam esse método criam um projeto nativo, instanciam um *browser* por meio de um componente *WebView* e executam o código da aplicação no *browser*. O acesso a funcionalidades nativas (sistema de notificações, GPS, câmera, sistema de arquivos, entre outras) é limitado e feito por meio de uma camada de abstração [Latif et al. 2016] (geralmente referida como *plugin*) oferecidas pelo *framework* escolhido. Sendo assim, os aplicativos consistem da junção do projeto nativo da plataforma alvo com o código base comum às plataformas. As ferramentas mais populares são PhoneGap, Ionic Framework e OnsenUI [Latif et al. 2017].

3.2.2. Interpretado

O método interpretado permite o uso de linguagens genéricas como JavaScript por meio de interpretadores nos dispositivos. As ferramentas diferem do método híbrido por não necessitarem do componente *WebView*, e conseguem acessar as APIs nativas (e, consequentemente, as funcionalidades nativas, como notificações, câmera, entre outras) por meio de uma camada de abstração. Os *frameworks* mais populares são React Native[Biørn-Hansen et al. 2019b], Appcelerator Titanium e NativeScript [Delia et al. 2019].

3.2.3. Cross-compiled

Nesse método, o código produzido na linguagem de programação escolhida pelo *framework* é compilado para código binário executável nativo de cada plataforma alvo, sem necessidade de camadas intermediárias como os métodos anteriores. O acesso às funcionalidades nativas nesse caso é feito por meio do SDK do *framework*. Por ser compilado para código binário nativo, a maior vantagem desse método sobre os anteriores é a performance[Latif et al. 2016]. O principal *framework* que implementa esse método é o Xamarin.

3.2.4. Orientado a modelos

O método orientado a modelos, MDD, se assemelha ao *cross-compiled* por usar geração automática de código nativo, porém partindo não de linguagens de programação mas de modelos da interface de usuário e da lógica de negócio descritos em UML ou outras linguagens DSL. Segundo [Biørn-Hansen et al. 2018], uma das filosofias do MDD, não exclusivo ao desenvolvimento de aplicativos, é permitir que indivíduos que não possuem conhecimento técnico de desenvolvimento de *software* mas que são especialistas na lógica de negócio da aplicação a ser desenvolvida possam modelar o sistema. Porém, uma pesquisa realizada por [Biørn-Hansen et al. 2019b] sobre as ferramentas de desenvolvimento multiplataforma na indústria não obteve nenhuma resposta citando o uso de MDD, o que pode indicar uma baixa adesão ao método pela comunidade de desenvolvimento.

4. Revisão Bibliográfica

Foi realizada uma revisão da literatura com o objetivo de identificar aspectos, questões ou desafios da área de desenvolvimento de aplicativos móveis, focado na comparação entre desenvolvimento nativo e desenvolvimento multiplataforma, que ainda não foram abordados, ou não foram abordados a contento.

O motor de busca escolhido foi o Scopus[7] e a *string* de busca utilizada foi ”(((”mobile application”) OR (”mobile development”)) AND (hybrid OR multi-platform OR cross-platform OR native))”. A primeira parte foca no desenvolvimento de aplicativos móveis, e a segunda limita a artigos que comentem o método de desenvolvimento,

[7] https://www.scopus.com/

entre nativo e multiplataforma ou híbrido. A busca também foi limitada a artigos publicados desde 2015.

A primeira etapa consistiu em analisar os 868 artigos recuperados pela busca, realizando a leitura dos títulos e diminuindo o escopo para 79 artigos. Na segunda etapa foram lidos os resumos, levando a uma seleção de 41 artigos. A etapa seguinte consistiu em ler os artigos na íntegra, resultando em uma seleção final de 33 artigos.

A partir da leitura dos artigos, foi identificada uma sugestão de trabalho futuro na revisão da literatura (em inglês, *survey*) realizada por [Biørn-Hansen et al. 2018]: o estudo da ferramenta Flutter como *framework* de desenvolvimento multiplataforma. A data de publicação recente do artigo, janeiro de 2019, e a extensa revisão realizada pelos autores eram boas indicações de que a ferramenta de fato ainda não fora abordada a contento, porém, como garantia, foram levantadas todas as ferramentas de desenvolvimento multiplataforma mencionadas pelos artigos lidos, com a limitação de artigos publicados a partir de 2017, ano de lançamento do *framework*.

No levantamento as ferramentas foram separadas em duas categorias: (1) ferramentas analisadas em artigos de comparação entre desenvolvimento nativo e desenvolvimento multiplataforma, que realizaram a implementação de alguma aplicação; e (2) ferramentas citadas apenas em revisões da literatura, revisões e outros artigos que realizaram estudo da ferramenta sem implementação. Flutter está na segunda categoria, tendo sido citado apenas no *survey* mencionado anteriormente, como sugestão de trabalho futuro, e por [dos Santos et al. 2019] como *framework* que deveria ter sido considerado no questionário apresentado pelo artigo. Após esse levantamento, e considerando o lançamento recente do *framework*, foi definido como objetivo deste trabalho o estudo da ferramenta, e a comparação do seu uso no desenvolvimento multiplataforma com o desenvolvimento nativo de aplicativos.

Para comparar os dois modos de desenvolvimento, foram levantados os critérios mais utilizados pelos artigos para comparação de *frameworks*, bem como problemas e desafios encontrados no desenvolvimento multiplataforma. Para isso foi feita uma segunda seleção dos artigos, retirando revisões da literatura, taxonomias e artigos que não mencionavam diretamente alguma dificuldade apresentada por um *framework* ou algum critério de análise, levando a 23 artigos. Foram encontrados mais de 40 itens, e selecionados os cinco mais citados.

4.1. Critérios de comparação

4.1.1. Performance

Performance é citada de diversas maneiras nos artigos, tanto como um conceito único, quanto pelos seus indicativos, sendo os mais citados: uso de memória, tamanho da aplicação, uso de CPU e tempo de inicialização. Cada um é abordado a seguir.

Consumo de memória

Consumo de memória se refere à quantidade de memória RAM alocada pela aplicação, sendo crucial em dispositivos de baixo custo, pois aplicativos se tornam lentos se não há memória suficiente, o que afeta a usabilidade dos mesmos [Ahti et al. 2016] [Biørn-Hansen et al. 2019a].

A quantidade de memória alocada varia conforme o ciclo de vida do aplicativo, e é interessante medi-la quando o aplicativo está em *background*, quando geralmente a aplicação necessita de menos RAM. [Willocx et al. 2015] observaram que Android aloca mais memória para um aplicativo comparado ao iOS, porém também libera consideravelmente mais memória ao mover um aplicativo para *background*. Os autores constataram também que enquanto há diferença de alocação de memória entre aplicativos nativos e multiplataforma quando comparados utilizando dispositivos Android de baixo e alto nível, o iOS apresenta comportamento semelhante entre os aplicativos, independente do dispositivo.

Já comparando entre métodos de desenvolvimento multiplataforma diferentes, [Willocx et al. 2016] verificaram que implementações do método *cross-compiled* se comportam semelhante a implementação nativa, tendo um leve crescimento no uso de memória em relação à nativa. Já aplicações que utilizam *frameworks* Javascript alocam muito mais memória, observado também por [Que et al. 2016].

Tamanho da aplicação

O tamanho da aplicação instalada é um fator importante no desenvolvimento móvel, visto que celulares tem memória persistente menor que computadores, especialmente dispositivos de baixo custo com recursos limitados. Assim, é desejável que aplicativos utilizem o menor espaço possível em disco [Ahti et al. 2016]. [Willocx et al. 2015], [Willocx et al. 2016] e [Jia et al. 2018] mensuraram o tamanho de aplicativos nativos e multiplataforma e observaram que aplicações nativas ocupam menos espaço em disco comparada a aplicações multiplataforma utilizando *frameworks* variados.

[Willocx et al. 2015] apontam também a importância do tamanho em disco do instalador da aplicação, em formato APK para Android e .ipa para iOS, visto que um instalador compacto é crucial para usuários que baixam aplicações por conexão de internet móvel. [Que et al. 2016] observaram que o tamanho do instalador também tem impacto no tempo de instalação - quanto maior, mais demorada a instalação do aplicativo.

Consumo de CPU

Consumo de CPU é a porcentagem da capacidade total de CPU de um dispositivo utilizada por uma aplicação durante um período de tempo. Segundo [Willocx et al. 2015], aplicações com alto uso de CPU podem impactar negativamente outros processos sendo executados no dispositivo. Os autores consideraram importante também mensurar o uso de CPU na inicialização de aplicações multiplataforma, que podem apresentar algum *overhead* adicional pela tecnologia utilizada pelo *framework*.

Os resultados apresentados nos artigos de [Willocx et al. 2015], [Willocx et al. 2016], [Que et al. 2016] indicam um maior consumo de CPU por aplicações multiplataformas comparadas às nativas. Entretanto, [Biørn-Hansen et al. 2019a] encontraram resultados divergentes dos citados, num estudo focado em animações nos aplicativos. Nesse caso, as ferramentas Ionic e Xamarin tiveram consumo de CPU inferior as implementações nativas, em iOS e em Android, respectivamente.

Tempo de inicialização

Tempo de inicialização é uma importante medida de performance e também de

experiência do usuário. Segundo [Ahti et al. 2016], como aplicações móveis muitas vezes são utilizadas por um curto período de tempo antes de serem terminadas ou passadas para segundo plano, é essencial que iniciem rapidamente.

[Ptitsyn and Radko 2016] citam que aplicações implementadas com a ferramenta Titanium, do método interpretado, tem um tempo de inicialização maior que o nativo, pois exige processamento de código adicional antes de executar a aplicação. [M. S. Ferreira et al. 2018] confirma esse fato com experimentos com Titanium, PhoneGap e Sencha Touch. Titanium teve resultado inferior em ambos os sistemas operacionais, enquanto Sencha Touch e PhoneGap apresentaram resultados melhores que a aplicação nativa em Android.

4.2. User experience

Experiência do usuário, mais comumente referida pelo seu termo em inglês *user experience* ou ainda pela sigla UX, é o segundo critério mais citado pelos artigos lidos, considerado por [Ahti et al. 2016] o aspecto mais importante de uma aplicação. Questionários realizadas por [Ahmad et al. 2018] e [Biørn-Hansen et al. 2019b], com 34 e 101 desenvolvedores participantes, respectivamente, indicam que *user experience* é um dos maiores desafios no desenvolvimento móvel, e mais crítico no desenvolvimento multiplataforma em comparação ao nativo. Foi, inclusive, um dos motivos que levou o Facebook a abandonar, em 2012, o desenvolvimento híbrido dos seus aplicativos, retornando, naquela época, ao desenvolvimento nativo (importante ressaltar que o *framework* React Native, do método interpretado, foi criado pela empresa 3 anos depois[Facebook 2012a] [Facebook 2012b]).

Já é um consenso a importância desse aspecto das aplicações móveis, porém, não há uma definição única do conceito e do melhor modo de mensurá-lo. [Delia et al. 2019] define UX como um conjunto de fatores relacionados à satisfação do usuário ao utilizar o produto, alguns deles sendo tempo de resposta, design da interface de usuário, usabilidade e similaridade com os padrões do sistema operacional. Ao comparar implementações nativas e multiplataforma, o autor realizou uma pesquisa com participantes utilizando os aplicativos, por considerar *user experience* um aspecto subjetivo. [Ahti et al. 2016] consideram difícil caracterizar em termos de medidas quantitativas, e em seu artigo focam na velocidade de resposta de uma aplicação aos gestos do usuário e em que tipo de visão holística a aplicação dá ao usuário. Similarmente, [Willocx et al. 2015] consideraram como métricas de UX em seus experimentos diferentes tempos de resposta (navegação entre páginas, iniciar, pausar e retomar o aplicativo), além do uso de CPU, uso de memória e uso de bateria.

Por fim, outro consenso entre os artigos que apresentaram experimentos foi a excelência dos aplicativos nativos no quesito *user experience*, contra resultados variados nas aplicações multiplataforma, dependendo muito do *framework* escolhido.

4.3. Acesso a funcionalidades nativas

Segundo [Delia et al. 2019], o acesso a funcionalidades nativas, ou ainda, funcionalidades específicas do dispositivo, consiste na disponibilidade de acesso a câmeras, sensores, notificações, calendários, entre outros recursos do dispositivo. Como visto por [Lim 2015] e [Delia et al. 2015], a abordagem híbrida necessita de *plugins* externos para

obter esse acesso, e o primeiro autor ainda observa que a camada de *browser engine* (mecanismo de renderização de um navegador *web*) do *framework* gera *overhead*, isto é, aumenta o tempo de acesso às funcionalidades.

[Botella et al. 2016] levantam também a importância de comparar a compatibilidade de componentes entre plataformas, isto é, um componente, disponibilizado pelo *framework* ou como um *plugin*, para acessar um recurso do dispositivo pode funcionar em apenas uma das plataformas. A partir de experimentos, o autor cita o Ionic como tendo bons resultados nesse critério, observado também por [Delia et al. 2019].

4.4. Documentação

[Vilcek and Jakopec 2017], [Brito et al. 2018] e [Delia et al. 2019] consideram uma boa documentação do *framework* importante pois diminui a curva de aprendizado e torna o desenvolvimento mais fácil e viável. A pesquisa realizada por [Meirelles et al. 2019] com estudantes e profissionais da área de desenvolvimento também indica a qualidade da documentação como um critério decisivo para a escolha de uma ferramenta.

Um exemplo citado de *framework* bem documentado foi o React Native, por [Martinez and Lecomte 2017] e por [Brito et al. 2018]. Porém, é importante ressaltar que nenhum dos artigos definiu um método empírico de mensurar a qualidade ou quantidade de documentação, tendo sido feito de forma subjetiva.

4.5. Reuso de código

Como exposto na seção de fundamentação teórica, no método de desenvolvimento nativo não existe compartilhamento, ou reuso, de código entre as plataformas-alvo das aplicações móveis. Isso foi possível com a criação dos *frameworks* multiplataforma, que permitem reutilizar parcial ou totalmente a base de código de um aplicativo.

Segundo [Martinez and Lecomte 2017] e [Delia et al. 2019], o compartilhamento de código entre plataformas reduz consideravelmente os custos de desenvolvimento e manutenção. [Delia et al. 2019] também ressaltam a importância da porcentagem de reuso que uma ferramenta consegue alcançar, por variar muito de método para método. [Willocx et al. 2016] citam que alguns *frameworks* que utilizam Javascript apresentam 100% de reuso de código, com a desvantagem de não ter acesso a todas as funcionalidades nativas. Ele segue comentando que outras ferramentas, como Titanium[8], oferecem uma interface gráfica uniforme entre plataformas mas ao mesmo tempo permitem uso de código específico para acesso de funcionalidades de cada plataforma, enquanto o Xamarin[9] implementa interface gráfica diferenciada para cada plataforma, compartilhando o resto do código. A porcentagem de reuso de código no Xamarin varia muito conforme observado entre os artigos, indo de 50% de compartilhamento [Delia et al. 2015] até 93% [Martinez and Lecomte 2017] .

5. Planejamento do experimento

Nesta seção são apresentadas as definições referentes ao experimento realizado para comparar o desenvolvimento nativo com o uso do *framework* Flutter.

[8] `https://www.appcelerator.com/titanium/titanium-sdk/4/`
[9] `https://dotnet.microsoft.com/apps/xamarin`

5.1. Definição do aplicativo

A fim de comparar o desenvolvimento nativo de aplicativos com o desenvolvimento multiplataforma, foi definido um aplicativo a ser implementado em três versões: (1) nativa para Android, (2) nativa para iOS, e (3) multiplataforma (para Android e iOS), utilizando Flutter. O aplicativo foi criado para ser de simples implementação, descrito por uma lista de requisitos funcionais.

1. O usuário pode registrar uma tarefa, informando uma breve descrição da tarefa, e, opcionalmente, uma data de lembrete.
2. O sistema deve mostrar uma lista com todas as tarefas criadas, ordenadas por realizadas e não realizadas.
3. O usuário pode informar se uma tarefa foi realizada.
4. O sistema deve notificar o usuário na data de lembrete, caso tenha sido definida, sobre uma tarefa ainda não realizada.
5. O usuário pode excluir uma tarefa da lista de tarefas.
6. O usuário pode registrar uma anotação, inserindo texto e/ou imagens, tiradas pela câmera do dispositivo ou selecionadas na galeria de fotos do mesmo.
7. O sistema deve mostrar uma lista de todas as anotações criadas, ordenadas por data de criação.
8. O usuário pode excluir uma anotação da lista.
9. O sistema deve garantir persistência de todos os dados cadastrados, isto é, deve manter os dados salvos após entrar em segundo plano ou ser fechado.

5.2. Definição das métricas

Para comparar as três versões do aplicativo, foram definidas métricas relacionadas aos critérios encontrados na seção 4.

Métrica 1.1: consumo de memória RAM ao abrir o aplicativo, em megabytes.

Métrica 1.2: consumo de memória RAM pelo aplicativo em segundo plano, em megabytes.

Métrica 1.3: consumo máximo de memória RAM durante o uso do aplicativo,em megabytes.

Métrica 2.1: espaço em disco utilizado pelo aplicativo, em megabytes.

Métrica 2.2: espaço em disco utilizado pelo instalador do aplicativo, em megabytes.

Métrica 3.1: consumo de CPU ao iniciar o aplicativo, em porcentagem sobre a capacidade total do dispositivo.

Métrica 3.2: consumo máximo de CPU durante o uso do aplicativo, em porcentagem sobre a capacidade total do dispositivo

Métrica 4.1: tempo de inicialização do aplicativo, em milissegundos.

Métrica 4.2: tempo para retomar o aplicativo, quando em segundo plano, em milissegundos.

Métrica 4.3: tempo de navegação entre páginas do aplicativo, em milissegundos.

Métrica 5: acesso às funcionalidades nativas necessárias (câmera, sistema de notificações e sistema de arquivos).

Métrica 6: reuso do código, em porcentagem de código escrito utilizado para ambas as plataformas sobre o total (não considera código gerado).

Métrica 7: qualidade da documentação, conforme a disponibilidade de três elementos:

1. Trechos de código, curtos e comentados, para compreender uma funcionalidade básica de um componente da API;
2. Tutoriais com passo-a-passo para compreender como desenvolver uma funcionalidade com vários componentes da API;
3. Manual de referência baixo nível documentando todos os componentes da API.

O critério performance foi dividido em 4 conjuntos de métricas diferentes, relacionadas a consumo de RAM, consumo de CPU, tamanho da aplicação em disco, e tempo de inicialização, como visto no capítulo anterior. Devido a falta de consenso quanto ao critério de *user experience* e ao escopo do trabalho, esse critério foi relacionado ao tempo de resposta, tempo de inicialização e tempo de retomada, como sugerido por [Delia et al. 2019]. Estão relacionados também consumo de memória, que pode tornar a a aplicação lenta, e consumo de CPU, que pode impactar negativamente outros processos executados no dispositivos, ambos piorando a experiência do usuário [Ahti et al. 2016][Willocx et al. 2015].

Como comentado na seção 4.5, nenhum dos artigos de comparação de *frameworks* utilizou um método para mensurar a qualidade da documentação destes. Buscou-se na literatura, um modo de avaliar a qualidade de documentação de APIs. Numa revisão da literatura, [Cummaudo et al. 2019] levantaram os três elementos mais citados necessários para uma documentação de API de qualidade, definidos em 1, 2 e 3.

5.3. Dispositivos utilizados

Para obter os resultados, os aplicativos desenvolvidos foram instalados em dois dispositivos *smartphone*, com sistemas operacionais diferentes. Em cada dispositivo foi instalada a versão nativa apropriada do aplicativo, e a versão em Flutter foi instalada em ambos.

Dispositivo	iOS	Android
Modelo	iPhone 8	Galaxy A10s
Sistema operacional	iOS 13	Android 10
Memória RAM	2GB	2GB
CPU	2.39 GHz hexa-core 64-bit	1.8GHz hecta-core
Armazenamento	64GB	32GB

Table 1. Dispositivos utilizados para análises de performance

5.4. Ferramentas utilizadas

Algumas ferramentas foram utilizadas para obter os resultados referentes as métricas 1 a 4, como pode ser visto na tabela 2.

Métrica	Ferramentas
1.1, 1.2 e 1.3	Android Profiler e Xcode
2.1 e 2.2	Visível no dispositivo
3.1 e 3.2	Android Profiler e Xcode
4.1, 4.2 e 4.3	Android Profiler, Flutter Driver e Xcode UI Testing

Table 2. Relação entre métricas obtidas e ferramentas utilizadas

6. Desenvolvimento

Nesta seção é apresentado o desenvolvimento das três versões do aplicativo proposto no capítulo anterior.

6.1. Flutter

Flutter é um *toolkit* de código aberto para desenvolvimento de aplicativos multiplataforma, incluindo os sistemas Android e iOS, focado no desenvolvimento de interface de usuário. Tem como proposta possibilitar a criação de aplicativos visualmente agradáveis, diminuindo o tempo, o custo e a complexidade do desenvolvimento [Flutter 2019].

A ferramenta se destaca de outras no mercado de desenvolvimento multiplataforma de aplicativo por não usar *WebView* nem componentes nativos do sistema operacional alvo, sendo classificada no método *cross-compiled* por [Biørn-Hansen et al. 2018]. Para renderizar a interface, Flutter utiliza a *engine* de renderização 2D Skia[10], escrita em C/C++.

Para ser executado tanto em dispositivos iOS quanto Android, o código da *engine* em C/C++ é compilado com NDK ou LLVM, enquanto o código Dart tanto da aplicação quanto do SDK são compilados *ahead-of-time* para código de máquina da plataforma alvo, que é então incluído num projeto nativo para formar o instalador do aplicativo, .ipa para iOS e APK para Android.

6.2. Versões desenvolvidas

Versão do aplicativo	A1	A2	A3
SDK	Android SDK	Flutter SDK 1.22	iOS SDK
Plataforma	Android	Android, iOS	iOS
Linguagem	Kotlin	Dart	Swift 5.0
IDE	Android Studio	VS Code	Xcode

Table 3. Versões do aplicativo desenvolvidas

Buscou-se tomar decisões de arquitetura similares em todas as versões. Para garantir o requisito 9, de persistência de dados, todos os dados inseridos na aplicação pelo usuário são salvos em banco de dados relacional local, em SQLite. As SDKs nativas possuem bibliotecas para implementação de banco de dados, sendo elas Core Data para

[10] `https://skia.org/`

iOS e Room para Android. A versão em Flutter necessitou de um *plugin* externo, sqflite, desenvolvido e mantido pela comunidade. Importante relembrar que o requisito é apenas persistência local, e o aplicativo não necessita de acesso a Internet para funcionar.

Os requisitos 4 e 6 necessitam que o aplicativo acesse o sistema de notificações, a câmera e o sistema de arquivos do dispositivo. Ambos os sistemas operacionais exigem que a aplicação solicite ao usuário permissão para acessar essas funcionalidades, além da definição prévia nas configurações de cada projeto sobre quais serão utilizadas. Os SDKs nativos oferecem as APIs necessárias para realizar a solicitação e acesso. Já em Flutter, novamente são disponibilizados pela comunidade *plugins* a serem adicionados ao projeto, que abstraem o acesso as APIs nativas de cada recurso. Assim como no caso nativo, também é necessário definir nos arquivos de configuração, um para cada plataforma, quais funcionalidades serão utilizadas.

7. Resultados

Nesta seção são apresentados os resultados obtidos a partir do experimento definido na seção 5.

7.1. Consumo de memória RAM

Métrica 1.1

A primeira métrica se refere ao consumo de memória RAM ao iniciar o aplicativo. No dispositivo Android, o consumo foi similar entre as versões nativa e multiplataforma, enquanto no dispositivo iOS, a versão nativa obteve notável vantagem ao utilizar apenas 32% da memória consumida pela aplicação em Flutter.

	Android	iOS
Nativo	89,6	22,7
Flutter	93,2	69,4

Table 4. Métrica 1.1: consumo de memória RAM ao abrir o aplicativo, em megabytes.

Métrica 1.2

Como apontado por [Willocx et al. 2015], é interessante observar a diferença de alocação de memória para o aplicativo quando é movido para segundo plano, ou *background*. Foi possível observar o comportamento apontado pelos autores: ambas as versões nativa e multiplataforma consumiram mais memória no dispositivo Android em primeiro plano, porém liberaram mais memória ao serem movidas para segundo plano, comparadas as versões instaladas no dispositivo iOS.

	Android	iOS
Nativo	46,5	22,3
Flutter	75,9	67

Table 5. Métrica 1.2: consumo de memória RAM pelo aplicativo em segundo plano, em megabytes.

Métrica 1.3

Uma última métrica de consumo de RAM foi obtida realizando diversas ações no aplicativo. Em cada versão e dispositivo, as mesmas ações foram executadas, na mesma ordem, e foi obtido o maior valor de consumo de memória. As ações foram: *scroll* pela lista de tarefas da tela inicial, criar uma tarefa nova, inserir data de lembrete na tarefa, abrir a aba de anotações e realizar *scroll* pela lista, criar anotação nova, obter foto pela câmera do dispositivo e adicionar na anotação, voltar para a lista de anotações e reabrir a anotação com foto.

	Android	iOS
Nativo	290,8	155,9
Flutter	250,3	148

Table 6. Métrica 1.3: consumo máximo de memória RAM durante o uso do aplicativo, em megabytes.

Em todas as versões e dispositivos, o pico no consumo de memória ocorreu ao executar a ação de adicionar a foto na anotação. É possível notar que, apesar de nas outras métricas a versão multiplataforma ter obtido desvantagem em relação a versão nativa no dispositivo iOS, nesse caso ambas obtiveram resultados muito próximos, com Flutter mostrando uma leve vantagem.

7.2. Espaço em disco

Métrica 2.1

Como visto na revisão bibliográfica, é esperado que versões multiplataforma ocupem mais espaço em disco do que versões nativas. Os resultados encontrados variaram drasticamente entre plataformas. Para Android, a versão multiplataforma, teve 2,65 vezes o tamanho da versão nativa, porém mantendo um tamanho ainda razoável, de 26,12MB.

	Android	iOS
Nativo	9,95	1,2
Flutter	26,12	254,9

Table 7. Métrica 2.1: espaço em disco utilizado pelo aplicativo, em megabytes.

Já na plataforma iOS, a diferença entre os tamanhos foi excepcional. Enquanto a versão nativa ocupa ínfimos 1,2MB, a versão multiplataforma chegou ao tamanho de 254,9MB. Devido a grande disparidade entre os valores, buscou-se entender a causa. O Flutter SDK disponibiliza uma ferramenta de linha de comando que informa um relatório do espaço utilizado por cada componente da aplicação. A partir do uso dessa ferramenta foi possível observar que 87,74% do tamanho da aplicação final consiste no próprio *framework*.

Para ambas as plataformas, foram instaladas as *build* versão *release* dos aplicativos. Porém, é possível que ao publicar o aplicativo nas respectivas lojas e realizar a instalação a partir delas, o tamanho final varie. Devido ao custo de se publicar um aplicativo na App Store, foram analisados apenas os tamanhos das *builds* geradas localmente.

Métrica 2.2

Também foram obtidos os tamanhos dos instaladores, como sugerido por [Willocx et al. 2016] e [Que et al. 2016]. Entretanto, não é possível gerar instaladores .ipa, utilizados pelos dispositivos iOS, sem uma conta no Apple Developer Program[11]. Novamente devido ao custo monetário para obter a conta, não foi possível obter valores referentes às versões nativa e multiplataforma para iOS.

	Android	iOS
Nativo	3,7	N/A
Flutter	8,4	N/A

Table 8. Métrica 2.2: espaço em disco utilizado pelo instalador, em megabytes.

Apesar do tamanho do instalador da versão em Flutter ser o dobro da versão nativa, ainda é um valor aceitável, considerando que a média de velocidade de download das redes de internet móvel no Brasil era 19,67MB/s em 2018 [OpenSignal 2018].

7.3. Consumo de CPU

Métrica 3.1

O consumo de CPU foi medido ao iniciar os aplicativos. Nesse critério, as versões nativas novamente mostraram resultados superiores ao *framework*, semelhante ao encontrado por [Willocx et al. 2016] e [Que et al. 2016] ao comparar o desenvolvimento nativo com outras ferramentas. Os comportamentos das versões nativas foram similares, assim como os comportamentos da versão Flutter em cada dispositivo.

	Android	iOS
Nativo	23%	22%
Flutter	28%	31%

Table 9. Métrica 3.1: consumo de CPU ao iniciar o aplicativo, em porcentagem sobre a capacidade total do dispositivo.

Métrica 3.2

Esta métrica é similar a métrica 1.3, consistindo no consumo máximo de um recurso durante o uso do aplicativo, dessa vez de CPU. As mesmas ações foram realizadas em todas as versões, e foi obtido o maior valor de consumo de CPU. As versões mantiveram consumo relativamente baixo no dispositivo Android, com os picos ocorrendo durante a navegação entre páginas e ao realizar *scroll* na lista de anotações, na versão nativa e Flutter respectivamente. Já no dispositivo iOS, o consumo foi consideravelmente mais elevado, e ocorreu ao realizar a ação de adicionar foto, em ambas as versões.

7.4. Tempo de resposta

Métrica 4.1

A métrica 4.1 consiste no tempo de inicialização do aplicativo. Foi coletado o tempo de inicialização logo após instalar o aplicativo, sem dados salvos no banco de dados. Como a tela inicial é uma lista de tarefas, carregadas do banco de dados da aplicação,

[11] `https://developer.apple.com/programs/`

	Android	iOS
Nativo	32%	72%
Flutter	41%	86%

Table 10. Métrica 3.2: consumo máximo de CPU durante o uso do aplicativo, em porcentagem sobre a capacidade total do dispositivo.

para simular uma situação de uso mais próxima da real cada versão do aplicativo foi populada com a mesma quantidade de tarefas, e novamente foram obtidos os tempos de inicialização.

	BD vazio		BD populado	
	Android	iOS	Android	iOS
Nativo	1.800	1.110	1.832	1.239
Flutter	1.060	1.254	1.765	1.412

Table 11. Métrica 4.1: tempo de inicialização do aplicativo, em milissegundos.

Os tempos obtidos foram de grandezas iguais, comparando entre versão Flutter e nativa. O *framework* até mesmo obteve vantagem sobre a versão Android. Porém teve um salto maior que a versão nativa ao carregar dados do banco para a lista, o que pode indicar que consultas ao banco sejam mais lentas na versão multiplataforma. Os resultados no dispositivo iOS foram diferentes. A versão nativa manteve uma vantagem de aproximadamente 12% sobre a multiplataforma em ambos os cenários de teste.

Métrica 4.2

A próxima métrica está relacionada ao estado de segundo plano pelo qual as aplicações podem passar durante seu ciclo de vida no dispositivo, isto é, um aplicativo pode ser suspendido durante o uso, mantendo alguns dados na memória, e retomado posteriormente por opção do usuário. Os testes foram realizados nos dispositivos com as bases de dados populadas.

	Android	iOS
Nativo	823	997
Flutter	651	647

Table 12. Métrica 4.2: tempo para retomar o aplicativo, quando em segundo plano, em milissegundos.

Nesse ponto, o *framework* obteve resultados melhores em ambos os dispositivos, demonstrando maior vantagem na plataforma iOS, ao utilizar apenas 64% do tempo gasto pela aplicação nativa para retomar o aplicativo.

Métrica 4.3

Essa métrica se refere ao tempo para o aplicativo navegar de uma tela para outra. Para os testes, esse cálculo foi feito ao abrir a tela de criar anotação. Em ambas as plataformas o *framework* obteve vantagem, em especial no dispositivo iOS.

	Android	iOS
Nativo	283,77	999
Flutter	163,82	52,17

Table 13. Métrica 4.3: tempo de navegação entre páginas do aplicativo, em milissegundos.

7.5. Acesso a funcionalidades nativas

Métrica 5

Como comentado no capítulo de desenvolvimento, os requisitos da aplicação requerem acesso ao sistema de notificações, câmera e sistema de arquivos do dispositivo, disponibilizado por APIs do SDK de cada plataforma, no desenvolvimento nativo.

Plugin	**Funcionalidade**	**Plataformas compatíveis**
Image Picker[12]	Câmera e galeria de fotos	Android, iOS
Path Provider[13]	Sistema de arquivos	Android, iOS
Flutter Local Notifications[14]	Sistema de notificações	Android, iOS

Table 14. *Plugins* utilizados para acesso a funcionalidades nativas

Já no desenvolvimento multiplataforma, foi possível acessar todos os recursos por meio de *plugins* adicionados ao projeto Flutter, que abstraem a camada de comunicação com as APIs nativas, similar ao comportamento relatado por [Lim 2015] em outros *frameworks*. Todos os *plugins* utilizados eram compatíveis com ambas as plataformas, necessitando de pouco código específico para cada uma, e a aplicação conseguiu satisfazer os requisitos da mesma forma que as nativas.

7.6. Reuso de código

Métrica 6

Quanto a métrica de reuso de código, as versões nativas, por definição, não possibilitam compartilhamento do código entre as plataformas. Já o *framework* Flutter permite escrever aplicações inteiras uma vez e utilizá-las em ambas as plataformas. A versão desenvolvida tem apenas 2,6% de código específico para uma plataforma, do total de código escrito (não foi considerado para o percentual os arquivos de código gerado pela *framework* nem bibliotecas), e essas linhas de código foram necessárias apenas para configurar o acesso ao sistema de notificação dos dispositivos.

Esse resultado pode ser comparado ao encontrado por [Willocx et al. 2016], ao citarem que *frameworks* que apresentam 100% de reuso de código têm como desvantagem a ausência de acesso a funcionalidades do dispositivo. De fato, caso a aplicação desenvolvida não necessitasse de acesso ao sistema de notificações, teria o código totalmente compartilhado entre plataformas. Porém, ressalta-se aqui que os *plugins* para Flutter abstraem muito da camada de comunicação com as APIs nativas, diminuindo ou

[2] `https://pub.dev/packages/image_picker`
[3] `https://pub.dev/packages/path_provider`
[4] `https://pub.dev/packages/flutter_local_notifications`

até eliminando a necessidade de código específico em alguns caso, como o do *plugin* para acesso a câmera do dispositivo.

7.7. Qualidade da documentação

Métrica 7

Foi realizada a comparação da documentação oferecida pela Google para desenvolvimento Android, a documentação para desenvolvimento iOS disponibilizada pela Apple, e a documentação do *framework* Flutter, também disponibilizada pela Google, utilizando os três elementos definidos na seção 5.

A Google disponibiliza para desenvolvimento nativo Android, além da documentação das classes e métodos das APIs, tutoriais para uso das bibliotecas nativas e trechos de código de exemplo para todos os componentes utilizados no desenvolvimento deste trabalho.

Já no caso da plataforma iOS, a documentação apresentava poucos tutoriais, que não abrangiam a maioria das funcionalidades da nova biblioteca gráfica nativa, SwiftUI, além de não apresentar nenhum trecho de código para os componentes individualmente. É uma documentação muito mais crua, no sentido de apresentar apenas a definição de cada componente e seus métodos.

	Android nativo	iOS nativo	Flutter
E1	✓		✓*
E2	✓		✓*
E3	✓	✓	✓

Table 15. Elementos presentes em cada documentação Legenda: * = varia entre *plugins*.

Flutter apresentou uma documentação mais completa. Todos os *widgets* têm seus métodos e atributos documentados, além de uma breve descrição do componente, e na maioria dos casos trechos de código de exemplo. São disponibilizados tutoriais oficiais demonstrando o uso de diversas funcionalidades (animações, listas, persistência, navegação, rede, entre outros).

Porém, o *framework* depende do uso de *plugins* externos para acessar funcionalidades do dispositivo, e essas ferramentas apresentam documentações separadas, de responsabilidade de cada autor, o que leva a alta variação na qualidade e disponibilidade. Para todos os *plugins*, a documentação oferecida apresentava o elemento E3, e apresentava pelo menos um dos outros dois elementos: trecho de código de exemplo ou tutorial passo-a-passo.

8. Conclusão

Com o crescimento do mercado de aplicativos para dispositivos móveis, surgem novas ferramentas para facilitar o desenvolvimento para múltiplas plataformas móveis. Este trabalho buscou comparar uma dessas ferramentas, o *framework* Flutter, com o desenvolvimento nativo. Foram levantados os critérios mais citados na revisão da literatura, e desenvolvidas três versões de um aplicativo, que foram, então, comparadas seguindo cada um desses critérios.

A performance das versões nativas, iOS e Android, se mostrou superior em algumas métricas, principalmente em tamanho da aplicação. Porém, a versão multiplataforma demonstrou vantagem nos tempos de resposta: inicialização mais rápida no dispositivo Android, e tempo para retomar a aplicação menor e navegação entre páginas mais rápida em ambos os dispositivos. Além disso, a aplicação em Flutter conseguiu acesso a todas as funcionalidades nativas necessárias (câmera, sistema de notificações e sistema de arquivos), com pouco código específico: 97,4% do código escrito era destinado a ambas as plataformas. A documentação do *framework* também se mostrou satisfatória, em especial comparada a documentação disponibilizada pela Apple.

O *framework* Flutter se mostrou como uma alternativa relevante no desenvolvimento *mobile*, pois além da vantagem intrínseca do desenvolvimento multiplataforma - uma única aplicação compatível com ambas as plataformas - conseguiu se equiparar e superar o desenvolvimento nativo em algumas métricas de performance, sem perda de funcionalidades nativas.

8.1. Trabalhos futuros

Como recomendação de trabalho futuro, sugere-se: a comparação do Flutter com React Native, outro *framework* de desenvolvimento multiplataforma, mais utilizado segundo relatório da [SlashData 2019b]; expandir os critérios de comparação, visto que vários outros foram levantados na literatura lida, porém com menos citações; desenvolvimento e avaliação de aplicações mais complexas utilizando Flutter, que tenham como requisito, por exemplo, conexão com a Internet.

9. References

References

Ahmad, A., Li, K., Feng, C., Syed, A., Yousif, A., and Ge, S. (2018). An empirical study of investigating mobile applications development challenges. *IEEE Access*, PP:1–1.

Ahti, V., Hyrynsalmi, S., and Nevalainen, O. (2016). An evaluation framework for cross-platform mobile app development tools: A case analysis of adobe phonegap framework.

Biørn-Hansen, A., Grønli, T.-M., and Ghinea, G. (2018). A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. *ACM Comput. Surv.*, 51(5):108:1–108:34.

Biørn-Hansen, A., Grønli, T.-M., and Ghinea, G. (2019a). Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance. *Sensors*, 19:2081.

Biørn-Hansen, A., Grønli, T.-M., Ghinea, G., and Alouneh, S. (2019b). An empirical study of cross-platform mobile development in industry. *Wireless Communications and Mobile Computing*, 2019:1–12.

Botella, F., Escribano, P., and Peñalver, A. (2016). Selecting the best mobile framework for developing web and hybrid mobile apps. pages 1–4.

Brito, H., Gomes, A., and Bernardino, J. (2018). Javascript in mobile applications: React native vs ionic vs nativescript vs native development. pages 1–6.

Cummaudo, A., Vasa, R., and Grundy, J. (2019). What should i document? a preliminary systematic mapping study into api documentation knowledge.

Delia, L., Galdámez, N., Thomas, P., Corbalan, L., and Pesado, P. (2015). Multi-platform mobile application development analysis. pages 181–186.

Delia, L., Thomas, P., Corbalan, L., Sosa, J. F., Cuitiño, A., Cáseres, G., and Pesado, P. (2019). Development approaches for mobile applications: Comparative analysis of features. In Arai, K., Kapoor, S., and Bhatia, R., editors, *Intelligent Computing*, pages 470–484, Cham. Springer International Publishing.

dos Santos, D. S., Nunes, H. D., Macedo, H. T., and Neto, A. C. (2019). Recommendation system for cross-platform mobile development framework. In *Proceedings of the XV Brazilian Symposium on Information Systems*, SBSI'19, pages 69:1–69:8, New York, NY, USA. ACM.

El-Kassas, W. S., Abdullah, B. A., Yousef, A. H., and Wahba, A. M. (2017). Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal*, 8(2):163–190.

Facebook (2012a). Under the hood: Rebuilding facebook for android. Acesso em 12/09/2020.

Facebook (2012b). Under the hood: Rebuilding facebook for ios. Acesso em 12/09/2020.

Flutter (2019). Faq - flutter. Acesso em 12/11/2019.

Jia, X., Ebone, A., and Tan, Y. (2018). A performance evaluation of cross-platform mobile application development approaches. pages 92–93.

Latif, M., Lakhrissi, Y., Nfaoui, E. H., and Es-Sbai, N. (2016). Cross platform approach for mobile application development: A survey. In *2016 International Conference on Information Technology for Organizations Development (IT4OD)*, pages 1–5.

Latif, M., Lakhrissi, Y., Nfaoui, E. H., and Es-Sbai, N. (2017). Review of mobile cross platform and research orientations. In *2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, pages 1–4.

Lim, S.-H. (2015). Experimental comparison of hybrid and native applications for mobile systems. *International Journal of Multimedia and Ubiquitous Engineering*, 10:1–12.

M. S. Ferreira, C., J. P. Peixoto, M., A. S. Duarte, P., B. B. Torres, A., L. Silva Junior, M., S. Rocha, L., and Viana, W. (2018). An evaluation of cross-platform frameworks for multimedia mobile applications development. *IEEE Latin America Transactions*, 16(4):1206–1212.

Martinez, M. and Lecomte, S. (2017). Towards the quality improvement of cross-platform mobile applications. pages 184–188.

Meirelles, P., Aguiar, C., Assis, F., Siqueira, R., and Goldman, A. (2019). *A Students' Perspective of Native and Cross-Platform Approaches for Mobile Application Development*, pages 586–601.

OpenSignal (2018). The state of lte(february 2018). Acesso em 19/10/2020.

Ptitsyn, P. and Radko, D. (2016). Analysis of cross-platform technologies for mobile applications development. 11:11300–11307.

Que, P., Guo, X., and Zhu, M. (2016). A comprehensive comparison between hybrid and native app paradigms. pages 611–614.

SlashData (2019a). Developer economics community. Acesso em 19/11/2019.

SlashData (2019b). Developer economics state of the developer nation 17th edition. Technical report.

Smutný, P. (2012). Mobile development tools and cross-platform solutions. In *Proceedings of the 2012 13th International Carpathian Control Conference, ICCC 2012*.

StatCounter (2019). Mobile operating system market share worldwide - october 2019. Acesso em 19/11/2019.

Vilcek, T. and Jakopec, T. (2017). Comparative analysis of tools for development of native and hybrid mobile applications. pages 1516–1521.

Willocx, M., Vossaert, J., and Naessens, V. (2015). A quantitative assessment of performance in mobile app development tools. pages 454–461.

Willocx, M., Vossaert, J., and Naessens, V. (2016). Comparing performance parameters of mobile app development strategies. pages 38–47.

APÊNDICE B – Código-fonte

B.1 Android

Arquivo: app/src/main/java/com/example/twonotes/dao/AppDataBase.kt

```
package com.example.twonotes.dao

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
import com.example.twonotes.model.Converters
import com.example.twonotes.model.Note
import com.example.twonotes.model.Todo

@Database(entities = [Todo::class, Note::class], version = 1)
@TypeConverters(Converters::class)
abstract class AppDatabase : RoomDatabase() {
    abstract fun todoDao(): TodoDao
    abstract fun noteDao(): NoteDao

    companion object {
        @Volatile private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context): AppDatabase {
            return INSTANCE ?: synchronized(this) {
                INSTANCE ?: buildDatabase(context).also { INSTANCE = it }
            }
        }

        private fun buildDatabase(context: Context): AppDatabase {
            return Room.databaseBuilder(context, AppDatabase::class.java,
                .build()
```

```
        }
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/dao/NoteDao.kt

```
package com.example.twonotes.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import androidx.sqlite.db.SupportSQLiteQuery
import com.example.twonotes.model.Note

@Dao
interface NoteDao {
    @Query("SELECT * FROM note ORDER BY editDate DESC")
    fun getAll(): LiveData<List<Note>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insert(note: Note): Long

    @Delete
    fun delete(note: Note)

    @Query("SELECT * FROM note WHERE id = :id LIMIT 1")
    fun findById(id: Int): Note?

    @RawQuery
    fun populate(query: SupportSQLiteQuery): Long
}
```

Arquivo: app/src/main/java/com/example/twonotes/dao/TodoDao.kt

```
package com.example.twonotes.dao

import androidx.lifecycle.LiveData
import androidx.room.*
import androidx.sqlite.db.SupportSQLiteQuery
```

```
import com.example.twonotes.model.Todo


@Dao
interface TodoDao {
    @Query("SELECT * FROM todo ORDER BY done, case when reminderDate is r
    fun getAll(): LiveData<List<Todo>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insert(todo: Todo): Long

    @Delete
    fun delete(todo: Todo)

    @Query("SELECT * FROM todo WHERE id == :id LIMIT 1")
    fun findById(id: Int): Todo?

    @RawQuery
    fun populate(query: SupportSQLiteQuery): Long
}
```

Arquivo: app/src/main/java/com/example/twonotes/data/NoteRepository.kt

```
package com.example.twonotes.data

import androidx.lifecycle.LiveData
import androidx.sqlite.db.SimpleSQLiteQuery
import com.example.twonotes.dao.NoteDao
import com.example.twonotes.model.Note

class NoteRepository(private val noteDao: NoteDao) {
    val notes: LiveData<List<Note>> = noteDao.getAll()

    fun insertOrUpdate(note: Note): Int {
        return noteDao.insert(note).toInt()
    }

    fun remove(note: Note) {
        noteDao.delete(note)
```

```
    }

    fun findById(id: Int): Note? {
        return noteDao.findById(id)
    }

    fun populate() {
        for (item in NOTE_SQL) {
            val query = SimpleSQLiteQuery(item, null)
            noteDao.populate(query)
        }
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/data/TodoRepository.kt

```
package com.example.twonotes.data

import androidx.lifecycle.LiveData
import androidx.sqlite.db.SimpleSQLiteQuery
import com.example.twonotes.dao.TodoDao
import com.example.twonotes.model.Todo


class TodoRepository(private val todoDao: TodoDao) {
    val todos: LiveData<List<Todo>> = todoDao.getAll()

    fun insertOrUpdate(todo: Todo): Int {
       return todoDao.insert(todo).toInt()
    }

    fun remove(todo: Todo) {
        todoDao.delete( todo)
    }

    fun findById(id: Int): Todo? {
        return todoDao.findById(id)
    }
```

```
    fun populate() {
        for (item in TODO_SQL) {
            val query = SimpleSQLiteQuery(item, null)
            todoDao.populate(query)
        }
    }

}
```

Arquivo: app/src/main/java/com/example/twonotes/formatter/DateFormatter.k

```
package com.example.twonotes.formatter

import java.text.SimpleDateFormat
import java.util.*

open class DateFormatter {

    fun toNoteDate(date: Date): String {
        return if (isToday(date)) toTime(date) else toDate(date)
    }

    fun toDate(date: Date): String {
        return SimpleDateFormat(DATE_FORMAT, Locale.getDefault()).apply {
    }

    fun toTime(date: Date): String {
        val pattern = when (date.minutes) {
            0 -> HOUR_FORMAT
            in 1..9 -> HOUR_MINUTE_0_FORMAT
            else -> HOUR_MINUTE_FORMAT
        }
        return SimpleDateFormat(pattern, Locale.getDefault()).apply { isL
    }

    fun toTimeAndDate(date: Date?): String {
        if (date == null) return ""
        val pattern = "$DATE_FORMAT " + when (date.minutes) {
            0 -> HOUR_FORMAT
```

```
            in 1..9 -> HOUR_MINUTE_0_FORMAT
            else -> HOUR_MINUTE_FORMAT
        }
        return SimpleDateFormat(pattern, Locale.getDefault()).apply { isLenie
    }

    fun toTimeStamp(date: Date): String {
        return SimpleDateFormat(TIME_STAMP, Locale.getDefault()).apply { isLe
    }

    private fun isToday(date: Date): Boolean {
        val c = Calendar.getInstance().apply { timeInMillis = date.time }
        val now = Calendar.getInstance()
        return c[Calendar.DATE] == now[Calendar.DATE]
    }

    companion object {
        const val DATE_FORMAT = "dd/MM"
        const val HOUR_FORMAT = "H'h'"
        const val HOUR_MINUTE_FORMAT = "H'h'm"
        const val HOUR_MINUTE_0_FORMAT = "H'h'0m"
        const val TIME_STAMP = "y'-'M'-'d'-'Hms"
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/model/Converters.kt

```
package com.example.twonotes.model

import androidx.room.TypeConverter
import java.util.*

class Converters {
    @TypeConverter
    fun fromTimestamp(value: Long?): Date? {
        return value?.let { Date(it) }
    }

    @TypeConverter
```

```
    fun dateToTimestamp(date: Date?): Long? {
        return date?.time
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/model/Note.kt

```
package com.example.twonotes.model

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import java.util.*

@Entity
class Note(): Observable() {

    @PrimaryKey(autoGenerate = true)
    var id: Int? = null

    @ColumnInfo
    var text: String? = null

    @ColumnInfo
    var imageUri: String? = null

    @ColumnInfo
    var editDate: Date = Date()

    constructor(text: String?, imageUri: String?) : this() {
        this.text = text
        this.imageUri = imageUri
        this.editDate = Date()
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/model/Todo.kt

```
package com.example.twonotes.model
```

```
import androidx.databinding.BaseObservable
import androidx.databinding.Bindable
import androidx.room.*
import com.airbnb.epoxy.databinding.BR
import java.util.*

@Entity
class Todo(): BaseObservable() {

    @PrimaryKey(autoGenerate = true)
    var id: Int? = null

    @ColumnInfo
    @Bindable
    var description: String? = null
        set(value) {
            field = value
            notifyPropertyChanged(BR.description)
        }

    @ColumnInfo
    @Bindable
    var reminderDate: Date? = null
        set(value) {
            field = value
            notifyPropertyChanged(BR.reminderDate)
        }

    @ColumnInfo
    var done: Boolean = false

    constructor(description: String, reminderDate: Date) : this() {
        this.reminderDate = reminderDate
        this.description = description
    }

    constructor(description: String) : this() {
        this.description = description
```

```
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/notifications/AlarmReceiv

```
package com.example.twonotes.notifications

import android.app.PendingIntent
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.os.AsyncTask
import android.util.Log
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import com.example.twonotes.App
import com.example.twonotes.App.Companion.CHANNEL_ID
import com.example.twonotes.R
import com.example.twonotes.dao.AppDatabase
import com.example.twonotes.data.TodoRepository
import com.example.twonotes.ui.todos.TodosFragment
import kotlinx.android.synthetic.main.edit_todo_dialog.view.*

class AlarmReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        val pendingResult: PendingResult = goAsync()
        val asyncTask = Task(context, pendingResult, intent)
        asyncTask.execute()
    }

    private class Task(private val context: Context, private val pendingI

        override fun doInBackground(vararg params: String?): Boolean {
            val todoRepository = TodoRepository(AppDatabase.getInstance(c

            val todoId = intent.getIntExtra(AlarmUtil.ALARM_TODO_ID, -1)

            Log.d(TAG, "Received broadcast with todoId = $todoId")
```

```
            val todoItem = todoRepository.findById(todoId) ?: return false

            val pendingIntent = Intent(context, TodosFragment::class.java).le
                PendingIntent.getBroadcast(App.appContext, 0, intent, 0)
            }

            val builder = NotificationCompat.Builder(context, CHANNEL_ID)
                .setContentTitle(todoItem.description)
                .setSmallIcon(R.drawable.ic_alarm_24)
                .setPriority(NotificationCompat.PRIORITY_HIGH)
                .setContentIntent(pendingIntent)
                .setAutoCancel(true)

            with(NotificationManagerCompat.from(context)) {
                notify(todoItem.id!!, builder.build())
            }

            return true
        }

        override fun onPostExecute(result: Boolean?) {
            super.onPostExecute(result)
            pendingResult.finish()
        }
    }

    companion object {
        const val TAG = "ALARM_RECEIVER"
    }

}
```

Arquivo: app/src/main/java/com/example/twonotes/notifications/AlarmUtil.kt

```
package com.example.twonotes.notifications

import android.app.AlarmManager
import android.app.PendingIntent
```

```
import android.content.Context
import android.content.Intent
import com.example.twonotes.App
import com.example.twonotes.model.Todo

class AlarmUtil {

    companion object {

        fun createAlarm(item: Todo) {
            val timeInMillis = item.reminderDate?.time ?: return

            val alarmManager = App.appContext.getSystemService(Context.AL
            val pendingIntent = Intent(App.appContext, AlarmReceiver::cla
                intent.putExtra(ALARM_TODO_ID, item.id!!)
                PendingIntent.getBroadcast(App.appContext, item.id!!, int
            }

            if (pendingIntent != null && alarmManager != null) {
                alarmManager.cancel(pendingIntent)
            }

            alarmManager?.setExact(
                AlarmManager.RTC_WAKEUP,
                timeInMillis,
                pendingIntent
            )
        }

        fun cancelAlarm(item: Todo) {
            val alarmManager = App.appContext.getSystemService(Context.AL
            val pendingIntent = Intent(App.appContext, AlarmReceiver::cla
                PendingIntent.getBroadcast(App.appContext, item.id!!, int
            }
            if (pendingIntent != null && alarmManager != null) {
                pendingIntent.cancel()
                alarmManager.cancel(pendingIntent)
            }
```

```
        }

        const val ALARM_TODO_ID = "com.example.twonotes.ALARM_TODO_ID"

    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/SplashActivity.kt

```
package com.example.twonotes.ui

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import com.example.twonotes.MainActivity


class SplashActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val intent = Intent(
            this@SplashActivity, MainActivity::class.java
        )
        startActivity(intent)
        finish()
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/todos/EditTodoDialog.kt

```
package com.example.twonotes.ui.todos

import android.app.*
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.view.LayoutInflater
import android.view.View
import android.widget.DatePicker
```

```
import android.widget.TimePicker
import androidx.databinding.DataBindingUtil
import com.example.twonotes.R
import com.example.twonotes.databinding.EditTodoDialogBinding
import com.example.twonotes.formatter.DateFormatter
import com.example.twonotes.model.Todo
import java.util.*

class EditTodoDialog(activity: Activity, val todo: Todo? = null, val onC
    View.OnClickListener, DatePickerDialog.OnDateSetListener, TimePicker
{

    private lateinit var mBinding: EditTodoDialogBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        this.setCancelable(false)

        mBinding = DataBindingUtil.inflate(LayoutInflater.from(context),

        mBinding.dateFormatter = DateFormatter()

        mBinding.dialogTitle.text = if (todo != null) context.getText(R.s
else context.getText(R.string.title_add_todo)

        mBinding.cancelButton.setOnClickListener(this)
        mBinding.confirmButton.setOnClickListener(this)
        mBinding.dateButton.setOnClickListener(this)
        mBinding.timeButton.setOnClickListener(this)
        mBinding.clearButton.setOnClickListener(this)

        mBinding.descriptionEt.addTextChangedListener(object : TextWatche
            override fun beforeTextChanged(s: CharSequence, start: Int, c

            override fun onTextChanged(s: CharSequence, start: Int, befor

            override fun afterTextChanged(s: Editable) {
                mBinding.todo?.description = s.toString()
            }
```

```
        })

        mBinding.todo = todo ?: Todo()

        setContentView(mBinding.root)
    }

    override fun onClick(v: View?) {
        when (v?.id) {
            R.id.confirm_button -> onConfirmClick()
            R.id.cancel_button -> onCancelButton()
            R.id.date_button -> onDateClick()
            R.id.time_button -> onTimeClicked()
            R.id.clear_button -> onClearDateClicked()
        }
    }

    private fun onConfirmClick() {
        this.onConfirmClicked(mBinding.todo as Todo)
        dismiss()
    }

    private fun onCancelButton() {
        dismiss()
    }

    private fun onClearDateClicked() {
        mBinding.todo?.reminderDate = null
    }

    private fun onDateClick() {
        val c = Calendar.getInstance()

        val todoDate = todo?.reminderDate
        if (todoDate != null) c.timeInMillis = todoDate.time

        val year = c.get(Calendar.YEAR)
        val month = c.get(Calendar.MONTH)
        val day = c.get(Calendar.DAY_OF_MONTH)
```

```
            DatePickerDialog(context, this, year, month, day).show()
        }

        private fun onTimeClicked() {
            val c = Calendar.getInstance()

            val todoDate = todo?.reminderDate
            if (todoDate != null) c.timeInMillis = todoDate.time

            val hour = c.get(Calendar.HOUR_OF_DAY)
            val minute = c.get(Calendar.MINUTE)

            TimePickerDialog(context, this, hour, minute, true).show()
        }

        override fun onDateSet(view: DatePicker?, year: Int, month: Int, dayC
            val c = Calendar.getInstance().apply {
                set(Calendar.YEAR, year)
                set(Calendar.MONTH, month)
                set(Calendar.DAY_OF_MONTH, dayOfMonth)
            }
            mBinding.todo?.reminderDate = Date(c.timeInMillis)
        }

        override fun onTimeSet(view: TimePicker?, hourOfDay: Int, minute: Int
            val c = Calendar.getInstance().apply {
                set(Calendar.HOUR_OF_DAY, hourOfDay)
                set(Calendar.MINUTE, minute)
            }
            mBinding.todo?.reminderDate = Date(c.timeInMillis)
        }
    }
```

Arquivo: app/src/main/java/com/example/twonotes/ui/todos/TodosFragment.kt

```
package com.example.twonotes.ui.todos

import android.app.Activity
```

```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProvider
import com.example.twonotes.MainActivity
import com.example.twonotes.R
import com.example.twonotes.databinding.FragmentTodosBinding
import com.example.twonotes.formatter.DateFormatter
import com.example.twonotes.model.Todo
import com.example.twonotes.todoItem

class TodosFragment : Fragment(), View.OnClickListener {

    private val viewModel: TodosViewModel by viewModels()
    private val dateFormatter = DateFormatter()
    private lateinit var mBinding: FragmentTodosBinding

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?
        mBinding = FragmentTodosBinding.inflate(inflater, container, false)

        lifecycle.addObserver(viewModel)

        viewModel.todos.observe(viewLifecycleOwner, Observer {
            rebuild(it)
        })

        mBinding.btnAdd.setOnClickListener(this)
        return mBinding.root
    }

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        (activity as MainActivity).setActionBarTitle(R.string.title_todos)
    }
```

```
private fun rebuild(todos: List<Todo>) {
    if (todos.isEmpty()) {
        mBinding.emptyLayout.visibility = View.VISIBLE
        mBinding.rvTodos.visibility = View.GONE
    } else {
        mBinding.emptyLayout.visibility = View.GONE
        mBinding.rvTodos.visibility = View.VISIBLE
    }
    mBinding.rvTodos.withModels {
        for (item in todos) {
            todoItem {
                id("${item.id}_${item.description}_${item.reminderDat
                todo(item)
                reminderText(dateFormatter.toTimeAndDate(item.reminde
                onCheckBoxClick {
                    viewModel.toggleTodoCheck(item)
                }
                onItemClick {
                    openEditDialog(item)
                }
                onRemoveClick {
                    viewModel.removeItem(item)
                }
            }
        }
    }
}

override fun onClick(v: View?) {
    if (v?.id == R.id.btn_add) {
        openEditDialog()
    }
}

private fun openEditDialog(item: Todo? = null) {
    EditTodoDialog(
        activity as Activity,
        item,
```

```
            onConfirmClicked = ::onEditFinished
        ).show()
    }

    private fun onEditFinished(item: Todo) {
        viewModel.onEditFinished(item)
    }

}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/todos/TodosViewModel.kt

```
package com.example.twonotes.ui.todos

import androidx.lifecycle.*
import com.example.twonotes.App
import com.example.twonotes.dao.AppDatabase
import com.example.twonotes.data.TodoRepository
import com.example.twonotes.model.Todo
import com.example.twonotes.notifications.AlarmUtil
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class TodosViewModel : ViewModel(), LifecycleObserver {

    private val repository: TodoRepository = TodoRepository(AppDatabase.getIn
    val todos: LiveData<List<Todo>> = repository.todos

    fun populateDB() = viewModelScope.launch(Dispatchers.IO) {
        repository.populate()
    }

    fun onEditFinished(item: Todo) = viewModelScope.launch(Dispatchers.IO) {
        item.id = repository.insertOrUpdate(item)
        if (item.reminderDate != null) {
            AlarmUtil.createAlarm(item)
        } else {
            AlarmUtil.cancelAlarm(item)
        }
```

```
    }

    fun toggleTodoCheck(item: Todo) = viewModelScope.launch(Dispatchers.I
        item.done = !item.done
        repository.insertOrUpdate(item)
    }

    fun removeItem(item: Todo) = viewModelScope.launch(Dispatchers.IO) {
        repository.remove(item)
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/notes/NotesFragment.kt

```
package com.example.twonotes.ui.notes

import android.content.Intent
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Observer
import com.example.twonotes.MainActivity
import com.example.twonotes.R
import com.example.twonotes.databinding.FragmentNotesBinding
import com.example.twonotes.formatter.DateFormatter
import com.example.twonotes.model.Note
import com.example.twonotes.noteItem
import com.example.twonotes.ui.editNote.EditNoteActivity
import com.example.twonotes.ui.editNote.EditNoteActivity.Companion.NOTE_I

class NotesFragment : Fragment(), View.OnClickListener {

    private val viewModel: NotesViewModel by viewModels()
    private lateinit var mBinding: FragmentNotesBinding

    override fun onCreateView(inflater: LayoutInflater, container: ViewGr
```

```
        mBinding = FragmentNotesBinding.inflate(inflater, container, false)

        lifecycle.addObserver(viewModel)

        viewModel.notes.observe(viewLifecycleOwner, Observer {
            rebuild(it)
        })

        mBinding.btnAdd.setOnClickListener(this)

        return mBinding.root
    }

    override fun onActivityCreated(savedInstanceState: Bundle?) {
        super.onActivityCreated(savedInstanceState)
        (activity as MainActivity).setActionBarTitle(R.string.title_notes)
    }

    private fun rebuild(notes: List<Note>) {
        if (notes.isEmpty()) {
            mBinding.emptyLayout.visibility = View.VISIBLE
            mBinding.rvNotes.visibility = View.GONE
        } else {
            mBinding.emptyLayout.visibility = View.GONE
            mBinding.rvNotes.visibility = View.VISIBLE
        }
        mBinding.rvNotes.withModels {
            for (item in notes) {
                noteItem {
                    id("${item.id}_${item.text}_${item.editDate.hashCode()}")
                    note(item)
                    editDate(DateFormatter().toNoteDate(item.editDate))
                    onItemClick {
                        onEditClicked(item)
                    }
                }
            }
        }
    }
```

```
    override fun onClick(v: View?) {
        if (v?.id == R.id.btn_add) {
            onEditClicked()
        }
    }

    private fun onEditClicked(item: Note? = null) {
        startActivity(
            Intent(context, EditNoteActivity::class.java).apply {
                if (item != null) putExtra(NOTE_ID, item.id)
            }
        )
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/notes/NotesViewModel.k

```
package com.example.twonotes.ui.notes

import androidx.lifecycle.LifecycleObserver
import androidx.lifecycle.LiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.twonotes.App
import com.example.twonotes.dao.AppDatabase
import com.example.twonotes.data.NoteRepository
import com.example.twonotes.model.Note
import com.example.twonotes.model.Todo
import com.example.twonotes.notifications.AlarmUtil
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class NotesViewModel : ViewModel(), LifecycleObserver {

    private val repository: NoteRepository = NoteRepository(AppDatabase.g
    val notes: LiveData<List<Note>> = repository.notes
```

```
    fun populateDB() = viewModelScope.launch(Dispatchers.IO) {
        repository.populate()
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/editNote/EditNoteActivity.

```
package com.example.twonotes.ui.editNote

import android.R.id
import android.app.Activity
import android.app.AlertDialog
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.os.Environment
import android.provider.MediaStore
import android.util.Log
import android.view.Menu
import android.view.MenuInflater
import android.view.MenuItem
import android.view.View
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.FileProvider
import androidx.core.net.toUri
import androidx.lifecycle.Observer
import com.example.twonotes.R
import com.example.twonotes.databinding.ActivityEditNoteBinding
import com.example.twonotes.model.Note
import java.io.File
import java.io.IOException
import java.text.SimpleDateFormat
import java.util.*

class EditNoteActivity : AppCompatActivity(), View.OnClickListener {

    private val viewModel: EditNoteViewModel by viewModels()
    private lateinit var mBinding: ActivityEditNoteBinding
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    mBinding = ActivityEditNoteBinding.inflate(layoutInflater, null,

    viewModel.note.observe(this, Observer {
        rebuild(it)
    })

    viewModel.setId(intent.getIntExtra(NOTE_ID, -1))

    mBinding.btnPhoto.setOnClickListener(this)
    mBinding.btnRemovePhoto.setOnClickListener(this)

    setContentView(mBinding.root)
}

override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater: MenuInflater = menuInflater
    inflater.inflate(R.menu.activity_edit_note, menu)
    return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        id.home -> {
            onBackPressed()
            true
        }
        R.id.delete_note -> {
            removeNote()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}

private fun rebuild(item: Note?) {
    mBinding.contents = item?.text
```

```
        val imageUriString = item?.imageUri
        if (imageUriString != null) {
            updateImageLayout(imageUriString.toUri())
            viewModel.setImageUri(imageUriString.toUri())
        }
    }

    private fun updateImageLayout(uri: Uri?) {
        if (uri == null) {
            mBinding.noteImage.setImageURI(uri)
            mBinding.noteImage.visibility = View.GONE
            mBinding.btnPhoto.visibility = View.VISIBLE
            mBinding.btnRemovePhoto.visibility = View.GONE
        } else {
            mBinding.noteImage.setImageURI(uri)
            mBinding.btnPhoto.visibility = View.GONE
            mBinding.btnRemovePhoto.visibility = View.VISIBLE
        }
    }

    override fun onBackPressed() {
        viewModel.addOrUpdate(mBinding.noteTv.text.toString())
        super.onBackPressed()
    }

    override fun onClick(v: View?) {
        when (v?.id) {
            R.id.btn_photo -> onAddPhotoClicked()
            R.id.btn_remove_photo -> removePhoto()
        }
    }

    private fun onAddPhotoClicked() {
        val options = arrayOf(
            getString(R.string.dialog_take_photo),
            getString(R.string.dialog_choose_gallery),
            getString(R.string.button_cancel)
        )
```

```
        AlertDialog.Builder(this).also {
            it.setTitle(R.string.dialog_photo_title)
            it.setCancelable(true)
            it.setItems(options) { dialog, which ->
                if (options[which] == getString(R.string.dialog_take_phot
                    dispatchTakePhotoIntent()
                } else if (options[which] == getString(R.string.dialog_ch
                    dispatchPickImageIntent()
                }
                dialog.cancel()
            }
        }.show()
    }


    private fun dispatchPickImageIntent() {
        val intent = Intent(Intent.ACTION_OPEN_DOCUMENT)
        intent.type = "image/*"
        startActivityForResult(
            Intent.createChooser(intent, "Select Picture"),
            PICK_IMAGE_REQUEST_CODE
        )
    }

    private fun dispatchTakePhotoIntent() {
        Intent(MediaStore.ACTION_IMAGE_CAPTURE).also { takePictureIntent
            takePictureIntent.resolveActivity(packageManager)?.also {
                val photoFile: File? = try {
                    createImageFile()
                } catch (ex: IOException) {
                    Log.v("EDIT_NOTE_ACTIVITY", "Error ocurred while crea
                    null
                }
                photoFile?.also {
                    val photoURI: Uri = FileProvider.getUriForFile(
                        this,
                        "com.example.android.fileprovider",
                        it
                    )
```

```
                takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photo
                startActivityForResult(takePictureIntent, TAKE_PHOTO_REQU
            }
        }
    }
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: In
    super.onActivityResult(requestCode, resultCode, data)
    if (resultCode != Activity.RESULT_OK) return

    if (requestCode == PICK_IMAGE_REQUEST_CODE && data != null) {
        onPickImageResult(data)
    } else if (requestCode == TAKE_PHOTO_REQUEST_CODE) {
        onTakePhotoResult()
    }
}

private fun onTakePhotoResult() {
    updateImageLayout(viewModel.getImageUri())
}

private fun onPickImageResult(data: Intent) {
    val uri = data.data ?: return
    contentResolver.takePersistableUriPermission(uri, Intent.FLAG_GRANT_
    updateImageLayout(uri)
    viewModel.setImageUri(uri)
}

@Throws(IOException::class)
private fun createImageFile(): File {
    val timeStamp: String = SimpleDateFormat("yyyyMMdd_HHmmss").format(Da
    val storageDir: File? = getExternalFilesDir(Environment.DIRECTORY_PIC
    return File.createTempFile(
        "JPEG_${timeStamp}_",
        ".jpg",
        storageDir
    ).apply {
        viewModel.setImageUri(absolutePath.toUri())
```

```
        }
    }

    private fun removeNote() {
        viewModel.remove()
        super.onBackPressed()
    }

    private fun removePhoto() {
        viewModel.removePhoto()
        updateImageLayout(null)
    }

    companion object {
        const val NOTE_ID = "com.example.twonotes.NOTE_ID"
        const val PICK_IMAGE_REQUEST_CODE = 13
        const val TAKE_PHOTO_REQUEST_CODE = 14
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/editNote/EditNoteViewM

```
package com.example.twonotes.ui.editNote

import android.net.Uri
import androidx.core.net.toUri
import androidx.lifecycle.*
import com.example.twonotes.App
import com.example.twonotes.dao.AppDatabase
import com.example.twonotes.data.NoteRepository
import com.example.twonotes.formatter.DateFormatter
import com.example.twonotes.model.Note
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import java.io.File
import java.io.IOException
import java.io.InputStream
import java.io.OutputStream
import java.util.*
```

```
class EditNoteViewModel : ViewModel() {

    private val repository: NoteRepository = NoteRepository(AppDatabase.getIn
    private var imageUri: Uri? = null

    private val noteId: MutableLiveData<Int> = MutableLiveData()
    val note = noteId.switchMap { id ->
        liveData(context = viewModelScope.coroutineContext + Dispatchers.IO)
            emit(repository.findById(id))
        }
    }

    fun setId(id: Int) {
        noteId.value = id
    }

    fun setImageUri(uri: Uri) {
        imageUri = uri
    }

    fun getImageUri(): Uri? {
        return imageUri
    }

    fun addOrUpdate(text: String) {
        val oldNote = note.value
        if (oldNote != null) {
            updateNote(oldNote, text)
        } else if (text.isNotBlank() || imageUri != null) {
            addNote(text)
        }
    }

    private fun addNote(text: String) = viewModelScope.launch(Dispatchers.IO)
        val localUri = saveImage(imageUri)
        val newNote = Note(if (text.isEmpty()) null else text, localUri)
        repository.insertOrUpdate(newNote)
    }
```

```
    private fun updateNote(note: Note, text: String) = viewModelScope.lau
        var oldUri: Uri? = null
        if (note.imageUri != null) oldUri = note.imageUri!!.toUri()
        if (text.isBlank() && imageUri == null) {
            repository.remove(note)
        } else {
            if (imageUri != null && imageUri != oldUri) {
                note.imageUri = saveImage(imageUri)
            } else if (imageUri == null && oldUri != null) {
                if (deleteImage(oldUri)) {
                    note.imageUri = null
                }
            }

            note.also {
                it.text = if (text.isBlank()) null else text
                it.editDate = Date()
            }
            repository.insertOrUpdate(note)
        }
    }

    fun remove() = viewModelScope.launch(Dispatchers.IO) {
        if (note.value != null) repository.remove(note.value!!)
    }

    fun removePhoto() = viewModelScope.launch(Dispatchers.IO) {
        imageUri = null
    }

    private fun saveImage(uri: Uri?): String? {
        if (uri == null) return null
        val localFile = File(App.appContext.filesDir, "image${DateFormatt
        val inputStream = App.appContext.contentResolver.openInputStream(
        val outputStream = localFile.outputStream()
        copyStream(inputStream, outputStream)
        return localFile.toUri().toString()
    }
```

```
    private fun deleteImage(uri: Uri): Boolean {
        val localFile = File(uri.path!!)
        return localFile.delete()
    }

    @Throws(IOException::class)
    fun copyStream(input: InputStream, output: OutputStream) {
        val buffer = ByteArray(1024)
        var bytesRead: Int
        while (input.read(buffer).also { bytesRead = it } != -1) {
            output.write(buffer, 0, bytesRead)
        }
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/ui/**package**-info.java

```
@EpoxyDataBindingLayouts({
        R.layout.note_item,
        R.layout.todo_item,
})

package com.example.twonotes.ui;

import com.airbnb.epoxy.EpoxyDataBindingLayouts;
import com.example.twonotes.R;
```

Arquivo: ndroid/app/src/main/java/com/example/twonotes/App.kt

```
package com.example.twonotes

import android.app.Application
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
import android.os.Build

class App : Application() {
```

```
    override fun onCreate() {
        super.onCreate()
        context = applicationContext
        createNotificationChannel()
    }

    private fun createNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val name = getString(R.string.channel_name)
            val descriptionText = getString(R.string.channel_description)
            val importance = NotificationManager.IMPORTANCE_DEFAULT
            val channel = NotificationChannel(CHANNEL_ID, name, importanc
                description = descriptionText
            }
            val notificationManager: NotificationManager = getSystemServi
            notificationManager.createNotificationChannel(channel)
        }
    }

    companion object {
        private lateinit var context: Context
        val appContext: Context
            get() = context

        const val CHANNEL_ID = "com.example.two.notes.Reminders"
    }
}
```

Arquivo: app/src/main/java/com/example/twonotes/MainActivity.kt

```
package com.example.twonotes

import android.os.Bundle
import com.google.android.material.bottomnavigation.BottomNavigationView
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.findNavController
import androidx.navigation.ui.AppBarConfiguration
import androidx.navigation.ui.setupActionBarWithNavController
import androidx.navigation.ui.setupWithNavController
```

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val navView: BottomNavigationView = findViewById(R.id.nav_view)

        val navController = findNavController(R.id.nav_host_fragment)

        val appBarConfiguration = AppBarConfiguration(setOf(R.id.navigation_t

        setupActionBarWithNavController(navController, appBarConfiguration)
        navView.setupWithNavController(navController)
    }

    fun setActionBarTitle(title: Int) {
        setTitle(title)
    }
}
```

B.2 iOS

Arquivo: TwoNotes/NoteList/NoteView.swift

```
import Foundation
import SwiftUI

struct EditNoteView: View {
    @Environment(\.managedObjectContext) var managedObjectContext
    @Environment(\.presentationMode) var presentationMode

    @ObservedObject private var keyboard = KeyboardResponder()

    @State private var text: String
    @State private var deleted = false
    @State private var image: UIImage? = nil
    @State private var showingActionSheet = false
```

```
@State private var showingImagePicker = false
@State private var sourceType = UIImagePickerController.SourceType.ph

var note: Note?
private var business = NoteBusiness()

init(note: Note?) {
    self.note = note
    self._text = State(initialValue: note?.text ?? "")
    if (note?.image != nil ) { self._image = State(initialValue: UIIm
}

private func onRemove() {
    self.deleted = true
    if self.note != nil {
        business.deleteNote(managedObjectContext, note: self.note!)
    }
    self.presentationMode.wrappedValue.dismiss()
}

private func onRemovePhoto() {
    self.image = nil
}

private func onClose() {
    if self.deleted { return }
    if self.note == nil {
        business.addNote(managedObjectContext, text: self.text, image
    } else if self.note != nil {
        business.updateNote(managedObjectContext, note: self.note!, t
    }
}

var body: some View {
    ZStack {
        VStack(alignment: .center) {
            if image != nil {
                ZStack(alignment: .topTrailing) {
                    Image(uiImage: self.image!)
```

```
                        .resizable()
                        .aspectRatio(image!.size, contentMode: .fit)
                        .padding()
                    Button(action: {
                        self.onRemovePhoto()
                    }) {
                        Image("Close").frame(width: 30, height: 30)
                        .foregroundColor(Color("Black23"))
                    }
                    .background(Color(.white))
                        .cornerRadius(38.5)
                        .padding()
                        .shadow(color: Color.black.opacity(0.3), radius:
                    .offset(x: -5, y: +5)
                }
            }
            TextView(text: $text, placeholder: "Add text").padding()
            Spacer()
        } // End VStack
        if image == nil {
            VStack {
                Spacer()
                HStack {
                 Spacer()
                 Button(action: {
                        self.showingActionSheet = true
                 }) {
                        Image("AddPhoto").frame(width: 55, height: 55)
                        .foregroundColor(Color.white)
                 }
                        .background(Color("Merengue"))
                        .cornerRadius(38.5)
                        .padding()
                        .shadow(color: Color.black.opacity(0.3), radius: 3, x
                }
            } // End Button VStack
            .padding()
            .padding(.bottom, keyboard.currentHeight)
            .edgesIgnoringSafeArea(.bottom)
```

```
                .animation(.easeOut(duration: 0.16))
            } // End if
            EmptyView()

            .navigationBarTitle(note == nil ? "Create note" : "Edit note"

            .navigationBarItems(trailing: Button(action: {
                self.onRemove()
            }) {
                Image("Delete")
            })

            .sheet(isPresented: $showingImagePicker) {
                ImagePicker(sourceType: self.sourceType, selectedImage: s
            }

            .actionSheet(isPresented: $showingActionSheet) {
                ActionSheet(title: Text("Add photo"), buttons: [
                    .default(Text("Take photo")) {
                        self.sourceType = .camera
                        self.showingImagePicker = true
                    },
                    .default(Text("Choose from gallery")) {
                        self.sourceType = .photoLibrary
                        self.showingImagePicker = true

                    },
                    .cancel()
                ])
            }

            .onDisappear {
                self.onClose()
            }

        }
    }
}
```

```
struct EditNoteViewPreviewContainer : View {
     @State private var note: Note? = nil

     var body: some View {
        NavigationView {
            EditNoteView(note: note)
        }
     }
}


struct EditNoteView_Previews: PreviewProvider {
    static var previews: some View {
        EditNoteViewPreviewContainer()
    }
}
```

Arquivo: TwoNotes/NoteList/NoteBusiness.swift

```
import Foundation
import CoreData
import SwiftUI

class NoteBusiness {

    func addNote(_ context: NSManagedObjectContext, text: String, image: UIIm
        if text.isEmpty && image == nil {
            return
        }
        let note = Note(context: context)
        note.id = UUID()
        note.text = text
        note.editDate = Date()
        note.image = image?.pngData()
        saveContext(context)
    }

    func updateNote(_ context: NSManagedObjectContext, note: Note, text: Stri
        if text.isEmpty && image == nil {
```

```
            deleteNote(context, note: note)
        }
        note.text = text
        note.editDate = Date()
        note.image = image?.pngData()
        saveContext(context)
    }

    func deleteNote(_ context: NSManagedObjectContext, note: Note) {
        context.delete(note)
        saveContext(context)
    }

    private func saveContext(_ context: NSManagedObjectContext) {
      do {
        try context.save()
      } catch {
        print("Error saving managed object context: \(error)")
      }
    }

    func populate(_ context: NSManagedObjectContext) {
        for _ in 1...50 {
            let note = Note(context: context)
            note.id = UUID()
            note.text = generateText(length: Int.random(in: 100...1000))
            note.editDate = Date(timeIntervalSince1970: Double.random(in:
        }
        saveContext(context)
    }

    func generateText(length: Int) -> String {
        var result = "";
        let characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrs
        let charactersLength = characters.count;
        for _ in 0...length {
            let index = characters.index(characters.startIndex, offsetBy:
            result = "\(result)\(characters[index])";
        }
```

```
        return result;
    }
}
```

Arquivo: TwoNotes/NoteList/NoteListView.swift

```
import Foundation
import SwiftUI

struct NoteListView: View {
    @Environment(\.managedObjectContext) var managedObjectContext

    @FetchRequest(
        entity: Note.entity(),
        sortDescriptors: [
            NSSortDescriptor(keyPath: \Note.editDate, ascending: false),
        ]
    ) var notes: FetchedResults<Note>

    @State private var action: Int? = 0

    private var business = NoteBusiness()

    var body: some View {
        ZStack {
            if notes.isEmpty {
                VStack {
                    VStack {
                        Image("EmptyNotes").resizable()
                        .aspectRatio(contentMode: .fit)
                    }.frame(width: 200.0, height: 200.0)
                    Text("No notes here yet!").font(.system(size: 20)).paddin
                }
            } else {
                List {
                    ForEach (notes, id: \.id) { note in
                        NavigationLink(destination: EditNoteView(note: note).
                            NoteCell(note: note).environment(\.managedObjectC
                        }
```

```
                    }
                } // End List
            } // Close else
            VStack {
                Spacer()
                HStack {
                    Spacer()
                    NavigationLink(destination: EditNoteView(note: nil).envir
                                   tag: 1, selection: $action) {
                        EmptyView()
                    }
                    Button(action: {
                        self.action = 1
                    }, label: {
                        Text("+")
                            .font(.system(size: 35))
                            .frame(width: 55, height: 50)
                            .foregroundColor(Color.white)
                            .padding(.bottom, 7)
                    })
                        .background(Color("Merengue"))
                        .cornerRadius(38.5)
                        .padding()
                        .shadow(color: Color.black.opacity(0.3), radius: 3, x
                }
            }// End VStack
        } // End ZStack
    }
}

struct NoteCell: View {
    @ObservedObject var note: Note
    let formatter = CustomFormatter()

    var body: some View {
        VStack(alignment: .leading) {
            Text(note.text ?? "Note without text").font(.system(size: 16,
            Text(note.editDate != nil ? formatter.toNoteDate(note.editDat
            Divider()
```

```
        }.padding(.top, 4).padding(.bottom, 4)
    }
}
```

Arquivo: TwoNotes/TodoList/EditTodoSheet.swift

```
import Foundation
import SwiftUI

struct EditTodoSheet: View {
    @Environment(\.managedObjectContext) var managedObjectContext
    @Binding private var todo: Todo?
    @Binding private var isPresented: Bool

    @State private var text: String
    @State private var selectedDate: Date
    @State private var showingDatePicker: Bool

    private var business: TodoBusiness = TodoBusiness()

    init(isPresented: Binding<Bool>, todo: Binding<Todo?>) {

        self._isPresented = isPresented
        self._todo = todo
        self._text = State(initialValue: todo.wrappedValue?.text ?? "")
        self._selectedDate = State(initialValue: todo.wrappedValue?.reminderD
        self._showingDatePicker = State(initialValue: todo.wrappedValue?.remi

        UIViewController().isModalInPresentation = true
    }

    private func onConfirm() {
        if self.todo == nil {
            business.addTodo(managedObjectContext, text: self.text, date: sho
        } else if self.todo != nil {
            business.updateTodo(managedObjectContext, todo: self.todo!, text:
        }
    }
```

```
var body: some View {
    NavigationView {
        VStack(alignment: .leading) {
            VStack {
                TextField("Add a description", text: $text)
                Divider()
            }.padding()
            Button(action: {
                withAnimation {
                    self.showingDatePicker.toggle()
                }
            }, label: {
                Image("Alarm").foregroundColor(Color("Sorvete"))
                Text(self.showingDatePicker ? "Remove reminder" : "Se
            }).padding(.leading).padding(.trailing) // End button

            if self.showingDatePicker {
                DatePicker("", selection: $selectedDate, in: Date()..
            }
            Spacer()
        } // End VStack (alignment)

        .navigationBarTitle(self.todo != nil ? "Edit to-do" : "Add ne

        .navigationBarItems(leading:
            Button(action: {
                self.isPresented = false
            }, label: {
                Text("Cancel").font(.system(size: 18)).foregroundColo
                }),
        trailing:
            Button(action: {
                    self.onConfirm()
                    self.isPresented = false
            }, label: {
                Text("Confirm").font(.system(size: 18, weight: .semib
        }).disabled(self.text.isEmpty)) // End NavigationBarItems
    }
```

```
    }
}

struct EditTodoSheetPreviewContainer : View {
     @State private var isPresented = false
     @State private var todo: Todo? = nil

     var body: some View {
        EditTodoSheet(isPresented: $isPresented, todo: $todo)
     }
}


struct EditTodoSheet_Previews: PreviewProvider {
    static var previews: some View {
        EditTodoSheetPreviewContainer()
    }
}
```

Arquivo: TwoNotes/TodoList/TodoBusiness.swift

```
import Foundation
import CoreData
import SwiftUI

class TodoBusiness {

    func addTodo(_ context: NSManagedObjectContext, text: String, date: Date?
        let todo = Todo(context: context)
        todo.id = UUID()
        todo.text = text
        todo.reminderDate = date
        if date != nil { addNotification(todo) }
        saveContext(context)
    }

    func updateTodo(_ context: NSManagedObjectContext, todo: Todo, text: Stri
        if date == nil && todo.reminderDate != nil { removeNotification(todo)
```

```
        if date != nil && date != todo.reminderDate { addNotification(tod
        todo.text = text
        todo.reminderDate = date
        saveContext(context)
    }

    func deleteTodo(_ context: NSManagedObjectContext, todo: Todo) {
        if todo.reminderDate != nil { removeNotification(todo) }
        context.delete(todo)
        saveContext(context)
    }

    func toggle(_ context: NSManagedObjectContext, todo: Todo) {
        todo.done.toggle()
        saveContext(context)
    }

    private func addNotification(_ todo: Todo) {
        let content = UNMutableNotificationContent()
        content.title = "\(todo.text ?? "Reminder")"
        content.sound = UNNotificationSound.default

        let dateComponents = Calendar.current.dateComponents([.year, .mon

        let trigger = UNCalendarNotificationTrigger(dateMatching: dateCon

        let request = UNNotificationRequest(identifier: todo.id!.uuidStri

        UNUserNotificationCenter.current().add(request) { (error) in
            if error != nil {
                print(error.debugDescription);
            }
        }
    }

    private func removeNotification(_ todo: Todo) {
        UNUserNotificationCenter.current().removePendingNotificationReque
    }
```

```
    private func saveContext(_ context: NSManagedObjectContext) {
      do {
        try context.save()
      } catch {
        print("Error saving managed object context: \(error)")
      }
    }

    func populate(_ context: NSManagedObjectContext) {
        for i in 1...100 {
            let todo = Todo(context: context)
            todo.id = UUID()
            todo.text = generateText(length: Int.random(in: 10...100))
            todo.done = Bool.random()
            todo.reminderDate = i % 2 == 0 ? nil : Date(timeIntervalSince1970
            print(todo)
            saveContext(context)
        }

    }

    func generateText(length: Int) -> String {
        var result = "";
        let characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw
        let charactersLength = characters.count;
        for _ in 0...length {
            let index = characters.index(characters.startIndex, offsetBy: Int
            result = "\(result)\(characters[index])";
        }
        return result;
    }
}
```

Arquivo: TwoNotes/TodoList/TodoListView.swift

```
import Foundation
import SwiftUI

struct TodoListView: View {
```

```
@Environment(\.managedObjectContext) var managedObjectContext

@FetchRequest(
    entity: Todo.entity(),
    sortDescriptors: [
        NSSortDescriptor(keyPath: \Todo.done, ascending: true),
        NSSortDescriptor(keyPath: \Todo.reminderDate, ascending: true
    ]
) var todos: FetchedResults<Todo>

@State private var showingDialog = false
@State private var showModally = true
@State private var editingTodo: Todo? = nil

private var business: TodoBusiness = TodoBusiness()

init() {
    UITableView.appearance().separatorStyle = .none
}

var body: some View {
    ZStack {
        if (todos.isEmpty) {
            VStack {
                VStack {
                    Image("EmptyTodos").resizable()
                    .aspectRatio(contentMode: .fit)
                }.frame(width: 200.0, height: 250.0)
                Text("List is empty").font(.system(size: 20)).padding
            }
        } else {
            List {
                ForEach (todos, id: \.id) { todo in
                    TodoCell(todo: todo) { todo in
                        self.showingDialog = true
                        self.editingTodo = todo
                    }.environment(\.managedObjectContext, self.manag
                }
            } // End List
```

```
            } // Close else
            VStack {
                Spacer()
                HStack {
                    Spacer()
                    Button(action: {
                        self.showingDialog = true
                        self.business.populate(self.managedObjectContext)
                    }, label: {
                        Text("+")
                            .font(.system(size: 35))
                            .frame(width: 55, height: 50)
                            .foregroundColor(Color.white)
                            .padding(.bottom, 7)
                    })
                        .background(Color("Merengue"))
                        .cornerRadius(38.5)
                        .padding()
                        .shadow(color: Color.black.opacity(0.3), radius: 3, x
                }
            } // End VStack

        }.sheet(isPresented: $showingDialog, onDismiss: {
            self.editingTodo = nil
        }) {
            EditTodoSheet(isPresented: self.$showingDialog,
                todo: self.$editingTodo).environment(\.managedObjectContext,
                self.managedObjectContext
                )
                .presentation(isModal: self.$showModally)
        }
    }
}

struct TodoCell: View {
    @Environment(\.managedObjectContext) var managedObjectContext
    @ObservedObject var todo: Todo
    var onClick: (Todo) -> Void
    let formatter = CustomFormatter()
```

```
let business = TodoBusiness()

func toggle() {
    business.toggle(managedObjectContext, todo: todo)
}

func delete() {
    business.deleteTodo(managedObjectContext, todo: todo)
}

var body: some View {
    HStack {
        Button(action: {
            self.toggle()
        }) {
          HStack {
            Image(systemName: todo.done ? "checkmark.square.fill" : "
            .resizable()
            .frame(width: 20.0, height: 20.0)
            .padding(.trailing)
                .foregroundColor(todo.done ? Color("Merengue") : Colo
          }
        }
        Button(action: {
            self.onClick(self.todo)
        }) {
            VStack(alignment: .leading) {
                Text(todo.text ?? "No description").font(.system(size
                if todo.reminderDate != nil {
                    Text(formatter.toDateAndTime(todo.reminderDate!))
                    .foregroundColor(Color("MerengueDark"))
                }
            }.accentColor(Color("Black80"))
            Spacer()
        }
        Button(action: {
            self.delete()
        }) {
            Image("Close")
```

```
            }.foregroundColor(Color("Black23"))
        }.padding(.top, 4).padding(.bottom, 4).buttonStyle(BorderlessButtonSt
    }
}

struct TodoListView_Previews: PreviewProvider {
    static var previews: some View {
        TodoListView()
    }
}
```

Arquivo: TwoNotes/Utils/CustomFormatter.swift

```
import Foundation

class CustomFormatter {
    private let formatter = DateFormatter()

    func toDateAndTime(_ date: Date) -> String {
        let calendar = Calendar.current
        let minutes = calendar.component(.minute, from: date)

        var pattern = "d'/'M␣"
        if (minutes == 0) {
          pattern += "H'h'";
        } else if (minutes < 10) {
          pattern += "H'h0'm"
        } else {
          pattern += "H'h'm"
        }
        formatter.dateFormat = pattern
        return formatter.string(from: date)
    }

    func toNoteDate(_ date: Date) -> String {
        if Calendar.current.isDateInToday(date) {
            return toTime(date)
        } else {
            return toDate(date)
```

```
        }
    }

    private func toTime(_ date: Date) -> String {
        let calendar = Calendar.current
        let minutes = calendar.component(.minute, from: date)

        var pattern = "";
        if (minutes == 0) {
          pattern += "H'h'"
        } else if (minutes < 10) {
          pattern += "H'h0'm"
        } else {
          pattern += "H'h'm"
        }
        formatter.dateFormat = pattern
        return formatter.string(from: date)
    }

    private func toDate(_ date: Date) -> String {
        formatter.dateFormat = "d'/'M"
        return formatter.string(from: date)
    }
}
```

Arquivo: TwoNotes/Utils/ImagePicker.swift

```
import Foundation
import UIKit
import SwiftUI

struct ImagePicker: UIViewControllerRepresentable {

    var sourceType: UIImagePickerController.SourceType = .photoLibrary

    @Binding var selectedImage: UIImage?
    @Environment(\.presentationMode) private var presentationMode

    func makeUIViewController(context: UIViewControllerRepresentableConte
```

```
        let imagePicker = UIImagePickerController()
        imagePicker.allowsEditing = false
        imagePicker.sourceType = sourceType
        imagePicker.delegate = context.coordinator

        return imagePicker
    }

    func updateUIViewController(_ uiViewController: UIImagePickerController,

    func makeCoordinator() -> Coordinator {
        Coordinator(self)
    }

    final class Coordinator: NSObject, UIImagePickerControllerDelegate, UINa

        var parent: ImagePicker

        init(_ parent: ImagePicker) {
            self.parent = parent
        }

        func imagePickerController(_ picker: UIImagePickerController, didFini

            if let image = info[UIImagePickerController.InfoKey.originalImage
                parent.selectedImage = image
            }

            parent.presentationMode.wrappedValue.dismiss()
        }
    }
}
```

Arquivo: TwoNotes/Utils/ModalSheet.swift

```
import Foundation
import SwiftUI
```

```
struct ModalView<T: View>: UIViewControllerRepresentable {
    let view: T
    @Binding var isModal: Bool
    let onDismissalAttempt: (()->())?

    func makeUIViewController(context: Context) -> UIHostingController<T>
        UIHostingController(rootView: view)
    }

    func updateUIViewController(_ uiViewController: UIHostingController<T
        uiViewController.parent?.presentationController?.delegate = conte
    }

    func makeCoordinator() -> Coordinator {
        Coordinator(self)
    }

    class Coordinator: NSObject, UIAdaptivePresentationControllerDelegate
        let modalView: ModalView

        init(_ modalView: ModalView) {
            self.modalView = modalView
        }

        func presentationControllerShouldDismiss(_ presentationController
            !modalView.isModal
        }

        func presentationControllerDidAttemptToDismiss(_ presentationCon
            modalView.onDismissalAttempt?()
        }
    }
}

extension View {
    func presentation(isModal: Binding<Bool>, onDismissalAttempt: (()->()
        ModalView(view: self, isModal: isModal, onDismissalAttempt: onDis
    }
}
```

Arquivo: TwoNotes/Utils/TextView.swift

```
import Foundation
import UIKit
import SwiftUI

struct TextView: UIViewRepresentable {
    @Binding var text: String
    var placeholder: String
    var placeholderLabel = UILabel()

    func makeCoordinator() -> Coordinator {
        Coordinator(self)
    }

    func makeUIView(context: Context) -> UITextView {

        let textView = UITextView()
        textView.delegate = context.coordinator

        textView.font = UIFont(name: "HelveticaNeue", size: 16)
        textView.isScrollEnabled = true
        textView.isEditable = true
        textView.isUserInteractionEnabled = true
        textView.backgroundColor = UIColor(white: 0.0, alpha: 0)

        placeholderLabel.text = placeholder
        placeholderLabel.font = UIFont.systemFont(ofSize: (textView.font?.poi
        placeholderLabel.sizeToFit()
        textView.addSubview(placeholderLabel)
        placeholderLabel.frame.origin = CGPoint(x: 5, y: (textView.font?.poin
        placeholderLabel.textColor = UIColor.lightGray
        placeholderLabel.isHidden = !text.isEmpty

        return textView
    }

    func updateUIView(_ uiView: UITextView, context: Context) {
```

```
        uiView.text = text
    }

    class Coordinator : NSObject, UITextViewDelegate {

        var parent: TextView

        init(_ uiTextView: TextView) {
            self.parent = uiTextView
        }

        func textView(_ textView: UITextView, shouldChangeTextIn range: N
            return true
        }

        func textViewDidChange(_ textView: UITextView) {
            self.parent.placeholderLabel.isHidden = !textView.text.isEmpt
            self.parent.text = textView.text
        }
    }
```

Arquivo: TwoNotes/AppDelegate.swift

```
import UIKit
import CoreData

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {



    func application(_ application: UIApplication, didFinishLaunchingWith
        return true
    }

    // MARK: UISceneSession Lifecycle
    func application(_ application: UIApplication, configurationForConnec
        return UISceneConfiguration(name: "Default Configuration", sessio
    }
```

```
    func application(_ application: UIApplication, didDiscardSceneSessions sc
    }

    // MARK: - Core Data stack
    lazy var persistentContainer: NSPersistentContainer = {
        let container = NSPersistentContainer(name: "TwoNotes")
        container.loadPersistentStores(completionHandler: { (storeDescription
                fatalError("Unresolved error \(error), \(error.userInfo)")
            }
        })
        return container
    }()

    // MARK: - Core Data Saving support
    func saveContext () {
        let context = persistentContainer.viewContext
        if context.hasChanges {
            do {
                try context.save()
            } catch {
                let nserror = error as NSError
                fatalError("Unresolved error \(nserror), \(nserror.userInfo)"
            }
        }
    }

}
```

Arquivo: TwoNotes/AppView.swift

```
import SwiftUI
import UserNotifications

struct AppView: View {
    @Environment(\.managedObjectContext) var managedObjectContext
    @State private var selection = 0

    init() {
```

```
        UITabBar.appearance().barTintColor = UIColor(named: "White")
        UITabBar.appearance().isTranslucent = false

        let coloredAppearance = UINavigationBarAppearance()
        coloredAppearance.configureWithOpaqueBackground()
        coloredAppearance.backgroundColor = UIColor(named: "Sorvete")
        coloredAppearance.titleTextAttributes = [.foregroundColor: UIColo
        coloredAppearance.largeTitleTextAttributes = [.foregroundColor: U

        UINavigationBar.appearance().standardAppearance = coloredAppearan
        UINavigationBar.appearance().scrollEdgeAppearance = coloredAppear

        UNUserNotificationCenter.current().requestAuthorization(options:
            if success {
                print("All set!")
            } else if let error = error {
                print(error.localizedDescription)
            }
        }
    }

    var body: some View {
        NavigationView {
            TabView(selection: $selection){
                TodoListView()
                    .tabItem {
                        VStack {
                            Image("LibraryAdd")
                                .renderingMode(.template)
                            Text("To-dos")
                        }
                    }
                    .tag(0)
                .environment(\.managedObjectContext, managedObjectContext
                NoteListView()
                    .tabItem {
                        VStack {
                            Image("Notes")
                            Text("Notes")
```

```
                        }
                    }
                    .tag(1)
                .environment(\.managedObjectContext, managedObjectContext)
            }.accentColor(Color("Sorvete"))
                .navigationBarTitle(
                    self.selection == 0 ? Text("To-dos") : Text("Notes"), dis
        }.accentColor( .white)
    }
}

struct AppView_Previews: PreviewProvider {
    static var previews: some View {
        AppView()
    }
}
```

Arquivo: TwoNotes/SceneDelegate.swift

```
import UIKit
import SwiftUI

class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?


    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, optio

        let context = (UIApplication.shared.delegate as! AppDelegate).persist
        let contentView = AppView().environment(\.managedObjectContext, conte

        if let windowScene = scene as? UIWindowScene {
            let window = UIWindow(windowScene: windowScene)
            window.rootViewController = UIHostingController(rootView: content
            self.window = window
            window.makeKeyAndVisible()
        }
    }
```

```
    func sceneDidDisconnect(_ scene: UIScene) {}

    func sceneDidBecomeActive(_ scene: UIScene) {}

    func sceneWillResignActive(_ scene: UIScene) {}

    func sceneWillEnterForeground(_ scene: UIScene) {}

    func sceneDidEnterBackground(_ scene: UIScene) {
        (UIApplication.shared.delegate as? AppDelegate)?.saveContext()
    }


}
```

B.3 Flutter

Arquivo: lib/business/notes_provider.dart

```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:path_provider/path_provider.dart';
import 'package:two_notes/data/note_repository.dart';
import 'package:two_notes/model/note.dart';
import 'package:two_notes/utils/date_formatter.dart';
import 'package:two_notes/utils/extensions.dart';

class Notes extends ChangeNotifier {
  final NoteRepository repository;

  List<Note> _notes;

  List<Note> get notes => _notes;

  bool _isLoading = false;
```

```
bool get loading => _isLoading;

Notes({
  @required this.repository,
  List<Note> notes,
}) {
  _notes = notes ?? [];
  _load();
}

void _load() async {
  _isLoading = true;
  notifyListeners();

  repository.getAll().then((value) {
    _notes = value;
    _isLoading = false;
    notifyListeners();
  }).catchError((err) {
    print(err);
    _isLoading = false;
    notifyListeners();
  });
}

void updateNote(Note note, String text, File newImage) async {
  assert(note != null);
  assert(note.id != null);

  if (text.isEmptyOrNull() && newImage == null) {
    return removeNote(note);
  }

  if (newImage != null && newImage.path != note.image?.path) {
    final File localImage = await _saveLocalImage(note.image);
    note.image = localImage;
  } else if (newImage == null && note.image != null) {
    _deleteLocalImage(note.image);
    note.image = null;
```

```
    }

    note.text = text.isEmptyOrNull() ? null : text;
    note.editDate = DateTime.now();

    notifyListeners();
    repository.update(note);
  }

  void addNote(String text, File image) async {
    File localImage;
    if (image != null) {
      localImage = await _saveLocalImage(image);
    }
    final newNote = await repository.insert(
        Note(text: text.isEmptyOrNull() ? null : text, image: localImage)
    _notes.add(newNote);
    notifyListeners();
  }

  void removeNote(Note note) {
    assert(note.id != null);
    _notes.removeWhere((it) => it.id == note.id);
    notifyListeners();
    repository.delete(note.id);
    if (note.image != null) _deleteLocalImage(note.image);
  }

  Future<File> _saveLocalImage(File image) async {
    final directory = await getApplicationDocumentsDirectory();
    final path = directory.path;

    final timeStamp = DateFormatter().toTimeStamp(DateTime.now());
    return image.copy('$path/image-$timeStamp');
  }

  void _deleteLocalImage(File image) {
    image.delete();
  }
```

```
}
```

Arquivo: lib/business/todos_provider.dart

```
import 'package:flutter/material.dart';
import 'package:two_notes/data/todo_repository.dart';
import 'package:two_notes/model/todo.dart';
import 'package:two_notes/utils/notifications.dart';

class Todos extends ChangeNotifier {
  final TodoRepository repository;

  List<Todo> _todos;

  List<Todo> get todos => _todos;

  bool _isLoading = false;

  bool get loading => _isLoading;

  Todos({@required this.repository, List<Todo> todos}) {
    _todos = todos ?? [];
    _load();
  }

  void _load() async {
    _isLoading = true;
    notifyListeners();

    repository.getAll().then((value) {
      _todos = value;
      _isLoading = false;
      notifyListeners();
    }).catchError((err) {
      print(err);
      _isLoading = false;
      notifyListeners();
    });
  }
```

```
void toggleTodo(Todo todo, bool value) {
  todo.done = value;
  updateTodo(todo);
}

void updateTodo(Todo todo) {
  assert(todo != null);
  assert(todo.id != null);
  var oldTodo = _todos.firstWhere((it) => it.id == todo.id);
  _updateNotifications(oldTodo, todo);
  repository.update(todo);
  _load();
}

void addTodo(Todo todo) async {
  final newTodo = await repository.insert(todo);
  notifyListeners();
  _updateNotifications(null, newTodo);
  _load();
}

void removeTodo(Todo todo) {
  assert(todo.id != null);
  _updateNotifications(todo, null);
  repository.delete(todo.id);
  _load();
}

void _updateNotifications(Todo oldTodo, Todo newTodo) {
  if (oldTodo != null &&
      oldTodo.reminderDate != null &&
      oldTodo.reminderDate.isAfter(DateTime.now())) {
    NotificationUtils.cancelNotification(oldTodo.id);
  }
  if (newTodo != null &&
      !newTodo.done &&
      newTodo.reminderDate != null &&
      newTodo.reminderDate.isAfter(DateTime.now())) {
```

```
      NotificationUtils.zonedScheduleNotification(newTodo);
    }
  }
}
```

Arquivo: lib/data/database_helper.dart

```
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import 'package:two_notes/data/populate_note.dart';
import 'package:two_notes/data/populate_todo.dart';

import 'package:two_notes/data/todo_repository.dart';
import 'package:two_notes/data/note_repository.dart';

class DatabaseHelper {
  static final _databaseName = "two_notes.db";
  static final _databaseVersion = 1;

  DatabaseHelper._privateConstructor();
  static final DatabaseHelper instance = DatabaseHelper._privateConstructor()

  static Database _database;
  Future<Database> get database async {
    if (_database != null) return _database;
    _database = await _initDatabase();
    return _database;
  }

  _initDatabase() async {
    final databasePath = await getDatabasesPath();
    String path = join(databasePath, _databaseName);
    return await openDatabase(path,
        version: _databaseVersion, onCreate: _onCreate);
  }

  Future _onCreate(Database db, int version) async {
    await TodoRepository.createTable(db);
    await NoteRepository.createTable(db);
```

```
    populateDB(db, TODO_SQL);
    populateDB(db, NOTE_SQL);
  }

  void populateDB(Database db, List<String> sql) async {
    final batch = db.batch();
    for (final line in sql) {
      batch.rawInsert(line);
    }
    await batch.commit(noResult: true);
  }

  Future close() async => _database.close();
}
```

Arquivo: lib/data/note_repository.dart

```
import 'package:sqflite/sqflite.dart';
import 'package:two_notes/model/note.dart';

import 'database_helper.dart';

final String tableNote = 'note';
final String columnNoteId = '_id';
final String columnText = 'text';
final String columnEditDate = 'editDate';
final String columnImage = 'image';

class NoteRepository {
  Future<Database> database = DatabaseHelper.instance.database;

  static Future createTable(Database db) {
    return db.execute('''
          create table $tableNote (
            $columnNoteId integer primary key autoincrement not null,
            $columnText text,
            $columnEditDate integer,
            $columnImage text);
          ''');
```

```
}

Future<Note> insert(Note note) async {
  final db = await database;
  note.id = await db.insert(tableNote, note.toMap());
  return note;
}

void insertAll(List<Note> list) async {
  final db = await database;
  final batch = db.batch();
  for (var note in list) {
    if (note.id == null)
      batch.insert(tableNote, note.toMap());
    else
      batch.update(tableNote, note.toMap());
  }
  await batch.commit(noResult: true);
}

Future<Note> findById(int id) async {
  final db = await database;
  List<Map> maps =
      await db.query(tableNote, where: '$columnNoteId = ?', whereArgs: [id]
  if (maps.length > 0) {
    return Note.fromMap(maps.first);
  }
  return null;
}

Future<List<Note>> getAll() async {
  final db = await database;
  List<Map> maps = await db.query(tableNote, orderBy: "$columnEditDate DESC
  return maps.map((e) => Note.fromMap(e)).toList();
}

Future<int> delete(int id) async {
  final db = await database;
  return await db
```

```
        .delete(tableNote, where: '$columnNoteId = ?', whereArgs: [id]);
  }

  Future<int> update(Note note) async {
    final db = await database;
    return await db.update(tableNote, note.toMap(),
        where: '$columnNoteId = ?', whereArgs: [note.id]);
  }
}
```

Arquivo: lib/data/todo_repository.dart

```
import 'package:sqflite/sqflite.dart';
import 'package:two_notes/data/database_helper.dart';

import '../model/todo.dart';

final String tableTodo = 'todo';
final String columnId = '_id';
final String columnDescription = 'description';
final String columnDone = 'done';
final String columnReminderDate = 'reminderDate';

class TodoRepository {
  Future<Database> database = DatabaseHelper.instance.database;

  static Future createTable(Database db) {
    return db.execute('''
          create table $tableTodo (
            $columnId integer primary key autoincrement not null,
            $columnDescription text not null,
            $columnDone integer not null,
            $columnReminderDate integer);
          ''');
  }

  Future<Todo> insert(Todo todo) async {
    final db = await database;
    todo.id = await db.insert(tableTodo, todo.toMap());
```

```
    return todo;
  }

  void insertAll(List<Todo> list) async {
    final db = await database;
    final batch = db.batch();
    for (var todo in list) {
      if (todo.id == null)
        batch.insert(tableTodo, todo.toMap());
      else
        batch.update(tableTodo, todo.toMap());
    }
    await batch.commit(noResult: true);
  }

  Future<Todo> findById(int id) async {
    final db = await database;
    List<Map> maps =
        await db.query(tableTodo, where: '$columnId = ?', whereArgs: [id]);
    if (maps.length > 0) {
      return Todo.fromMap(maps.first);
    }
    return null;
  }

  Future<List<Todo>> getAll() async {
    final db = await database;
    List<Map> maps = await db.query(tableTodo,
        orderBy: "$columnDone, $columnReminderDate ASC");
    return maps.map((e) => Todo.fromMap(e)).toList();
  }

  Future<int> delete(int id) async {
    final db = await database;
    return await db.delete(tableTodo, where: '$columnId = ?', whereArgs: [id]
  }

  Future<int> update(Todo todo) async {
    final db = await database;
```

```
    return await db.update(tableTodo, todo.toMap(),
        where: '$columnId = ?', whereArgs: [todo.id]);
  }
}
```

Arquivo: lib/model/note.dart

```
import 'dart:io';

import 'package:two_notes/data/note_repository.dart';

class Note {
  int id;
  String text;
  DateTime editDate;
  File image;

  Note({this.text, this.image}) {
    this.editDate = DateTime.now();
  }

  Map<String, dynamic> toMap() {
    var map = <String, dynamic>{
      columnText: text,
      columnEditDate: editDate?.millisecondsSinceEpoch,
      columnImage: image?.path,
    };
    if (id != null) {
      map[columnNoteId] = id;
    }
    return map;
  }

  Note.fromMap(Map<String, dynamic> map) {
    id = map[columnNoteId];
    text = map[columnText];
    if (map[columnEditDate] != null) {
      editDate = DateTime.fromMillisecondsSinceEpoch(map[columnEditDate])
    }
```

```
    if (map[columnImage] != null) {
      image = File(map[columnImage]);
    }
  }

  @override
  bool operator ==(Object other) {
    return other is Note &&
        this.id == other.id &&
        this.text == other.text &&
        this.editDate == other.editDate &&
        this.image?.path == other.image?.path;
  }

  @override
  int get hashCode => super.hashCode;
}
```

Arquivo: lib/model/todo.dart

```
import 'package:two_notes/data/todo_repository.dart';

class Todo {
  int id;
  String description;
  DateTime reminderDate;
  bool done;

  Todo({this.description, this.reminderDate, this.done = false});

  Map<String, dynamic> toMap() {
    var map = <String, dynamic>{
      columnDescription: description,
      columnDone: done == true ? 1 : 0,
      columnReminderDate: reminderDate?.millisecondsSinceEpoch,
    };
    if (id != null) {
      map[columnId] = id;
    }
```

```
    return map;
  }

  Todo.fromMap(Map<String, dynamic> map) {
    id = map[columnId];
    description = map[columnDescription];
    done = map[columnDone] == 1;
    if (map[columnReminderDate] != null) {
      reminderDate =
          DateTime.fromMillisecondsSinceEpoch(map[columnReminderDate]);
    }
  }
}
```

Arquivo: lib/screens/edit_note_screen.dart

```
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:provider/provider.dart';
import 'package:two_notes/business/notes_provider.dart';
import 'package:two_notes/model/note.dart';
import 'package:two_notes/utils/colors.dart';
import 'package:two_notes/widgets/header.dart';
import 'package:two_notes/utils/extensions.dart';

class EditNoteScreen extends StatefulWidget {
  EditNoteScreen({Key key, this.note}) : super(key: key);

  final Note note;

  @override
  _EditNoteScreenState createState() => _EditNoteScreenState();
}

class _EditNoteScreenState extends State<EditNoteScreen> {
  TextEditingController _controller;
  bool confirmarEnabled = false;
```

```
final picker = ImagePicker();
File image;

void initState() {
  super.initState();
  _controller = TextEditingController();
  _controller.text = widget.note?.text;
  image = widget.note?.image;
  confirmarEnabled = _controller.text.isNotEmpty;
}

void dispose() {
  _controller.dispose();
  super.dispose();
}

void _onBack(Notes notes) {
  if (widget.note != null) {
    notes.updateNote(widget.note, _controller.text, image);
  } else if (!_controller.text.isEmptyOrNull() || image != null) {
    notes.addNote(_controller.text, image);
  }
  Navigator.pop(context);
}

void _onTextChanged(String text) {
  setState(() {
    confirmarEnabled = _controller.text.isNotEmpty;
  });
}

void _onDelete(Notes notes) {
  if (widget.note != null) {
    notes.removeNote(widget.note);
  }
  Navigator.pop(context);
}

void _onRemovePhoto() {
```

```
    setState(() {
      image = null;
    });
  }

  Future _addImage() async {
    await showDialog(
        context: context,
        builder: (BuildContext context) {
          return SimpleDialog(
            title: const Text('Add photo'),
            children: <Widget>[
              SimpleDialogOption(
                onPressed: () {
                  _getImage(ImageSource.camera);
                  Navigator.pop(context);
                },
                child: const Text('Take photo'),
              ),
              SimpleDialogOption(
                onPressed: () {
                  _getImage(ImageSource.gallery);
                  Navigator.pop(context);
                },
                child: const Text('Choose from gallery'),
              ),
              SimpleDialogOption(
                onPressed: () {
                  Navigator.pop(context);
                },
                child: const Text('Cancel'),
              ),
            ],
          );
        });
  }

  Future _getImage(ImageSource source) async {
    try {
```

```
      final pickedFile = await picker.getImage(source: source);

      setState(() {
        if (pickedFile != null) {
          image = File(pickedFile.path);
        }
      });
    } catch (e) {}
  }

  @override
  Widget build(BuildContext context) {
    final notes = Provider.of<Notes>(context);
    final title = widget.note == null ? 'Create note' : 'Edit note';

    return Scaffold(
        backgroundColor: Colors.white,
        body: Container(
          decoration: BoxDecoration(
            gradient: LinearGradient(
              begin: Alignment.topCenter,
              end: Alignment.bottomCenter,
              colors: [TwoNotesColors.sorvete, Colors.white],
              tileMode: TileMode.repeated,
            ),
          ),
          child: SafeArea(
            child: Container(
              color: Colors.white,
              child: Column(
                children: [
                  Header(
                    title: title,
                    back: () {
                      _onBack(notes);
                    },
                    rightButton: IconButton(
                        icon: Icon(
                          Icons.delete,
```

```
                size: 24.0,
                color: Colors.white,
              ),
              onPressed: () {
                _onDelete(notes);
              }),
          ),
          if (image != null)
            Flexible(
              flex: 1,
              child: Padding(
                padding: const EdgeInsets.symmetric(
                    vertical: 12.0, horizontal: 16.0),
                child: Stack(
                    alignment: Alignment.topRight,
                    children: <Widget>[
                      Image.file(image),
                      Padding(
                        padding: const EdgeInsets.all(8.0),
                        child: CircleAvatar(
                          radius: 20,
                          backgroundColor: Colors.white,
                          child: IconButton(
                            icon: Icon(
                              Icons.close,
                              size: 20.0,
                              color: TwoNotesColors.black23,
                            ),
                            onPressed: _onRemovePhoto,
                          ),
                        ),
                      )
                    ]),
              ),
            ),
          Flexible(
            flex: 1,
            child: Padding(
              padding: const EdgeInsets.symmetric(horizontal: 16.
```

```
                        child: TextField(
                          maxLength: null,
                          maxLines: 100,
                          style: TextStyle(fontSize: 16),
                          onChanged: _onTextChanged,
                          controller: _controller,
                          decoration: InputDecoration(
                              border: InputBorder.none, hintText: 'Add text'),
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
          floatingActionButton: image == null
              ? FloatingActionButton(
                  onPressed: _addImage,
                  tooltip: 'Add',
                  child: Icon(
                    Icons.add_a_photo,
                    color: Colors.white,
                  ),
                )
              : null);
  }
}
```

Arquivo: lib/screens/edit_todo_dialog.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:two_notes/business/todos_provider.dart';
import 'package:two_notes/utils/date_formatter.dart';

import '../model/todo.dart';
import '../utils/colors.dart';
```

```
class EditTodoDialog extends StatefulWidget {
  EditTodoDialog({Key key, this.todo}) : super(key: key);

  final Todo todo;

  @override
  _EditTodoDialogState createState() => _EditTodoDialogState();
}

class _EditTodoDialogState extends State<EditTodoDialog> {
  TextEditingController _controller;
  DateTime selectedDate;
  bool confirmarEnabled = false;

  void initState() {
    super.initState();
    _controller = TextEditingController();
    _controller.text = widget.todo?.description;
    selectedDate = widget.todo?.reminderDate;
    confirmarEnabled = _controller.text.isNotEmpty;
  }

  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _dismiss() {
    Navigator.pop(context);
  }

  void _onConfirm(Todos todos) {
    if (_controller.text.isNotEmpty) {
      if (widget.todo != null) {
        widget.todo.reminderDate = selectedDate;
        widget.todo.description = _controller.text;
        todos.updateTodo(widget.todo);
      } else {
        final newTodo =
```

```
            Todo(description: _controller.text, reminderDate: selectedDate);
        todos.addTodo(newTodo);
      }
    }
    _dismiss();
  }

  void _openDatePicker() async {
    final DateTime initialDate = selectedDate ?? DateTime.now();
    final DateTime datePicked = await showDatePicker(
      context: context,
      initialDate: initialDate,
      firstDate: DateTime(2010),
      lastDate: DateTime(2030),
    );

    if (datePicked != null) {
      setState(() {
        selectedDate = DateTime(datePicked.year, datePicked.month,
            datePicked.day, initialDate.hour, initialDate.minute);
      });
    }
  }

  void _openTimePicker() async {
    final initialDate = selectedDate ?? DateTime.now();
    final TimeOfDay timePicked = await showTimePicker(
        context: context, initialTime: TimeOfDay.fromDateTime(initialDate));

    if (timePicked != null) {
      setState(() {
        selectedDate = DateTime(initialDate.year, initialDate.month,
            initialDate.day, timePicked.hour, timePicked.minute);
      });
    }
  }

  void _cleanDate() {
    setState(() {
```

```
      selectedDate = null;
    });
  }

  void _onTextChanged(String text) {
    setState(() {
      confirmarEnabled = _controller.text.isNotEmpty;
    });
  }

  @override
  Widget build(BuildContext context) {
    final todos = Provider.of<Todos>(context);
    final title = widget.todo == null ? 'Add to-do' : 'Edit to-do';

    return Dialog(
      child: Wrap(
        children: [
          Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              Container(
                color: TwoNotesColors.sorvete,
                height: 60,
                child: Padding(
                  padding: const EdgeInsets.only(top: 20.0, left: 16.0),
                  child: Text(
                    '$title',
                    style: TextStyle(fontSize: 20, color: Colors.white),
                  ),
                ),
              ),
            ],
          ),
          Padding(
            padding:
                const EdgeInsets.symmetric(horizontal: 16.0, vertical: 10
            child: TextField(
              maxLines: 1,
```

```
        maxLength: 255,
        onChanged: _onTextChanged,
        controller: _controller,
        decoration: InputDecoration(
            border: UnderlineInputBorder(),
            hintText: 'Add a description'),
      ),
    ),
    ButtonBar(
      alignment: MainAxisAlignment.start,
      children: [
        FlatButton.icon(
          label: Text(selectedDate == null
              ? 'Set date'
              : DateFormatter().toDate(selectedDate)),
          icon: Icon(Icons.calendar_today),
          textColor: TwoNotesColors.sorvete,
          color: Colors.white,
          onPressed: _openDatePicker,
        ),
        FlatButton.icon(
          label: Text(selectedDate == null
              ? 'Set time'
              : DateFormatter().toTime(selectedDate)),
          icon: Icon(Icons.alarm),
          textColor: TwoNotesColors.sorvete,
          color: Colors.white,
          onPressed: _openTimePicker,
        ),
        Visibility(
          child: IconButton(
            icon: Icon(Icons.close),
            color: TwoNotesColors.black23,
            onPressed: _cleanDate,
          ),
          visible: selectedDate != null,
        )
      ],
    ),
```

```
            ButtonBar(
              children: [
                OutlineButton(
                  child: Text('Cancel'),
                  color: TwoNotesColors.sorvete,
                  textColor: TwoNotesColors.sorveteDark,
                  onPressed: _dismiss,
                  highlightedBorderColor: TwoNotesColors.sorveteDark,
                ),
                FlatButton(
                  child: Text('Confirm'),
                  textColor: Colors.white,
                  color: TwoNotesColors.sorvete,
                  onPressed: confirmarEnabled
                      ? () {
                          _onConfirm(todos);
                        }
                      : null,
                ),
              ],
            ),
          ],
        ),
      );
  }
}
```

Arquivo: lib/screens/home_screen.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:two_notes/business/todos_provider.dart';
import 'package:two_notes/utils/date_formatter.dart';

import '../model/todo.dart';
import '../utils/colors.dart';

class EditTodoDialog extends StatefulWidget {
  EditTodoDialog({Key key, this.todo}) : super(key: key);
```

```
  final Todo todo;

  @override
  _EditTodoDialogState createState() => _EditTodoDialogState();
}

class _EditTodoDialogState extends State<EditTodoDialog> {
  TextEditingController _controller;
  DateTime selectedDate;
  bool confirmarEnabled = false;

  void initState() {
    super.initState();
    _controller = TextEditingController();
    _controller.text = widget.todo?.description;
    selectedDate = widget.todo?.reminderDate;
    confirmarEnabled = _controller.text.isNotEmpty;
  }

  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _dismiss() {
    Navigator.pop(context);
  }

  void _onConfirm(Todos todos) {
    if (_controller.text.isNotEmpty) {
      if (widget.todo != null) {
        widget.todo.reminderDate = selectedDate;
        widget.todo.description = _controller.text;
        todos.updateTodo(widget.todo);
      } else {
        final newTodo =
            Todo(description: _controller.text, reminderDate: selectedDate);
        todos.addTodo(newTodo);
```

```
      }
    }
    _dismiss();
  }

  void _openDatePicker() async {
    final DateTime initialDate = selectedDate ?? DateTime.now();
    final DateTime datePicked = await showDatePicker(
      context: context,
      initialDate: initialDate,
      firstDate: DateTime(2010),
      lastDate: DateTime(2030),
    );

    if (datePicked != null) {
      setState(() {
        selectedDate = DateTime(datePicked.year, datePicked.month,
            datePicked.day, initialDate.hour, initialDate.minute);
      });
    }
  }

  void _openTimePicker() async {
    final initialDate = selectedDate ?? DateTime.now();
    final TimeOfDay timePicked = await showTimePicker(
        context: context, initialTime: TimeOfDay.fromDateTime(initialDate

    if (timePicked != null) {
      setState(() {
        selectedDate = DateTime(initialDate.year, initialDate.month,
            initialDate.day, timePicked.hour, timePicked.minute);
      });
    }
  }

  void _cleanDate() {
    setState(() {
      selectedDate = null;
    });
```

```
}

void _onTextChanged(String text) {
  setState(() {
    confirmarEnabled = _controller.text.isNotEmpty;
  });
}

@override
Widget build(BuildContext context) {
  final todos = Provider.of<Todos>(context);
  final title = widget.todo == null ? 'Add to-do' : 'Edit to-do';

  return Dialog(
    child: Wrap(
      children: [
        Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            Container(
              color: TwoNotesColors.sorvete,
              height: 60,
              child: Padding(
                padding: const EdgeInsets.only(top: 20.0, left: 16.0),
                child: Text(
                  '$title',
                  style: TextStyle(fontSize: 20, color: Colors.white),
                ),
              ),
            ),
          ],
        ),
        Padding(
          padding:
              const EdgeInsets.symmetric(horizontal: 16.0, vertical: 10.0),
          child: TextField(
            maxLines: 1,
            maxLength: 255,
            onChanged: _onTextChanged,
```

```
        controller: _controller,
        decoration: InputDecoration(
            border: UnderlineInputBorder(),
            hintText: 'Add a description'),
      ),
    ),
    ButtonBar(
      alignment: MainAxisAlignment.start,
      children: [
        FlatButton.icon(
          label: Text(selectedDate == null
              ? 'Set date'
              : DateFormatter().toDate(selectedDate)),
          icon: Icon(Icons.calendar_today),
          textColor: TwoNotesColors.sorvete,
          color: Colors.white,
          onPressed: _openDatePicker,
        ),
        FlatButton.icon(
          label: Text(selectedDate == null
              ? 'Set time'
              : DateFormatter().toTime(selectedDate)),
          icon: Icon(Icons.alarm),
          textColor: TwoNotesColors.sorvete,
          color: Colors.white,
          onPressed: _openTimePicker,
        ),
        Visibility(
          child: IconButton(
            icon: Icon(Icons.close),
            color: TwoNotesColors.black23,
            onPressed: _cleanDate,
          ),
          visible: selectedDate != null,
        )
      ],
    ),
    ButtonBar(
      children: [
```

```
                OutlineButton(
                  child: Text('Cancel'),
                  color: TwoNotesColors.sorvete,
                  textColor: TwoNotesColors.sorveteDark,
                  onPressed: _dismiss,
                  highlightedBorderColor: TwoNotesColors.sorveteDark,
                ),
                FlatButton(
                  child: Text('Confirm'),
                  textColor: Colors.white,
                  color: TwoNotesColors.sorvete,
                  onPressed: confirmarEnabled
                      ? () {
                          _onConfirm(todos);
                        }
                      : null,
                ),
              ],
            ),
          ],
        ),
      );
  }
}
```

Arquivo: lib/screens/notes_tab.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:two_notes/business/notes_provider.dart';
import 'package:two_notes/model/note.dart';
import 'package:two_notes/widgets/note_item.dart';

import 'edit_note_screen.dart';

class NotesTab extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => _NotesTabState();
}
```

```
class _NotesTabState extends State<NotesTab> {
  void _editNote(Note note) {
    Navigator.push(
        context,
        MaterialPageRoute(
          builder: (BuildContext context) => EditNoteScreen(
            note: note,
          ),
        ));
  }

  @override
  Widget build(BuildContext context) {
    final notes = Provider.of<Notes>(context);
    notes.notes.sort((a, b) => b.editDate.compareTo(a.editDate));
    return notes.notes.isEmpty
        ? Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Image.asset(
                  'assets/images/empty_notes.png',
                  height: 200,
                ),
                Padding(
                  padding: const EdgeInsets.only(top: 20.0),
                  child: Text(
                    'No notes here yet!',
                    style: TextStyle(fontSize: 20),
                  ),
                )
              ],
            ),
          )
        : ListView.builder(
            itemCount: notes.notes.length,
            itemBuilder: (context, index) {
              return NoteItem(
```

```
                note: notes.notes[index],
                onClick: _editNote,
              );
            },
          );
  }
}
```

Arquivo: lib/screens/todos_tab.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:two_notes/business/todos_provider.dart';
import 'package:two_notes/model/todo.dart';
import 'package:two_notes/widgets/todo_item.dart';

import 'edit_todo_dialog.dart';

class TodosTab extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => _TodosTabState();
}

class _TodosTabState extends State<TodosTab> {
  void _editTodo(Todo todo) {
    showDialog(
      barrierDismissible: true,
      context: context,
      builder: (_) {
        return EditTodoDialog(
          todo: todo,
        );
      },
    );
  }

  @override
  Widget build(BuildContext context) {
    final todos = Provider.of<Todos>(context);
```

```
    return (todos.todos.isEmpty)
        ? Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Image.asset(
                  'assets/images/empty_todos.png',
                  width: 200,
                ),
                Padding(
                  padding: const EdgeInsets.only(top: 20.0),
                  child: Text(
                    'List is empty',
                    style: TextStyle(fontSize: 20),
                  ),
                )
              ],
            ),
          )
        : ListView.builder(
            itemCount: todos.todos.length,
            itemBuilder: (context, index) {
              return TodoItem(
                todo: todos.todos[index],
                onClick: _editTodo,
              );
            },
          );
  }
}
```

Arquivo: lib/utils/colors.dart

```
import 'package:flutter/material.dart';

abstract class TwoNotesColors {
  static const sorvete = Color(0xFF9ba3eb);
```

```
  static const sorveteDark = Color(0xFF737ee6);

  static const merengue = Color(0xFF6ce6db);
  static const merengueDark = Color(0xFF59bdb4);

  static const black23 = Color(0x3B000000);
  static const black80 = Color(0xB3000000);
}
```

Arquivo: lib/utils/date_formatter.dart

```
import 'package:intl/intl.dart';
import 'package:intl/date_symbol_data_local.dart';

class DateFormatter {
  DateFormatter() {
    initializeDateFormatting('pt_BR', null);
  }

  String toNoteDate(DateTime date) {
    if (isToday(date)) {
      return toTime(date);
    } else {
      return toDate(date);
    }
  }

  String toDate(DateTime date) {
    return DateFormat("d'/'M", 'pt_BR').format(date);
  }

  String toTime(DateTime date) {
    String pattern = "H'h'm";
    if (date.minute == 0) {
      pattern = "H'h'";
    } else if (date.minute < 10) {
      pattern = "H'h0'm";
    }
    return DateFormat(pattern, 'pt_BR').format(date);
```

```
  }

  String toDateAndTime(DateTime date) {
    String pattern = "d'/'M ";
    if (date.minute == 0) {
      pattern += "H'h'";
    } else if (date.minute < 10) {
      pattern += "H'h0'm";
    } else {
      pattern += "H'h'm";
    }
    return DateFormat(pattern, 'pt_BR').format(date);
  }

  String toTimeStamp(DateTime date) {
    return DateFormat("y'-'M'-'d'-'Hms", 'pt_BR').format(date);
  }

  bool isToday(DateTime date) {
    return date.day == DateTime.now().day;
  }
}
```

Arquivo: lib/utils/extensions.dart

```
extension EmptyOrNull on String {
  bool isEmptyOrNull() {
    return this == null || this.isEmpty;
  }
}
```

Arquivo: lib/utils/notifications.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.d
import 'package:flutter_native_timezone/flutter_native_timezone.dart';
import 'package:rxdart/subjects.dart';
```

```
import 'package:timezone/data/latest.dart' as tz;
import 'package:timezone/timezone.dart' as tz;
import 'package:two_notes/model/todo.dart';

class ReceivedNotification {
  ReceivedNotification({
    @required this.id,
    @required this.title,
    @required this.body,
    @required this.payload,
  });

  final int id;
  final String title;
  final String body;
  final String payload;
}

class NotificationUtils {
  static final FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugi
      FlutterLocalNotificationsPlugin();

  static final BehaviorSubject<ReceivedNotification>
      didReceiveLocalNotificationSubject =
      BehaviorSubject<ReceivedNotification>();

  static const MethodChannel platform =
      MethodChannel('dexterx.dev/flutter_local_notifications_example');

  static Future<void> configureLocalTimeZone() async {
    tz.initializeTimeZones();
    String timeZoneName;
    try {
      timeZoneName = await FlutterNativeTimezone.getLocalTimezone();
      tz.setLocalLocation(tz.getLocation(timeZoneName));
    } on PlatformException {
      timeZoneName = 'Failed to get the timezone.';
    }
  }
```

```
static void configureNotificationsPlugin() async {
  const AndroidInitializationSettings initializationSettingsAndroid =
      AndroidInitializationSettings('app_icon');

  final IOSInitializationSettings initializationSettingsIOS =
      IOSInitializationSettings(
          requestAlertPermission: false,
          requestBadgePermission: false,
          requestSoundPermission: false,
          onDidReceiveLocalNotification:
              (int id, String title, String body, String payload) async
            didReceiveLocalNotificationSubject.add(ReceivedNotification
                id: id, title: title, body: body, payload: payload));
          });

  final InitializationSettings initializationSettings =
      InitializationSettings(
          android: initializationSettingsAndroid,
          iOS: initializationSettingsIOS);

  await flutterLocalNotificationsPlugin.initialize(initializationSettin
      onSelectNotification: (String payload) async {
    if (payload != null) {
      debugPrint('notification payload: $payload');
    }
  });
}

static void requestPermissions() {
  flutterLocalNotificationsPlugin
      .resolvePlatformSpecificImplementation<
          IOSFlutterLocalNotificationsPlugin>()
      ?.requestPermissions(
        alert: true,
        badge: true,
        sound: true,
      );
}
```

```
  static Future<void> zonedScheduleNotification(Todo todo) async {
    await flutterLocalNotificationsPlugin.zonedSchedule(
        todo.id,
        '${todo.description ?? ''}',
        '',
        tz.TZDateTime.from(todo.reminderDate, tz.local),
        const NotificationDetails(
            android: AndroidNotificationDetails(
                CHANNEL_ID, CHANNEL_NAME, CHANNEL_DESCRIPTION)),
        androidAllowWhileIdle: true,
        uiLocalNotificationDateInterpretation:
            UILocalNotificationDateInterpretation.absoluteTime);
  }

  static Future<void> cancelNotification(int id) async {
    await flutterLocalNotificationsPlugin.cancel(id);
  }

  static const CHANNEL_ID = 'com.example.two_notes';
  static const CHANNEL_NAME = 'Reminders';
  static const CHANNEL_DESCRIPTION = 'Show silently';
}
```

Arquivo: lib/widgets/header.dart

```
import 'package:flutter/material.dart';
import 'package:two_notes/utils/colors.dart';

class Header extends StatelessWidget {
  final VoidCallback back;
  final String title;
  final Widget rightButton;

  const Header({Key key, this.back, this.title, this.rightButton})
      : super(key: key);

  Widget build(BuildContext context) {
    return Container(
```

```
child: Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: <Widget>[
    Material(
      color: TwoNotesColors.sorvete,
      child: InkWell(
        child: IconButton(
          icon: Icon(
            Icons.arrow_back,
            size: 24.0,
            color: Colors.white,
          ),
          onPressed: back,
        ),
      ),
    ),
    Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text(title,
              style: TextStyle(
                  fontSize: 20,
                  color: Colors.white,
                  fontWeight: FontWeight.w600)),
        ],
      ),
    ),
    Material(
      color: TwoNotesColors.sorvete,
      child: Padding(
        padding: const EdgeInsets.only(right: 4.0),
        child: InkWell(
          child: (rightButton != null)
              ? rightButton
              : SizedBox(
                  width: 20,
                ),
        ),
```

```
            ),
          ),
        ],
      ),
      width: double.infinity,
      height: 56,
      color: TwoNotesColors.sorvete,
    );
  }
}
```

Arquivo: lib/widgets/note_item.dart

```
import 'package:flutter/material.dart';
import 'package:two_notes/model/note.dart';
import 'package:two_notes/utils/colors.dart';
import 'package:two_notes/utils/date_formatter.dart';

class NoteItem extends StatelessWidget {
  final Note note;

  @required
  final Function(Note) onClick;

  const NoteItem({Key key, this.note, this.onClick});

  @override
  Widget build(BuildContext context) {
    return InkWell(
      onTap: () {
        onClick(note);
      },
      child: Padding(
        padding: const EdgeInsets.only(right: 16.0, left: 16.0, top: 12.0),
        child: Row(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: <Widget>[
            Expanded(
              child: Column(
```

```
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text(
                        note.text != null && note.text.isNotEmpty
                            ? note.text
                            : 'Note without text',
                        overflow: TextOverflow.ellipsis,
                        maxLines: 1,
                        style: TextStyle(
                            fontSize: 16,
                            fontWeight: FontWeight.bold,
                            color: TwoNotesColors.black80),
                      ),
                      Text(
                        DateFormatter().toNoteDate(note.editDate),
                        style: TextStyle(fontSize: 12),
                      ),
                      Padding(
                        padding: const EdgeInsets.only(top: 12.0),
                        child: Divider(
                          color: TwoNotesColors.black23,
                        ),
                      ),
                    ],
                  ),
                ),
              ],
            ),
          ),
        );
    }
}
```

Arquivo: lib/widgets/todo_item.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:two_notes/business/todos_provider.dart';
import 'package:two_notes/model/todo.dart';
```

```
import 'package:two_notes/utils/colors.dart';
import 'package:two_notes/utils/date_formatter.dart';

class TodoItem extends StatefulWidget {
  @required
  final Todo todo;

  @required
  final Function(Todo) onClick;

  const TodoItem({Key key, this.todo, this.onClick});

  @override
  _TodoItemState createState() => _TodoItemState();
}

class _TodoItemState extends State<TodoItem> {
  @override
  Widget build(BuildContext context) {
    final todos = Provider.of<Todos>(context);
    return InkWell(
      onTap: () {
        widget.onClick(widget.todo);
      },
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 10.0, vertical: 4.0),
        child: Row(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: <Widget>[
            Checkbox(
                value: widget.todo.done,
                onChanged: (bool value) {
                  todos.toggleTodo(widget.todo, value);
                }),
            Expanded(
              child: Padding(
                padding: const EdgeInsets.only(left: 8.0),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
```

```
                    children: [
                      Text(
                        widget.todo.description,
                        style: TextStyle(fontSize: 16),
                      ),
                      if (widget.todo.reminderDate != null)
                        Text(
                          DateFormatter().toDateAndTime(widget.todo.reminde
                          style: TextStyle(
                              fontSize: 12, color: TwoNotesColors.merengueD
                          )
                    ],
                  ),
                ),
              ),
              IconButton(
                icon: Icon(Icons.close),
                color: TwoNotesColors.black23,
                onPressed: () {
                  todos.removeTodo(widget.todo);
                },
              )
            ],
          ),
        ),
      );
  }
}
```

Arquivo: lib/main.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.d
import 'package:provider/provider.dart';

import 'package:two_notes/business/notes_provider.dart';
import 'package:two_notes/data/note_repository.dart';
import 'package:two_notes/business/todos_provider.dart';
import 'package:two_notes/data/todo_repository.dart';
```

```
import 'package:two_notes/screens/home_screen.dart';
import 'package:two_notes/utils/colors.dart';
import 'package:two_notes/utils/notifications.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  await NotificationUtils.configureLocalTimeZone();

  NotificationUtils.configureNotificationsPlugin();

  final NotificationAppLaunchDetails notificationAppLaunchDetails =
      await NotificationUtils.flutterLocalNotificationsPlugin
          .getNotificationAppLaunchDetails();

  runApp(App(notificationAppLaunchDetails: notificationAppLaunchDetails));
}

class App extends StatelessWidget {
  App({this.notificationAppLaunchDetails});

  final NotificationAppLaunchDetails notificationAppLaunchDetails;

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => Notes(repository: NoteRepository()),
      lazy: false,
      child: ChangeNotifierProvider(
        create: (context) => Todos(repository: TodoRepository()),
        lazy: false,
        child: MaterialApp(
          title: '2Notes',
          theme: ThemeData(
              primaryColor: TwoNotesColors.sorvete,
              accentColor: TwoNotesColors.merengue,
              primaryColorDark: TwoNotesColors.sorveteDark,
              primaryTextTheme:
                  TextTheme(headline6: TextStyle(color: Colors.white))),
```

```
            home: HomeScreen(notificationAppLaunchDetails, title: '2Notes')
          ),
        ),
      );
  }
}
```