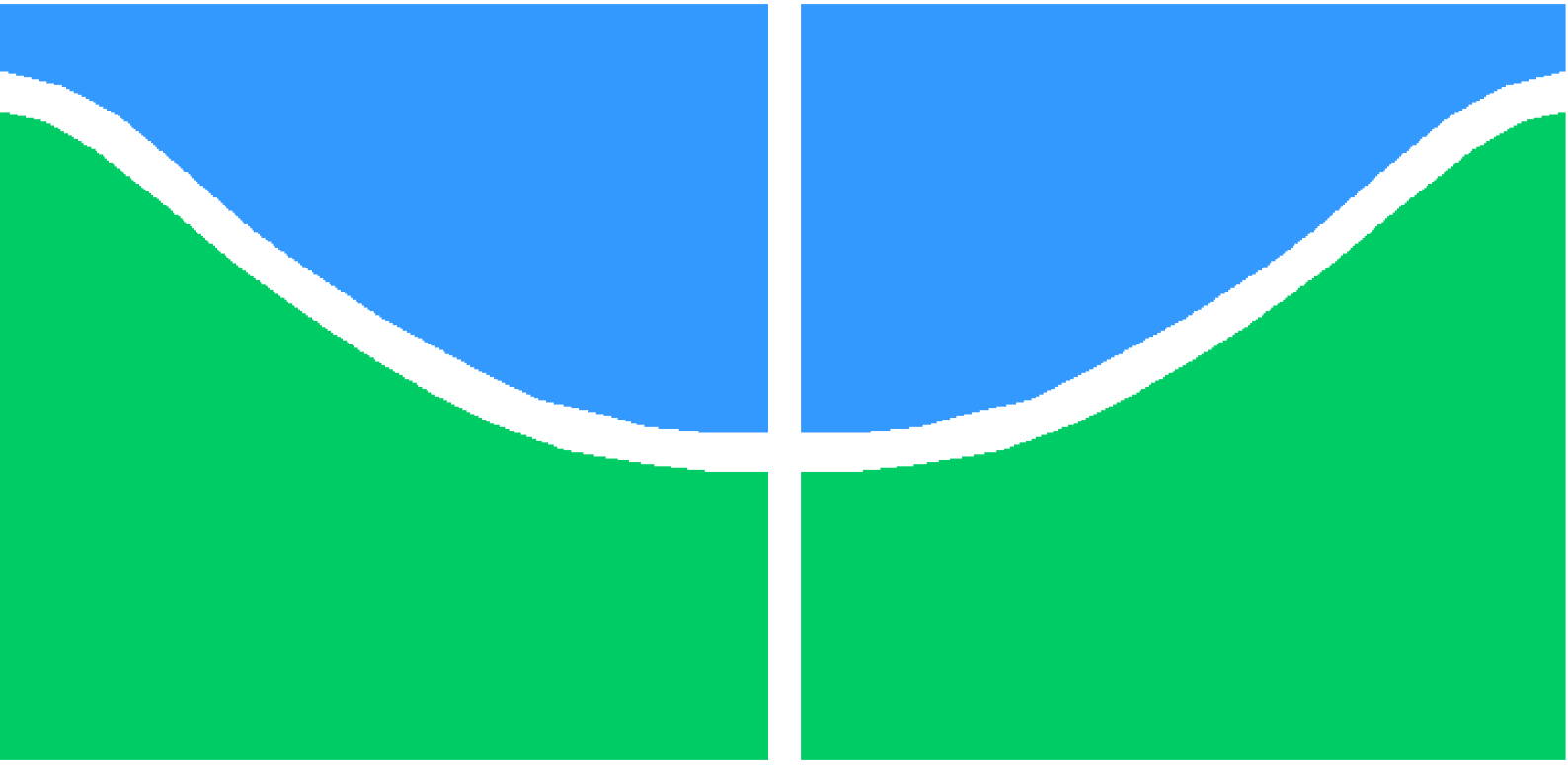

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataforma

Autor: Beatriz Rezener Dourado Matos, João Gabriel de Britto e Silva

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2016

Beatriz Rezener Dourado Matos, João Gabriel de Britto e Silva

Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataforma

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2016

Beatriz Rezener Dourado Matos, João Gabriel de Britto e Silva

Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataforma/ Beatriz Rezener Dourado Matos, João Gabriel de Britto e Silva. – Brasília, DF, 2016-

61 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2016.

1. mobile. 2. multiplataforma. I. Prof. Dr. Paulo Roberto Miranda Meirelles. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataforma

CDU 02:141:005.6

Beatriz Rezener Dourado Matos, João Gabriel de Britto e Silva

Estudo comparativo entre o desenvolvimento de aplicativos móveis utilizando plataformas nativas e multiplataforma

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 18 de julho de 2016:

Prof. Dr. Paulo Roberto Miranda Meirelles
Orientador

Prof. Dr. Rodrigo Bonifácio de Almeida
Convidado 1

Prof. Msc. Renato Coral Sampaio
Convidado 2

Brasília, DF
2016

"Tudo que havia ao seu redor era o branco da neve,
que reluzia a pureza, a ingenuidade e a ignorância.
Mal sabia o guerreiro que o branco marcava o início
de sua árdua, porém gloriosa jornada.
(Naves, Wesley)"

Resumo

O desenvolvimento de aplicativos móveis intensificou-se nos últimos anos devido ao crescimento acelerado e popularização dos *smartphones*. Cada empresa do ramo móvel possui seu próprio sistema operacional, loja de aplicativos, parcela de mercado e ambientes de desenvolvimento. No entanto, quanto mais sistemas diferentes existem, maior o esforço, custo e tempo para desenvolver um *app* para todas as plataformas existentes. Surgiu então, o conceito de desenvolvimento multiplataforma, com a premissa de codificar apenas uma vez e abranger várias plataformas. Nesse contexto, o presente trabalho busca definir vantagens e desvantagens da abordagem multiplataforma quando comparada com a abordagem nativa. Por meio da análise de um exemplo de uso, que consistiu na recriação de um aplicativo nativo, implementado originalmente para a plataforma iOS, utilizando o *framework* Ionic, foi possível comparar e comprovar empiricamente os dados obtidos na literatura. As ferramentas multiplataforma, atualmente, não possuem mais as limitações apontadas pela literatura, pois evoluíram muito rapidamente ao longo dos últimos cinco anos. Concluiu-se, ao fim deste trabalho, que cada abordagem tem um momento certo para ser utilizada, não sendo uma melhor que a outra, mas apenas diferentes entre si, e deve-se avaliar cada caso, considerando-se uma série de fatores para escolher qual abordagem utilizar.

Palavras-chaves: desenvolvimento. móvel. multiplataforma. nativo. ionic.

Abstract

Mobile application development has intensified in recent years due to rapid growth and popularization of smartphones. Each company of mobile branch has its own operating system, application store, market share and development environments. However, the more different systems exist, the greater the effort, cost and time to develop an app for all existing platforms. Then came the concept of cross-platform development, with the premise of code only once and span multiple platforms. In this context, this paper seeks to define advantages and disadvantages of the cross-platform approach compared with the native approach. Through an analysis of study, which consisted of the recreation of a native application, originally implemented for iOS platform, using the Ionic framework, it was possible to compare and empirically verify the data obtained in the literature. Cross-platform tools currently no longer have the limitations mentioned in the literature, they evolved very rapidly over the last five years. It was concluded at the end of this work, that each approach has a certain time to be used, not being better than the other, but only different, and developers must evaluate each case considering a number of factors to choose which approach is better to be used in each context.

Key-words: development. mobile. cross. platform. native. ionic.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CSS	Cascade Style Sheet
CLI	Command-Line Interface
DAO	Data Access Object
DSL	Dynamic Shared Libraries
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
HTML	HiperText Markup Language
IDE	Integrated Development Enviroment
LPS	Linha de Produto de Software
MVC	Model-View-Controller
OHA	Open Handset Alliance
SDK	Software Development Kit
SO	Sistema Operacional
SPA	Single Page Application

Sumário

1 Introdução

Com o aumento da quantidade de dispositivos móveis no mercado, a demanda por aplicações móveis também aumentou (CEVALLOS, 2014). De início, a única opção que havia era desenvolver aplicações específicas para uma plataforma, utilizando todo o ambiente daquela plataforma, por exemplo, no caso do iOS, a liguagem de programação era o *Objective-C* com a *IDE Xcode* e o *SDK* do iOS (HEITKöTTER; HANSCHKE; MAJCHRZAK, 2013).

Segundo Prezotto e Boniati (2014), essa forma de desenvolvimento é conhecida como desenvolvimento **nativo** e é aquele no qual um aplicativo é projetado e construído especificamente para uma plataforma. Todas as funcionalidades da plataforma estão disponíveis sem restrição e existem padrões de interface gráfica e experiência de usuário específicos, que ajudam o usuário a entender como aquele aplicativo funciona, já que todos os outros aplicativos daquela plataforma seguem os mesmos padrões (CORRAL; JANES; REMENCIUS, 2012).

No entanto, com o aumento exponencial da demanda por aplicativos, surgiu a necessidade de desenvolvimento rápido para muitas plataformas distintas, o que era caro e difícil, visto que, cada plataforma tinha um ambiente de desenvolvimento completamente diferente das demais, o que demandava da equipe muitos conhecimentos diferentes para poder desenvolver e manter vários aplicativos (PREZOTTO; BONIATI, 2014).

Segundo Heitkötter, Hanschke e Majchrzak (2013), com esse novo panorama do mercado, surgiu a necessidade da criação de apenas um *app* que pudesse ser executado em várias plataformas. Essa forma de desenvolvimento ficou conhecida como **multiplataforma** (*cross-platform*). Consiste na criação de uma página *web*, normalmente utilizando *HTML*, *CSS* e *JavaScript*, que pode ser mostrada dentro de uma *web view* embutida em um aplicativo nativo, ou seja, é feito apenas um código, que é executado e mostrado em um *container* dentro de um aplicativo nativo (STARK, 2010; HEITKöTTER; HANSCHKE; MAJCHRZAK, 2013).

Dessa forma, é possível desenvolver um mesmo aplicativo que será executado dentro de várias plataformas diferentes, sem a necessidade de ter uma equipe especialista em cada plataforma. No entanto, não é possível ter acesso total às funcionalidades da plataforma, assim como no desenvolvimento nativo, e também não existe padrão de interface ou de experiência de usuário que sirva para várias plataformas ao mesmo tempo, podendo causar uma sensação ruim na usabilidade e experiência de uso do aplicativo (CORRAL; JANES; REMENCIUS, 2012).

O presente trabalho visa auxiliar desenvolvedores, no momento inicial da criação de

aplicativos para dispositivos móveis, a escolherem a melhor abordagem que se encaixa no contexto em que estão inseridos, considerando as reais vantagens e desvantagens presentes atualmente no desenvolvimento multiplataforma.

1.1 Justificativa

Sabendo que existem diversas plataformas diferentes para dispositivos móveis atualmente e que há uma crescente necessidade de mercado de abarcar o maior número possível de plataformas, o mais rápido possível, com a melhor qualidade e menor custo possíveis, o presente trabalho visa auxiliar desenvolvedores *mobile* a compreender melhor o mundo do desenvolvimento móvel e tomar a decisão de desenvolver nativamente ou multiplataforma com mais consciência das vantagens e desvantagens de cada abordagem.

1.2 Objetivos

O objetivo deste trabalho é definir por meio de pesquisa e comprovação empírica as vantagens e desvantagens de desenvolvimento multiplataforma de aplicações móveis em relação ao desenvolvimento nativo. Além desse objetivo principal, existem ainda outros objetivos secundários que precisam ser atingidos. São eles:

- Pesquisar sobre o desenvolvimento de aplicativos móveis;
- Compreender a arquitetura das duas principais plataformas móveis da atualidade (iOS e Android);
- Desenvolver uma réplica de um aplicativo criado nativamente para uma plataforma, utilizando tecnologias multiplataforma;
- Comparar o desenvolvimento do aplicativo multiplataforma com o aplicativo nativo;

Com base nos objetivos traçados para o presente trabalho, a questão problema a ser respondida é:

Quais as vantagens e desvantagens do desenvolvimento multiplataforma de aplicações móveis em relação ao desenvolvimento nativo?

A partir da pergunta feita, foram formuladas duas hipóteses acerca do tema, que são apresentadas a seguir.

- **H1**: As plataformas *cross-development* estão evoluídas ao ponto de apresentarem mais vantagens do que desvantagens.

- **H2**: Ao longo do tempo, as possíveis desvantagens do desenvolvimento multiplataforma serão sanadas ou mitigadas.

1.3 Organização do Trabalho

Este trabalho foi dividido em cinco capítulos. No Capítulo 2, é feito um estudo sobre como é feito hoje e como evoluiu ao longo dos últimos anos, o desenvolvimento de aplicações móveis, Também foi feito um comparativo para avaliar vantagens e desvantagens entre as duas abordagens de desenvolvimento. No Capítulo 3, é apresentada a metodologia adotada para o desenvolvimento do trabalho, o planejamento feito e como será executado, para que o trabalho possa ser replicado e/ou continuado futuramente. No Capítulo 4, é descrito uma análise de exemplo de uso, realizado durante o trabalho, sobre o desenvolvimento de aplicativos móveis. Por fim, no Capítulo 5, apresentam-se as considerações preliminares acerca do desenvolvimento multiplataforma e da realização da análise do exemplo de uso, bem como apresenta-se um cronograma preliminar do desenvolvimento da continuação deste trabalho no Trabalho de Conclusão de Curso II.

2 Desenvolvimento de aplicativos móveis

Segundo El-Kassas et al. (2015), o desenvolvimento de aplicativos móveis diferencia-se do desenvolvimento de outros tipos de *software* por possuir particularidades e restrições. Os desenvolvedores devem ter em mente aspectos como as capacidades do dispositivo móvel, a mobilidade, as especificações dos dispositivos, o design e navegabilidade de interface do usuário, segurança e privacidade do usuário. De acordo com Corral, Janes e Remencius (2012), aplicações móveis são desenvolvidas de forma dinâmica e lançadas no mercado em pequenos ciclos. Os produtos finais costumam ser de pequeno porte e comercializados a preços baixos. As equipes de desenvolvimento também tendem a ser pequenas. Algumas particularidades a serem consideradas no desenvolvimento *mobile* são listadas a seguir, baseadas em El-Kassas et al. (2015).

- **Limitações de recursos**: As capacidades de processamento e de armazenamento dos dispositivos móveis são limitadas e a conectividade pode ser afetada por ser uma plataforma móvel.

- **Ecossistema heterogêneo**: O desenvolvimento de aplicações móveis encontra-se em um contexto heterogêneo devido aos diferentes sistemas operacionais e à grande quantidade de dispositivos distintos, com variações de poder computacional, configurações de *hardware* e tamanhos de tela. Essas singularidades devem ser consideradas no desenvolvimento de *apps*, o que pode implicar na necessidade de diferentes versões de uma mesma aplicação.

- **Experiência de uso**: é importante que as aplicações sejam simples e que possuam uma interface amigável para atender às expectativas dos usuários, caso contrário, devido ao grande número de aplicações disponíveis, o *app* pode ser descartado e substituído por outros presentes nas lojas de aplicativos.

- **Manutenção**: plataformas *mobiles* passam por constantes atualizações que podem afetar *apps* já desenvolvidos, a ponto de torná-los inutilizáveis ou causarem desconforto aos usuários. Para que a aplicação continue a funcionar corretamente, são necessárias frequentes manutenções e atualizações. Atualizações de *apps* desenvolvidos para diferentes plataformas implicam na alteração de cada uma das versões desenvolvidas.

2.1 Desenvolvimento Nativo

Aplicações nativas são desenvolvidas com o uso de ferramentas e linguagens de programação específicas para determinada plataforma, usando o *SDK* e *frameworks* providos por ela. Os *apps* ficam vinculados a esse ambiente, executando apenas nos dispositivos da plataforma alvo (EL-KASSAS et al., 2015). Caso haja a necessidade de implementação para múltiplas plataformas, a aplicação deve ser desenvolvida separadamente para cada uma delas (HEITKOTTER; HANSCHKE; MAJCHRZAK, 2013).

Posto que o *app* é desenvolvido em ambiente próprio, segue-se os padrões técnicos, de interface e de experiência de usuário determinados pela plataforma, provendo um *look and feel* de aplicação nativa ao usuário. Aplicações nativas tem fácil acesso, por meio de *APIs* fornecidas pelas plataformas, a recursos dos dispositivos móveis, como sensores, câmera, *GPS*, contatos e *e-mail* (EL-KASSAS et al., 2015).

Nas seções a seguir serão descritas as arquiteturas das plataformas iOS e Android, duas das plataformas mais conhecidas e utilizadas para desenvolvimento nativo de aplicações *mobile* (JOBE, 2013).

2.1.1 iOS

O sistema operacional iOS foi lançado juntamente com o primeiro *iPhone* criado em Janeiro de 2007 e funciona como uma interface entre as aplicações desenvolvidas pelos programadores (*apps*) e o *hardware* dos dispositivos (*iPhone*, *iPad*, *iPod*) (Apple Inc., 2007). Dessa forma, a comunicação com o *hardware* do dispositivo se dá por meio de um conjunto bem definido de interfaces do sistema, o que facilita o desenvolvimento de *apps* que funcionam entre os variados tipos de *hardware* dos dispositivos da Apple (Apple Inc., 2016b).

Como recomendação, a Apple explica que deve-se preferir o uso de camadas mais elevadas, pois as camadas de mais alto nível possuem abstrações orientadas à objeto de funcionalidades das camadas mais baixas. Isso torna o desenvolvimento mais fácil, pois reduz a quantidade de código que deve ser criado e mantém funcionalidades complexas das camadas mais baixas encapsuladas por meio das interfaces. No entanto, não há problema em usar funcionalidades presentes nas camadas mais baixas, se essas não estiverem disponíveis por meio de abstrações nas camadas superiores (Apple Inc., 2014b).

A maioria das interfaces disponíveis para uso são disponibilizadas por meio de *frameworks*, que são diretórios, que podem ser adicionados ao projeto no *Xcode*, contendo *DSLs* e os recursos necessários como, imagens, aplicativos auxiliares e arquivos *header*, para o *framework* funcionar corretamente (Apple Inc., 2014b).

2.1.1.1 Arquitetura iOS

Sua arquitetura é baseada nas camadas listadas, conforme a Figura 1, a seguir.

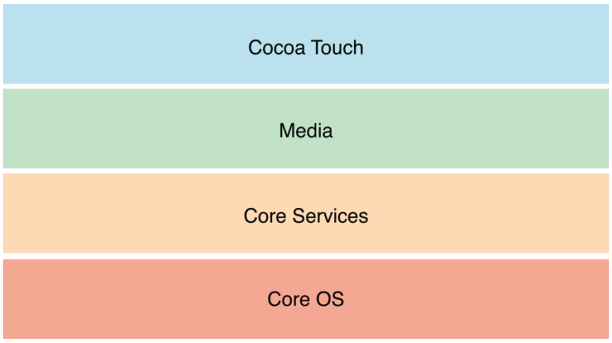

Figura 1: Arquitetura iOS. Fonte: (Apple Inc., 2014b).

As camadas do iOS são explicadas com mais detalhes a seguir.

- A camada *Cocoa Touch* é a camada mais alto nível onde são fornecidos serviços básicos de interação com o usuário como entrada baseada em toques, notificações *Push* e outras tecnologias necessárias para melhorar a experiência do usuário como multitarefas, Continuidade (*Handoff*) e *AirDrop*, além de *frameworks* de alto nível que permitem acesso a funcionalidades do sistema como *AddressBook* para contatos, *EventKit* para eventos relacionados ao calendário e *MapKit* para mapas (Apple Inc., 2014b).

- A camada logo abaixo da *Cocoa Touch* é a camada *Media* que contém tecnologias e *frameworks* necessários para a implementação de experiências multimídia com áudio, vídeo e gráficos (Apple Inc., 2014b).

- A próxima camada, logo abaixo da camada *Media*, é a camada *Core Services*. Essa camada está mais próxima do *hardware* e portanto possui acesso a funcionalidades de mais baixo nível como localização, telefonia, *threads* e *SQLite*. Aqui residem dois dos *frameworks* mais importantes do iOS que são o *Foundation* e o *Core Foundation*, ambos relacionados com o gerenciamento de dados e alguns serviços e definem todos os tipos básicos de dados que todos os *apps* usam, como por exemplo, coleções, *strings*, data e hora, *sockets* e *threads* (Apple Inc., 2014b).

- A última camada é a camada *Core OS*, na qual as funcionalidades de mais baixo nível são construídas e provavelmente utilizadas por outros *frameworks* em outras camadas. Se a aplicação possui requisitos de segurança ou comunicação com acessórios externos mais complicados, é possível usar as funcionalidades dessa camada (Apple Inc., 2014b).

2.1.1.2 Desenvolvimento iOS

O *SDK* iOS permite que os desenvolvedores criem suas aplicações e a testem em emuladores. Contudo, para a utilização de recursos avançados e para a distribuição na *App Store*, loja de aplicativos da Apple (Apple Inc., 2016a), é exigida uma licença do *Apple Developer Program*, com custo de US$99 anuais.

Para a distribuição de aplicativos iOS, além da licença citada, é necessário submeter a aplicação para avaliação e aprovação da Apple antes de ir para a *App Store*. Para agilizar o processo de aprovação, é necessário que o aplicativo esteja de acordo com as diretrizes estabelecidas pela Apple, como as orientações de interface gráfica e as orientações de revisão da *App Store* (Apple Inc., 2016c).

Desenvolver aplicativos para a plataforma iOS requer um computador com o sistema operacional Mac OS e o ambiente de desenvolvimento *Xcode* (HEITKOTTER; HANSCHKE; MAJCHRZAK, 2013). O desenvolvimento é atrelado a duas linguagens de programação, o *Objective-C* e o *Swift*. O Objective-C constitui-se em um superconjunto da linguagem de programação C, da qual herda a maior parte da sua sintaxe, incluindo tipos primitivos e fluxo de declarações, e adiciona sintaxe própria para definição de classes e métodos (Apple Inc., 2014a). O *Swift* foi lançado pela Apple como um sucessor do *Objective-C*. É uma linguagem *software* livre e pode ser integrada a códigos *Objective-C* (Apple Inc., 2016d). Pouco tempo após o seu lançamento, tornou-se uma das linguagens de programação mais utilizadas no mundo (REBOUAS et al., 2016). Segundo a Apple, o *Swift* possui uma sintaxe concisa e intuitiva (Apple Inc., 2016d).

2.1.2 Android

O sistema operacional Android foi criado pela *start-up* homônima Android Inc. em outubro de 2003. Em agosto de 2005 foi adquirida pela empresa Google, que lançou em novembro de 2007, juntamente com a *OHA*, o sistema Android, *open-source* e baseado no *kernel* do Linux. Uma semana depois, foi liberada a primeira versão do *SDK* para Android. O sistema foi concebido originalmente para câmeras fotográficas, no entanto, foi percebido por seus criadores um mercado maior no ramo da telefonia e desviou-se o foco para *smartphones*, competindo diretamente com Symbian e Windows Mobile (PAPAJORGJI, 2015).

2.1.2.1 Arquitetura Android

A arquitetura Android é baseada nas camadas apresentadas na Figura 2 e são explicadas com mais detalhes a seguir.

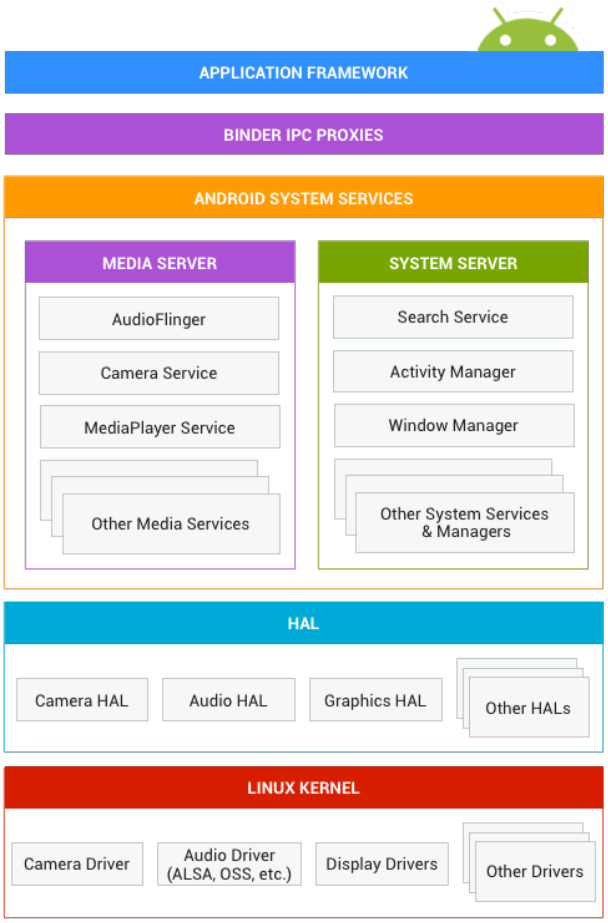

Figura 2: Arquitetura Android. Fonte: (Android, 2016a).

- A camada mais elevada é a *Application Framework*, onde as aplicações construídas pelos desenvolvedores são instaladas (Android, 2016a).
- A camada logo a abaixo da *Application Framework* é a *Binder IPC Proxies*, onde *IPC* significa *Inter-Process Comunication*. É a camada que faz uma ponte entre as aplicações instaladas na camada superior com a próxima camada, a *System Services*. É uma interface que permite que *APIs* de alto nível interajam com serviços do sistema que residem na camada abaixo dela (Android, 2016a).
- Abaixo da *Binder IPC Proxies* fica a *System Services*, que , por meio de módulos, permite acesso a *hardwares* específicos. Cada serviço presente nessa camada, foi desenvolvido para gerenciar um componente específico, como busca e notificações. Os serviços foram divididos em duas categorias, *Media* e *System*, explicadas a seguir (Android, 2016a).
  - *Media Server*: são os serviços responsáveis por gerenciar conteúdos de mídia como gravação e reprodução de audio e vídeo.
  - *System Server*: são serviços responsáveis por gerenciar os demais tipos de serviço do sistema como notificações e *windows*.

- Abaixo da camada *System Services* fica a camada *HAL* que permite que fornecedores de *hardware* criem interfaces e *drivers* para os *hardwares* que ele oferecem. Com isso, é possível criar novas funcionalidades e implementá-las sem afetar o resto das camadas do sistema (Android, 2016a).

- A camada mais baixa é a *Linux Kernel* que é uma versão do *kernel* do Linux com algumas modificações como uma gerenciamento de memória mais avançados e próprio para dispositivos móveis e funcionalidades para dispositivos embarcados (Android, 2016a).

2.1.2.2 Desenvolvimento Android

Para o desenvolvimento de aplicações Android é recomendada a linguagem de programação *Java*. Não é requerido sistema operacional específico e a *IDE* indicada pelo Google é o *Android Studio*, oficial para desenvolvimento de aplicativos Android (Android, 2016b).

Para a publicação na loja de aplicativos do Google, *Google Play Store*, é cobrada uma taxa única de US$25. Antes de serem publicados, os aplicativos passam por um processo de revisão para assegurar que não violam as políticas estabelecidas pela loja (MEIER, 2015).

2.2 Desenvolvimento *Cross-platform*

Para o desenvolvimento de uma aplicação nativa que possa ser executada em diferentes plataformas, se faz necessária a criação de uma para cada plataforma, o que exige tempo, investimento e uma equipe com conhecimento nas variadas tecnologias usadas pelas plataformas alvo. Soluções *cross-platform* possibilitam a implementação de uma aplicação que pode ser executada em diferentes plataformas (EL-KASSAS et al., 2015). Conforme Heitkotter, Hanschke e Majchrzak (2013), essas aplicações são comumente categorizadas em *webapps* e aplicações híbridas.

Web apps são aplicações implementadas utilizando tecnologias *web*, levando em conta aspectos intrínsecos aos dispositivos móveis, como tamanho de tela e modo de uso (HEITKOTTER; HANSCHKE; MAJCHRZAK, 2013). Segundo Stark (2010), *web apps* não são instalados no dispositivo e não ficam disponíveis nas lojas de aplicativos, em vez disso, são acessados pelos *browsers* dos dispositivos. Eles não possibilitam o uso de alguns recursos de *hardware* dos dispositivos como sensores e *GPS* (HEITKOTTER; HANSCHKE; MAJCHRZAK, 2013). Conforme Heitkotter, Hanschke e Majchrzak (2013), *web apps* costumam ter a aparência e o comportamento de páginas *web* comuns, divergindo dos elementos de tela padrão providos pelas plataformas *mobile*.

A abordagem híbrida surgiu como uma combinação do uso das tecnologias *web* junto à possibilidade de acesso a funcionalidades nativas (HEITKOTTER; HANSCHKE; MAJCHRZAK, 2013). Em essência, aplicações híbridas são *web app* empacotadas em um aplicativo nativo (STARK, 2010).

Nas próximas subseções serão descritas algumas das tecnologias utilizadas para o desenvolvimento multiplataforma.

2.2.1 PhoneGap e Cordova

PhoneGap é um *framework* para desenvolvimento de aplicativos híbridos, criado pela empresa Nitobi. Após a empresa ser comprada pela Adobe Systems Inc. o PhoneGap teve seu código doado para a Apache Software Foundation para garantir que outras empresas pudessem contribuir (BEZERRA; SCHIMIGUEL, 2016).

Como o PhoneGap é uma marca registrada de propriedade da Adobe, na Apache teve seu nome alterado para Cordova. Tanto o *Apache Cordova* quanto o *Adobe PhoneGap*, são gratuitos e *open sources*, no entanto, o PhoneGap possui um ambiente integrado com serviços da Adobe, como o *PhoneGap Build* (BEZERRA; SCHIMIGUEL, 2016).

Cordova permite que aplicações não nativas tenham acesso a funcionalidades nativas do dispositivo como sensores, contatos e câmera. No entanto, o Cordova apenas consegue fazer um aplicativo criado em *HTML* rodar como se fosse nativo em um dispositivo com Android e iOS, mas não consegue imitar a usabilidade e aparência dos dispositivos nativos (BEZERRA; SCHIMIGUEL, 2016).

Para preencher essa lacuna, outros *frameworks* foram criados em cima do Cordova como um complemento, provendo bibliotecas *HTML* e *CSS* para a criação de um *front-end* que se aproxime o máximo possível da usabilidade e experiência de usuário de aplicações móveis nativas (BEZERRA; SCHIMIGUEL, 2016).

O Cordova possui muitos *plugins* para poder acessar funcionalidades específicas do aparelho em que está sendo executado. Esses devem ser instalados no projeto do aplicativo via *CLI* (BEZERRA; SCHIMIGUEL, 2016).

A fim de se evitar confusões em relação aos nomes dados ao PhoneGap e ao Cordova, pode-se entender o PhoneGap como uma distribuição do *Apache Cordova*, mantido pela Adobe Systems Inc.

2.2.2 AngularJS

HTML é a principal linguagem de marcação para a criação de páginas *web*. No entanto, os conteúdos exibidos são estáticos e o *HTML* não fornece suporte para conteúdos dinâmicos. AngularJS é um *framework open source*, mantido pelo Google, que permite

estender a sintaxe do *HTML* para poder criar páginas *web* com conteúdos dinâmicos (BEZERRA; SCHIMIGUEL, 2016). Com ele é possível utilizar *tags* não nativas do *HTML* para gerenciar conteúdos que ainda não foram dispostos na tela (GOOGLE, 2016).

O usuário faz uma ação por meio da interface gráfica, o AngularJS interpreta a ação e faz uma requisição para o servidor pedindo apenas o conteúdo necessário, que é diferente do que já está sendo mostrado para o usuário. O servidor retorna o conteúdo específico e o AngularJS apresenta o novo conteúdo inserindo-o corretamente junto com o antigo já mostrado (URSINO; CAPANNA, 2015). O ciclo do AngularJS é esquematizado na Figura 3 a seguir.

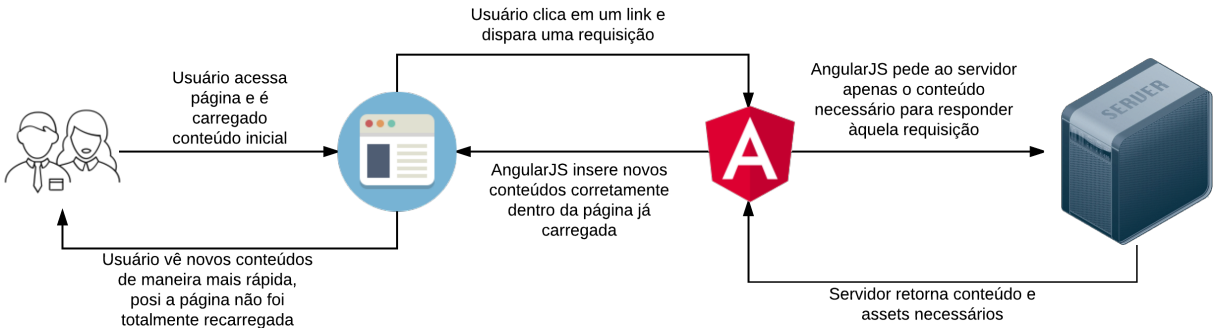

Figura 3: Ciclo de atualização com AngularJS. Fonte: Baseado em (URSINO; CAPANNA, 2015).

Foi projetado para criação de *SPAs* e com isso faz com que as páginas sejam mais rápidas, visto que, apenas é carregado o conteúdo que precisa ser mostrado, deixando outros conteúdos para serem carregados dinamicamente depois, de acordo com as interações do usuário (RODRíGUEZ; BALDRICH, 2015).

2.2.3 Ionic

Ionic é um *framework software* livre para desenvolvimento de aplicativos híbridos utilizando tecnologias *web* como *HTML*, *CSS* e *JavaScript* otimizadas para dispositivos móveis (DRIFTY, 2016c).

Foi criado pela empresa Drifty Co. em 2013 com base na necessidade que seus clientes apresentavam de criarem *apps*. Foi projetado para ser muito performático e funcionar com padrões e tecnologias *web* modernas (DRIFTY, 2016a).

Criado sobre os *frameworks* Cordova e AngularJS, com apenas um código é possível criar aplicativos para várias plataformas como iOS e Android, por exemplo, dentre outras. O Ionic conta ainda com um conjunto de ferramentas para auxiliar no desenvolvimento dos aplicativos (DRIFTY, 2016c).

2.2.3.1 Arquitetura de um projeto Ionic

Um projeto Ionic pode ser dividido em cinco camadas, listadas e detalhadas a seguir.

- *Views*: Camada de apresentação, onde as informações são mostradas para o usuário. Pode ser comparada como o seu homônimo no padrão *MVC*. Muitas vezes são chamadas de *templates*, pois é a forma com o AngularJS se refere a elas. São, normalmente, arquivos *HTML* separados, que são chamados e inseridos quando necessário pelo AngularJS. É possível utilizar *data binding* nas Views para poder estabelecer uma conexão com a controladora e compartilhar informações entre as duas camadas (DRIFTY, 2016d).

- *Controllers*: É a camada responsável por controlar o fluxo de dados e lógica da aplicação. Também pode ser comparada com a camada homônima no padrão *MVC*. Ela é responsável por apresentar ao usuário as *Views* e chamar as camadas de dados (*Services/Factories*) para ligar os dados reais da aplicação, por meio de *data binding*, à interface gráfica. As controladoras possuem uma variável chamada *scope*, que possui todas as informações que são necessárias para a criação da *View* (DRIFTY, 2016d).

- *Data(Services/Factories)*: É a camada que se aproxima da camada *model* do padrão *MVC*. Encapsula dados da aplicação e provê esses dados, geralmente, por meio de um *web service*. Essa camada responde às requisições da controladora com os dados a serem utilizados para a criação da *View* e para serem mostrados para o usuário (DRIFTY, 2016d).

- *App Configuration*: Nesta camada, as controladoras são ligadas às suas interfaces por meio de rotas. É possível, também, criar rotas padrão, para o caso de não haver nenhuma rota que esteja sendo identificada, o que poderia fazer o sistema quebrar (DRIFTY, 2016d).

- *Directives*: Essa camada serve para especificar comportamentos específicos em elementos de uma página *HTML*, ou seja, elas são um elemento ou um atributo que podem iniciar um comportamento específico definido pelo programador. É possível, por exemplo, criar uma diretiva para sempre colocar uma imagem padrão caso uma validação retorne um valor falso (DRIFTY, 2016d).

2.2.3.2 Desenvolvimento Ionic

Para o desenvolvimento de aplicações utilizando Ionic, não é necessário nenhum equipamento específico, assim como no desenvolvimento Android. É necessário apenas um computador com qualquer *SO*, no entanto, para o desenvolvimento do *app iOS*, ainda é

necessário que se possua um computador com o sistema operacional *Mac OS* e com o Xcode (DRIFTY, 2016b). Pode-se utilizar qualquer *IDE* de acordo com as preferências do programador, não há qualquer tipo de recomendação técnica por parte da equipe ou presente na documentação do Ionic. A linguagem utilizada é o *JavaScript*, usando ainda *HTML* e *CSS* para criação da interface gráfica.

2.2.3.3 Ferramentas de Apoio

O Ionic provê, além do *HTML* e *CSS* otimizado para dispositivos móveis, uma série de ferramentas, que compõem um ecossistema de apoio e fornecem velocidade e facilidade no desenvolvimento de aplicativos híbridos. Essas ferramentas são listadas a seguir com uma breve explicação do que cada uma pode fazer (DRIFTY, 2016c).

- **Ionic Framework**: É o *framework front-end* para criação de aplicativos híbridos com tecnologias *web* construído em cima do *Apache Cordova*. Facilita a criação de aplicativos com interface gráfica e interações muito semelhantes aos dispositivos nativos, de modo que a experiência de usuário não seja diferente no uso de uma aplicação nativa e multiplataforma (DRIFTY, 2016c).

- **Ionic Creator**: Com essa ferramenta *on-line* é possível criar protótipos funcionais utilizando *drag and drop* , que podem ser executados em dispositivos ou simuladores, e depois exportar o projeto criado para a inserção de lógicas de negócio. Ela é gratuita para teste, ou seja, é possível criar um protótipo e fazer o *download* do código para continuar o desenvolvimento, no entanto, no momento em que este trabalho foi conduzido, só era possível fazer a pré-visualização em dispositivos reais para usuários pagantes (DRIFTY, 2016e).

- **Ionic Lab**: É uma ferramenta *desktop* para testar o projeto que está sendo criado com o Ionic. Nela é possível executar o projeto em templates prontos de iOS e Android, rodar os simuladores de cada plataforma, gerenciar plugins do Cordova e ver um console com *logs* relativos à execução do projeto. Encapsula o *CLI* do Ionic com uma interface gráfica simples e intuitiva (DRIFTY, 2016f).

- **Ionic Platform**: É a plataforma *on-line* do Ionic que possui serviços de *back-end* em nuvem para análise de dados de uso, usuários, configuração de *Push Notifications*, criação dos pacotes nativos e dar *deploy* nas aplicações criadas sem a necessidade de resubmeter o aplicativo para as respectivas lojas (DRIFTY, 2016g).

- **Ionic Playground**: É uma ferramenta *on-line* para criar e testar códigos *HTML*, *CSS* e *JavaScript* já integrado com a plataforma AngularJS. Nela é possível editar códigos nas três linguagens citadas e ver em tempo real as alterações feitas aparecerem dentro de um *template* de dispositivo móvel, sem a necessidade de salvar,

compilar ou executar o projeto e sem ter que configurar o ambiente necessário para o desenvolvimento (DRIFTY, 2016h).

- **Ionic View App**: É um aplicativo disponível para as plataformas iOS e Android onde é possível testar o aplicativo criado no Ionic diretamente em um dispositivo real como se fosse nativo. É possível compartilhar os projetos criados para serem testados por clientes e testadores sem a necessidade de passar pelas lojas de aplicativos de cada plataforma. É semelhante ao serviço *Testflight* da Apple para testar aplicativos, ainda em desenvolvimento, e receber *feedbacks* dos clientes e testadores que o testarem (DRIFTY, 2016i).

2.3 Comparativo (Nativo x *Cross-platform*)

Cada plataforma possui seu próprio ambiente de desenvolvimento, arquitetura, ferramentas, mercado de distribuição e público alvo (SHAKSHUKI et al., 2013). E cada abordagem de desenvolvimento possui suas características próprias e inerentes à forma como se dá o projeto, desenvolvimento e distribuição dos *apps* (CORRAL; JANES; REMENCIUS, 2012).

Algo importante que deve ser feito pela equipe de desenvolvimento é decidir para qual plataforma desenvolver, tendo sempre em vista que ao não criar o aplicativo para uma plataforma, uma grande parcela do mercado pode estar sendo desconsiderada, o que acaba por reduzir o alcance da aplicação e por consequência os lucros obtidos com o *app* (CORRAL; JANES; REMENCIUS, 2012).

Para o desenvolvimento de várias aplicações para cada plataforma de maneira nativa, é preciso passar por todo o processo de desenvolvimento novamente, pois para cada plataforma há uma arquitetura distinta que deve ser seguida, padrões de código e interface de usuário e *APIs* próprias (HOLZINGER; TREITLER; SLANY, 2012). Dessa forma, cada aplicação deve ser codificada considerando as diferentes arquiteturas, componentes e *guidelines*, customizadas o máximo possível para ser a mesma aplicação em plataformas distintas e por fim, gerados novos executáveis e distribuídos nas diferentes lojas de aplicativos (CORRAL; JANES; REMENCIUS, 2012).

Isso implica em planejar cada *app* com base em uma plataforma distinta, por mais que se trate do mesmo aplicativo, o que acaba por encarecer e tornar mais lento o desenvolvimento (EL-KASSAS et al., 2015). Por exemplo, existem componentes que não existem em todas as arquiteturas (*navigationController* e *tabBarController* presentes na plataforma iOS, por exemplo) e que devem ser considerados no projeto do *app* para as demais plataformas (SHAKSHUKI et al., 2013).

No entanto, quando se pensa em desenvolvimento multiplataforma, o projeto se

torna agnóstico em relação à plataforma, pois o projeto é feito pensando-se apenas em uma arquitetura, um ambiente de desenvolvimento e um conjunto único de ferramentas e não de acordo com as características de cada plataforma. Após o desenvolvimento do núcleo da aplicação, podem ser feitas customizações para adaptar melhor o aplicativo para cada plataforma alvo da distribuição, criando assim um produto final verdadeiramente multiplataforma (CORRAL; JANES; REMENCIUS, 2012).

Na Figura 4 são apresentadas as diferenças no processo de desenvolvimento de um *app* utilizando tecnologias *cross-plataform* e nativas.

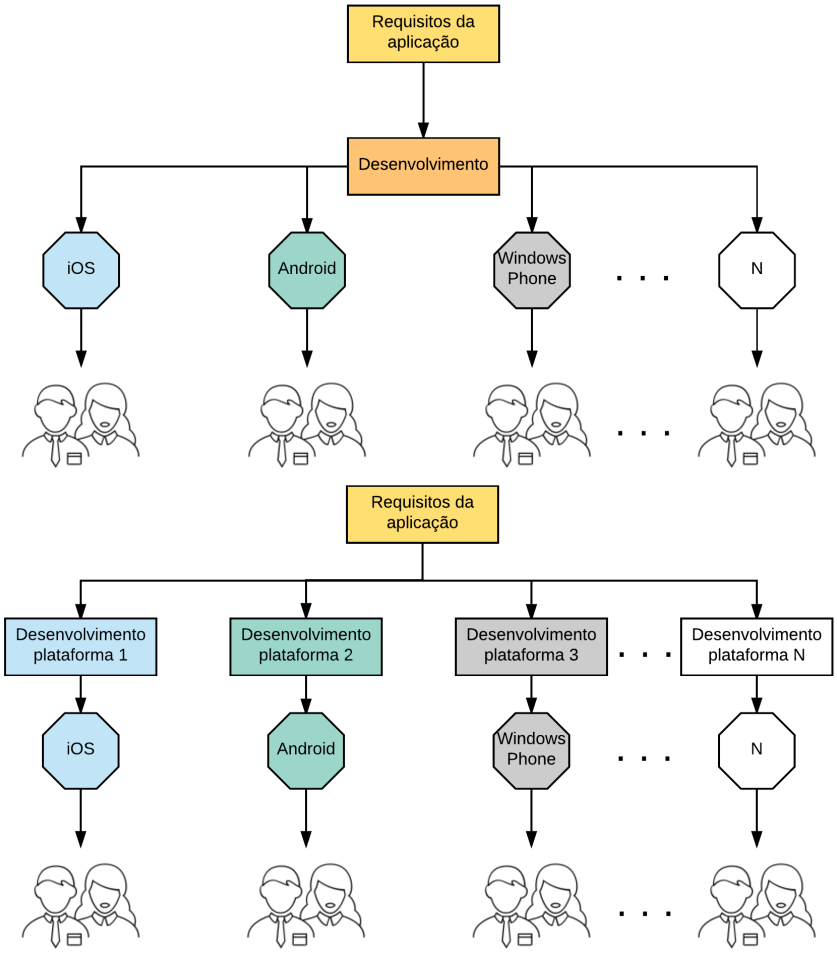

Figura 4: Desenvolvimento *Cross-plataform* x Nativo. Fonte: Baseado em (CORRAL; JANES; REMENCIUS, 2012)

A Tabela 1 sumariza as principais vantagens e desvantagens de cada tipo de desenvolvimento para uma melhor compreensão de cada abordagem. As informações contidas na tabela foram reunidas de vários trabalhos encontrados durante a pesquisa realizada. Esses trabalhos são Corral, Janes e Remencius (2012), Holzinger, Treitler e Slany (2012), Charland e Leroux (2011), BARTH (2014) e El-Kassas et al. (2015).

	Vantagens	Desvantagens
Nativo	• Explora todas as capacidades do dispositivos • Maior performance • Oferece experiência nativa de usabilidade para o usuário • Integração próxima com o sistema e outros aplicativos	• Necessário um aplicativo por plataforma • Mais caro e mais difícil de manter, por ter que manter vários *apps* diferentes • Requer domínio de vários ambientes e linguagens para cada plataforma
Multiplataformas	• Só é preciso dominar uma linguagem e um ambiente • Desenvolve apenas um código e pode distribuir em várias lojas de *apps*, o que faz com que o alcance do *app* seja maior • Reduz custo, tempo e esforço de desenvolvimento e manutenção	• Não possui acesso a todas as funcionalidades do dispositivo • Menor performance comparada ao nativo, linguagens interpretadas e compiladas • Não possui a mesma usabilidade e experiência de uso das aplicações nativas • Não possui as últimas atualizações lançadas pelo *SO*, pois para cada atualização do *SO*, é preciso ser feita uma atualização na plataforma *cross*

Tabela 1: Vantagens e desvantagens das abordagens de desenvolvimento nativo e multiplataforma

3 Metodologia

Neste capítulo são descritas as pesquisas feitas e os procedimentos realizados a fim de atingir os objetivos do trabalho, para detalhar como se deu o desenvolvimento do mesmo e, dessa forma, fornecer insumos para a sua correta reprodução futuramente por outros pesquisadores.

Este trabalho baseia-se na comparação entre duas abordagens de desenvolvimento móvel, são elas, nativa e multiplataforma. Com isso, o objetivo principal é responder a seguinte questão problema:

Quais as vantagens e desvantagens do desenvolvimento multiplataforma de aplicações móveis em relação ao desenvolvimento nativo?

A partir da questão feita, as duas hipóteses formuladas acerca do tema, são apresentadas e explicadas a seguir.

- **H1**: As plataformas *cross-development* estão evoluídas ao ponto de apresentarem mais vantagens do que desvantagens.
  - As ferramentas para desenvolvimento *cross-plataform* apresentam, atualmente, benefícios suficientes para serem consideradas uma opção viável ou talvez melhor que a abordagem nativa a depender da situação que envolve a aplicação a ser construída.
- **H2**: Ao longo do tempo, as possíveis desvantagens do desenvolvimento multiplataforma serão sanadas ou mitigadas.
  - Observando a evolução das ferramentas multiplataforma ao longo dos últimos anos, pode-se perceber que não apresentam mais os gargalos citados na literatura.

Para responder a questão problema será feita uma pesquisa sobre aplicações móveis e duas de suas abordagens de desenvolvimento: **nativa** e **multiplataforma**. Para isso, serão utilizados artigos e materiais *on-line* para definição de como é esse desenvolvimento e, com isso, será feita uma comparação entre os dois modos.

Após o estudo e a definição de como se dá esse desenvolvimento específico de *software*, será feita uma pesquisa de plataformas para desenvolvimento *cross-platform* e então será selecionada uma ferramenta.

Encerrada a fase de pesquisas, será realizado um primeiro estudo de uso, que consistirá em replicar um aplicativo nativo para plataforma iOS utilizando a ferramenta selecionada, para comprovar empiricamente as diferenças pesquisadas e descritas no Capítulo 2 deste trabalho.

No decorrer da realização do estudo, que será descrito mais adiante no Capítulo 4, serão anotadas as diferenças na implementação das mesmas funcionalidades no ambiente nativo e no multiplataforma, e após isso, ambas as abordagens poderão ser comparadas de maneira empírica com embasamento teórico oriundo das pesquisas feitas anteriormente.

Com base nas experiências obtidas no estudo realizado, serão feitas as considerações preliminares acerca dos resultados obtidos e um planejamento do que poderá ser feito na continuação deste trabalho.

3.1 Trabalhos relacionados

Durante a pesquisa conduzida, foram encontrados alguns trabalhos relacionados com o tema abordado por este trabalho, como Prezotto e Boniati (2014) que implementou um aplicativo utilizando a ferramenta multiplataforma *Apache Cordova*. Concluiu-se que a abordagem multiplataforma não é melhor que a nativa, mas que há situações onde cada abordagem se encaixa melhor.

Bezerra e Schimiguel (2016) também fez a implementação de um aplicativo utilizando ferramenta multiplataforma, nesse caso, PhoneGap. Concluiu-se que há a necessidade de avaliar os requisitos da aplicação a ser criada para se tomar a melhor decisão, sobre qual abordagem optar. Cabe ressaltar, no entanto, que ambos os trabalhos não realizaram o desenvolvimento nativo para uma comparação mais precisa entre as duas formas de desenvolvimento tratadas neste trabalho.

Charland e Leroux (2011) faz um estudo comparativo teórico sobre desenvolvimento nativo e multiplataforma, no entanto, não faz uma comparação prática entre as duas abordagens. Concluiu-se que o desenvolvimento híbrido é a solução mais provável no embate *nativo vs cross*, no entanto, há de se avaliar as necessidades da aplicação e do negócio.

Em Corral, Janes e Remencius (2012), é feito um levantamento sobre potenciais vantagens e desvantagens da abordagem *cross-platform* sob a perspectiva de três *stakeholders* distintos: Usuário, Desenvolvedor e Provedor de Plataformas. Concluiu-se que os usuários são os principais guias do mercado e as suas preferências é que devem definir o futuro do desenvolvimento móvel. No entanto, não foi feito nenhum estudo prático comparando as duas abordagens alvo deste trabalho.

Embora a pesquisa tenha revelado que está sendo discutida qual a melhor aborda-

gem para o desenvolvimento móvel, os estudo feitos, em sua maioria, são de três a cinco anos atrás, fazendo-se necessárias novas avaliações do desenvolvimento multiplataforma, devido à evolução das ferramentas e *frameworks* neste meio tempo. Não foram encontrados trabalhos recentes com a proposta de desenvolvimento de um mesmo *app* utilizando as duas abordagens, para se ter um comparativo mais embasado das características de cada abordagem, suas vantagens e desvantagens.

3.2 Planejamento do Exemplo de Uso

Nas subseções a seguir, são apresentados os critérios usados pelos autores para a seleção do projeto a ser recriado e da ferramenta a ser utilizada para um primeiro estudo de uso, assim como um planejamento do desenvolvimento do projeto.

3.2.1 Seleção do projeto

Para a realização do exemplo de uso, foi escolhido o aplicativo Mini Farma, criado nativamente para a plataforma iOS. A escolha se deu por alguns fatores, listados a seguir:

- É de propriedade de um dos autores deste trabalho, o que facilita na obtenção e no entendimento do código;
- Explora funcionalidades nativas do dispositivo, como localização, que serão explicitadas mais adiante;
- É um aplicativo pequeno e simples, o que o torna ideal para ser refeito dentro do tempo de execução do trabalho;

3.2.2 Seleção das plataformas

Existem, atualmente, muitas ferramentas e *frameworks* para o desenvolvimento de aplicações móveis multiplataforma. Dentre elas, a escolha do Ionic se deu por alguns fatores, listados a seguir.

- É um *framework* sob uma licença de *software* livre;
- Possui um ambiente de apoio gratuito e robusto, com muitas ferramentas e tutoriais disponíveis;
- Utiliza AngularJS e Cordova que são dois *frameworks* estáveis, muito conhecidos e utilizados e mantidos por duas grandes empresas;
- Possui muitos *plugins* disponíveis e uma comunidade muito ativa para resolução de problemas;

- Consegue criar aparência e usabilidade próximas às das plataformas nativas;
- Possui uma arquitetura projetada para otimizar a performance nos dispositivos móveis;

3.2.3 Planejamento do desenvolvimento

Após definidos o projeto e a ferramenta que serão utilizadas, o próximo passo é planejar como será feito o desenvolvimento do *app* multiplataforma. A replicação do projeto passou por algumas etapas, citadas e detalhadas a seguir.

- Configuração do *framework* Ionic e suas dependências nas máquinas dos desenvolvedores;
- Definição e implantação de ambiente de desenvolvimento igual para ambos os desenvolvedores;
- Familiarização com *framework* Ionic e seu ambiente por meio de criação de projetos teste e tutoriais;
- Familiarização com AngularJS e sua arquitetura;
- Separação do aplicativo Mini Farma nas suas principais funcionalidades a serem recriadas:
- Relato de observações acerca das diferenças, dificuldades, facilidades, vantagens e desvantagens;

4 Exemplo de Uso

Com o intuito de comparar o desenvolvimento de aplicações móveis utilizando a abordagem nativa e multiplataforma, foi realizado um estudo preliminar. Para isso, um aplicativo feito em iOS nativo foi recriado utilizando o *framework* Ionic. A seguir, o aplicativo escolhido é detalhado em termos de funcionalidades e recursos utilizados.

4.1 Descrição do projeto selecionado

Mini Farma é um aplicativo criado inicialmente para a plataforma iOS que serve para controle dos medicamentos que as pessoas possuem em casa em suas "farmacinhas" particulares.

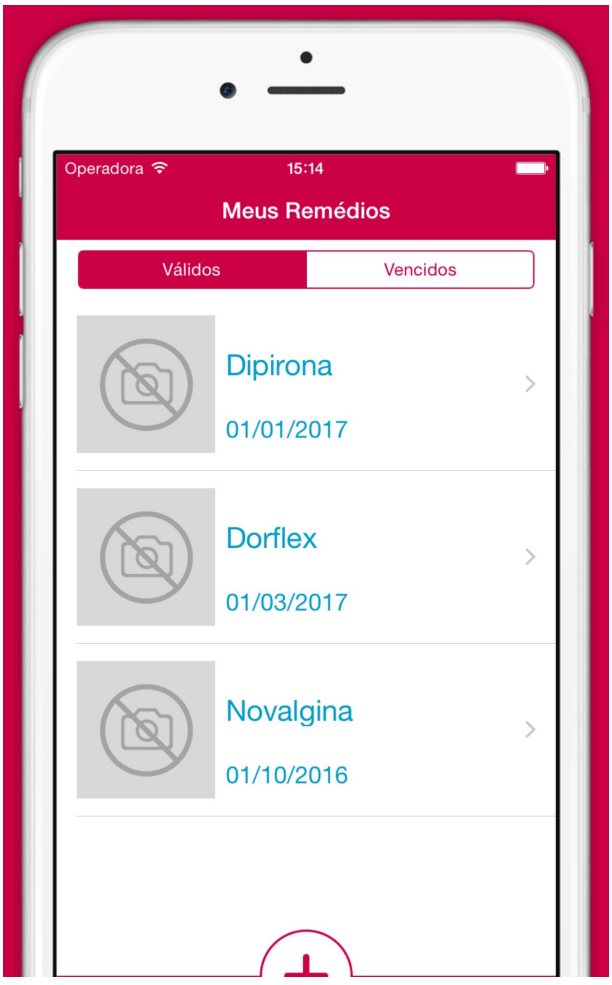

Figura 5: Tela inicial do aplicativo Mini Farma.

O aplicativo utiliza recursos nativos do sistema, os quais são descritos a seguir.

- **Localização geográfica**: Define a posição geográfica do dispositivo que o usuário está utilizando para salvar a localização da farmácia onde ele comprou um medicamento para, caso seja necessário, explicar para alguém onde a farmácia fica. Também é possível com base nessa localização da farmácia e do usuário, traçar uma rota para levá-lo diretamente à farmácia em questão;

- **Notificações locais**: Diferente das notificações *Push*, as notificações locais são criadas e agendadas na central de notificações do dispositivo e o sistema se encarrega de entregá-las corretamente de acordo com parâmetros definidos pelo aplicativo. No caso do Mini Farma, as notificações são usadas para lembrar o usuário, na data e hora corretas, de que o mesmo deve tomar seus medicamentos;

- **Ligação**: As notificações podem conter ações que executam um determinado bloco de código dentro do aplicativo. No Mini Farma, uma das notificações possíveis é a de aviso de pouca quantidade ou remédio esgotado, no caso do medicamento ter acabado. Quando essa notificação é enviada, pode ser feita uma ligação diretamente pela ação da notificação para o número da farmácia cadastrado no aplicativo. Dessa forma, o usuário pode solicitar uma nova quantidade de medicamento diretamente com a farmácia.

- **Câmera**: Para facilitar a identificação dos medicamentos, é possível tirar uma foto com a câmera do dispositivo para cada remédio cadastrado. Além de uma foto para o medicamento em si, é possível tirar uma foto da receita dele, caso haja;

- **Rolo de câmera**: Se o usuário já tiver uma foto que o ajude a identificar o seu medicamento ou da receita do mesmo salva no rolo de câmera, é possível escolher a foto sem precisar tirar uma nova;

Com exceção da Ligação, todos os outros recursos nativos do sistema precisam necessariamente serem autorizados pelo usuário para funcionar. Caso o usuário não autorize, o aplicativo fica com funcionalidades reduzidas.

Como banco de dados foi usado o SQLite, por meio do *framework* externo e *open-source* *FMDB*[1], disponível no Github;

A arquitetura do sistema foi criada com base no padrão de projeto *MVC*, possuindo uma camada *DAO* para comunicação com o banco de dados. No entanto, o *MVC* tradicional difere em alguns aspectos do *MVC* usado no iOS.

No iOS, o *MVC* trás uma controladora ligada à classe de *view*, que é chamado de *ViewController*, responsável por criar a ponte entre a interação do usuário pela *view* com as classes modelos. Dessa forma, as *ViewControllers* têm a responsabilidade de repassar

[1] <https://github.com/ccgus/fmdb>

as entradas do usuário para a modelo e controlar a *view* para apresentar os resultados que a modelo retornar.

Diferente do *MVC* tradicional, no qual a controladora apenas repassa as informações que estão entrando na fronteira da aplicação, as *ViewControllers* têm, além dessa responsabilidade, também a responsabilidade de instanciar e gerenciar a *view* no tocante à organização dos elementos e apresentação das informações.

Outra observação sobre as diferenças entre os dois *MVCs*, é que no *MVC* tradicional pode haver uma controladora apenas para todas as modelos ou uma controladora por modelo, mas nenhuma ligada diretamente a uma *view*, no entanto, no iOS há uma *ViewController* por *view*, podendo haver ainda uma controladora relacionada com cada modelo conforme o padrão tradicional do *MVC*.

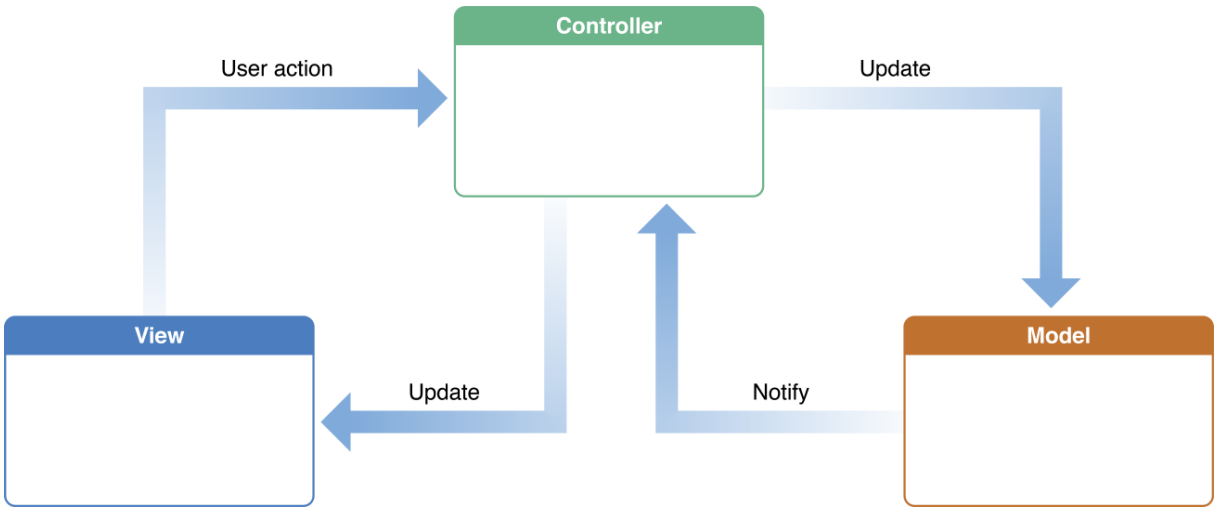

Figura 6: Padrão *Model-View-Controller*. Fonte: Apple Documentation.

4.1.1 Ambiente de desenvolvimento

Nesta seção é apresentado todo o ambiente de desenvolvimento do *app* Mini Farma quando foi feito originalmente para a plataforma iOS. O projeto foi executado contando com uma equipe de dois integrantes com conhecimentos intermediários na plataforma iOS. Os detalhes técnicos são listados a seguir.

- **Máquinas**: Dois MacBook Pro Retina 13", processador *Intel Core i5*, 8 GB de *RAM*;
- **Sistema Operacional das máquinas**: *Mac OS X Yosemite* v10.10.5;
- **Sistema Operacional alvo do aplicativo**: iOS 8;
- ***IDE***: *Xcode* v6.4;

- **Linguagem de Programação**: *Swift* v1.3;
- **Banco de dados**: SQLite v3.8.10.2;
- **Ambiente de suporte**: *Plugin SQLite Manager* para *Mozilla Firefox*, para criação do banco de dados;

4.2 Desenvolvimento multiplataforma do projeto

A partir do aplicativo feito em iOS, foi construída uma versão utilizando o Ionic para as plataformas Android e iOS, disponível em um repositório aberto no Github[2]. O desenvolvimento foi dividido nas seguintes funcionalidades:

- **Listagem de Remédios e Alertas**;
- **Cadastro de Remédios**: sendo necessário o acesso a câmera fotográfica e galeria de fotos do dispositivo;
- **Cadastro de Farmácias**: sendo necessário o acesso à localização do dispositivo e ligação de voz;
- **Cadastro de Alertas**: sendo necessário o acesso à central de notificações locais do dispositivo;

Para a criação dessa versão do *app*, a equipe foi alterada, mas permaneceu com dois integrantes, que são os autores deste trabalho. No entanto, os integrantes não possuíam qualquer conhecimento prévio nos *frameworks* Ionic, AngularJS e Cordova e apenas possuíam um conhecimento básico em *HTML*, *CSS* e *JavaScript*. Os detalhes técnicos são listados a seguir.

- **Máquinas**: Dois MacBook Pro Retina 13", processador *Intel Core i5*, 8 GB de *RAM*;
- **Sistema Operacional das máquinas**: *Mac OS X El Capitan* v10.11.5;
- **Sistema Operacional alvo do aplicativo**: iOS 9.3 e Android 6.0.0;
- ***IDE***: *WebStorm* v2016.1.1;
- ***Frameworks***:
  - **Ionic**: *Copenhagen* v1.2.4[3];

[2] <https://github.com/crossdev-tcc/ionic-apps>

[3] Durante a execução deste trabalho, já haviam sido disponibilizadas as versões 2 do Ionic e do AngularJS, no entanto, os autores optaram por não utilizá-las por ainda se tratarem de versões *beta* e portanto instáveis, o que poderia impactar negativamente no desenvolvimento do projeto.

  - **AngularJS**: *foam-acceleration* v1.4.3[3];
  - **Cordova**: v6.0.0;

- **Linguagem de Programação**: *JavaScript* v1.7.
  - *HTML* 5 e *CSS* 3 foram usados paralelamente para a criação da interface gráfica do aplicativo;

- **Banco de dados**: SQLite v3.8.10.2;

- **Ambiente de suporte**: Google Chrome, para *debug* do aplicativo;

- **Simuladores**:
  - **iOS**: iPhone 6S Plus
  - **Android**: Nexus 7

4.2.1 Relato de desenvolvimento

É apresentado, a seguir, o relato das experiências obtidas durante o desenvolvimento com Ionic, dividido nas funcionalidades já descritas na Subseção 3.2.3 deste trabalho, bem como feita a comparação entre o desenvolvimento nativo para iOS e o multiplataforma Ionic. Há, ainda, um tópico de observações gerais pertinentes à implementação do *app* como um todo.

- **Listagem de Remédios e Alertas**;
  - Para a tela inicial do *app*, a lista de remédios e alertas, foi preciso realizar uma pequena alteração em relação ao original, para considerar, agora, dois sistemas operacionais diferentes (iOS e Android). Foi preciso criar duas *views* para a mesma tela, pois o componente de abas (conhecido como *tabs*), é diferente no iOS e no Android. Com isso, quando foi feito pensando-se no iOS do Ionic, ele não correspondia muito bem ao padrão de interface do Android, por isso foi preciso criar duas telas separadas e fazer uma verificação se o dispositivo é iOS ou Android. Após feita a verificação de dispositivo, o próprio Ionic se encarrega de definir a aparência do componente para o padrão do sistema no qual o aplicativo está sendo executado.
  - Para a criação da lista de remédios e alertas que o usuário possui cadastrados, o componente usado no iOS é chamado de *UITableView*. No Ionic, o componente similar é o conjunto das diretivas *ion-list* e *ion-item* do Ionic com a diretiva *ng-repeat* do AngularJS. É muito simples de ser implementada, possuindo documentação ampla e exemplos de uso nos *sites* oficiais dos *frameworks*.

– Para a realização das filtragens dos remédios por data de validade, e dos alertas por ativos e inativos, bastou usar o componente *filter* padrão do AngularJS. Dessa forma, foi possível mostrar apenas as informações que deveriam ser mostradas. Novamente, possuindo muitos exemplos e ajuda na documentação do *framework*.

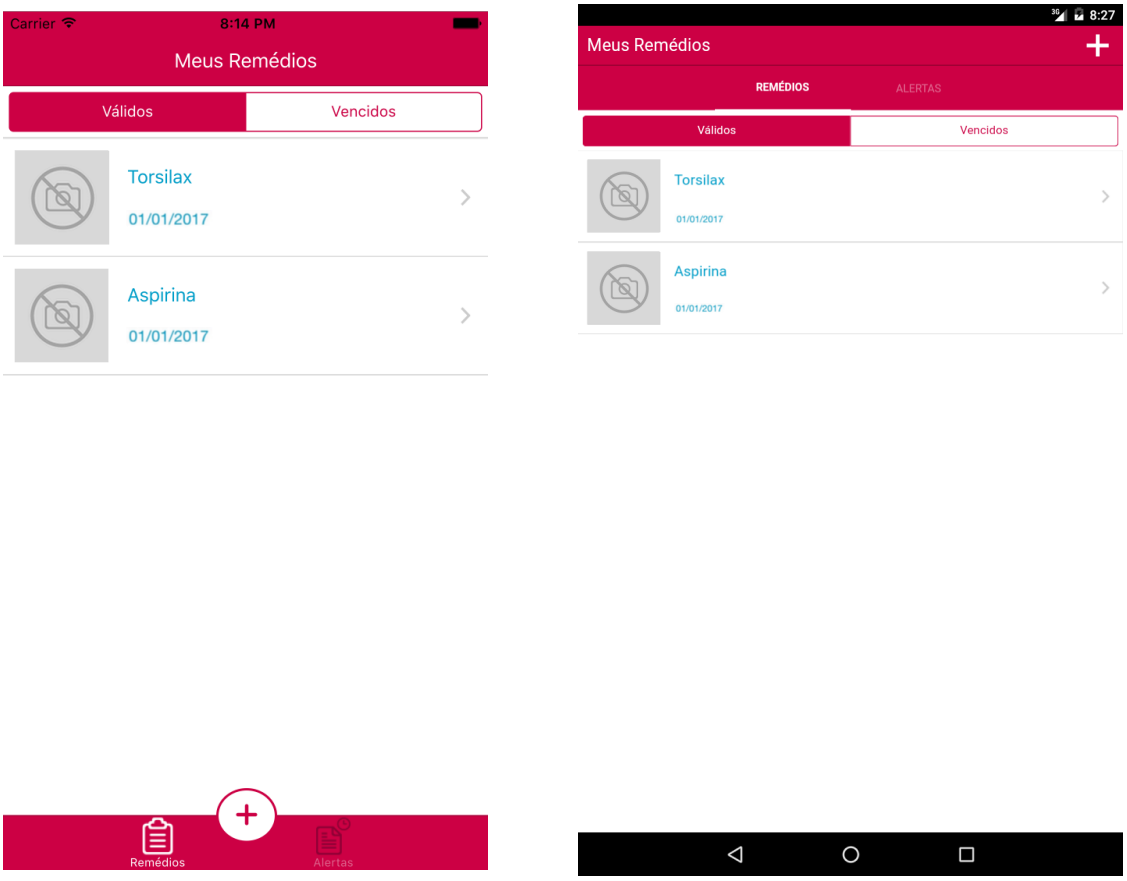

Figura 7: Telas da lista de remédios (iOS *versus* Android).

- **Cadastro de Remédios**;

  – No cadastro de remédios há o uso da câmera do dispositivo para tirar uma foto do medicamento e para registrar a prescrição médica. Opcionalmente, pode-se selecionar as fotos a partir da biblioteca de imagens do dispositivo. Com o uso do *plugin* Cordova de acesso à câmera, essa foi uma tarefa tranquila e que funcionou corretamente em ambas as plataformas, Android e iOS.

  – Para que o usuário opte entre tirar uma nova foto e escolher da galeria do dispositivo, foi utilizado um componente conhecido no iOS como *UIActionSheet*. Com o uso de um único código, o Ionic possibilitou a geração de um componente exatamente igual ao nativo do iOS e de um correspondente no Android. Na Figura 8 é possível observar as diferenças de interface, as quais foram adequadas pelo Ionic para cada plataforma.

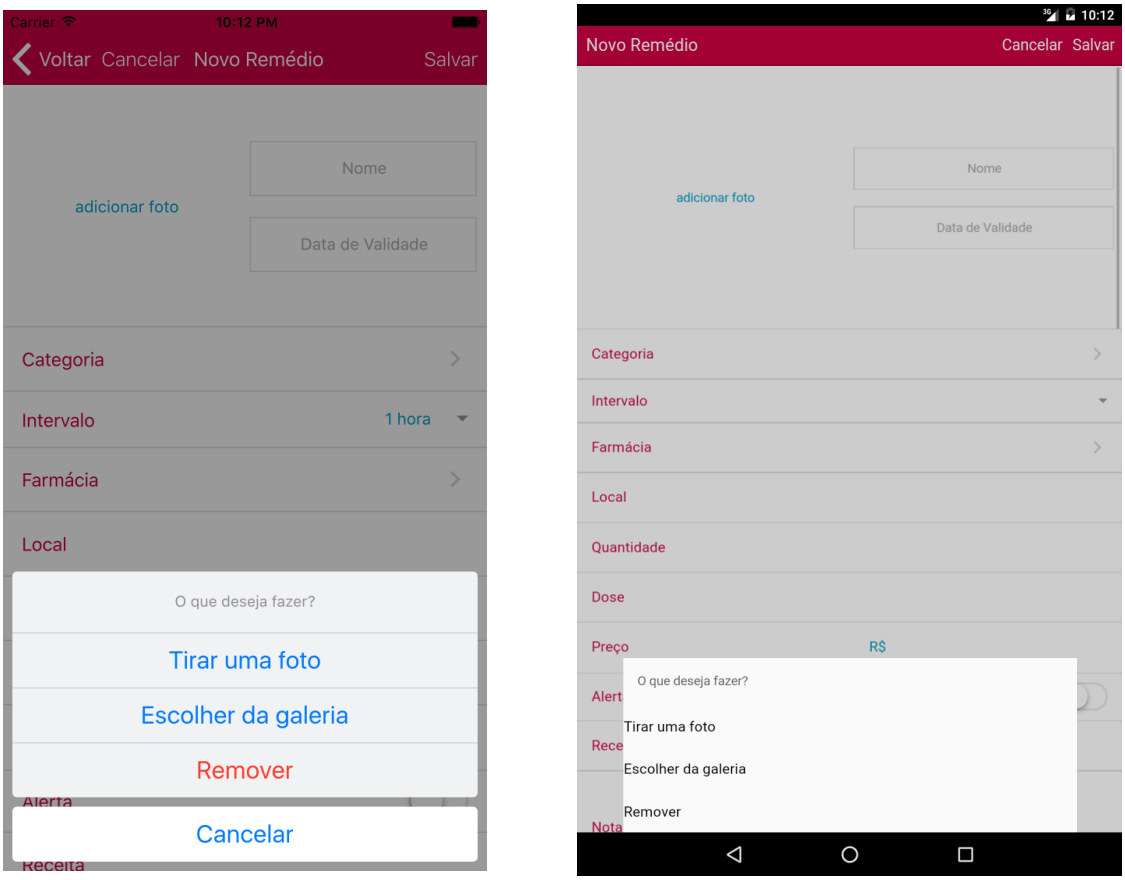

Figura 8: Telas de cadastro de foto de remédio (iOS *versus* Android).

– Outro componente utilizado e propriamente adaptado pelo Ionic para as plataformas foi o denominado no iOS como *UIPickerView*. Ele foi utilizado para a escolha de intervalo de uso do medicamento e é apresentado na Figura 9.

– No formulário de cadastro há a opção de selecionar uma categoria para o medicamento, para isso, o usuário é direcionado à uma página que contém a lista de categorias disponíveis, seleciona alguma delas ou adiciona uma nova e é redirecionado à página de cadastro, com o campo de categoria preenchido com a escolha feita. Para isso é necessário o envio de dados de uma tela para a antecessora. No aplicativo nativo essa funcionalidade foi implementada com o uso dos chamados *Protocols and Delegates*. Já no aplicativo multiplataforma não foi encontrada uma correspondência aos protocolos do iOS, foi, então, utilizada uma *factory* para armazenar temporariamente o valor da categoria. Observou-se que o nível de dificuldade de ambas as abordagens é semelhante, não apresentando grande dificuldade de implementação.

– Para a criação de uma nova categoria de remédios, foi utilizada interação por meio de uma caixa de diálogo, também conhecido como alerta. Para isso, utilizou-se os chamados *popups* do Ionic. Apesar de facilmente implementados e personalizáveis, esses componentes não se assemelham às caixas de diálogo

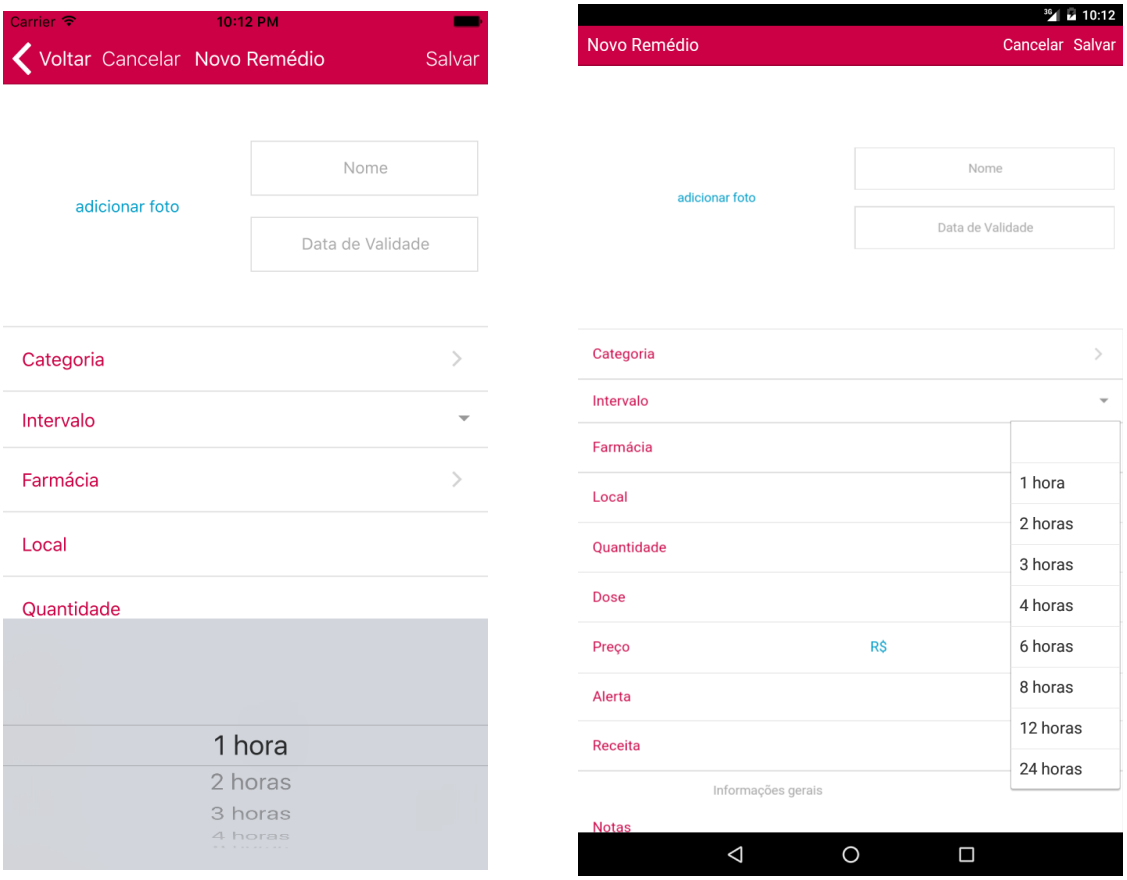

Figura 9: Telas de cadastro de foto de remédio (iOS *versus* Android).

nativas do iOS, *UIAlertView* e do Android, *AlertDialog*. O alerta criado é apresentado na Figura 10. Posteriormente, foi verificado que há no Cordova uma *API* para caixas de diálogo que, diferente da disponibilizada pelo Ionic, gera uma interface adequada aos padrões de ambas as plataformas. Um exemplo de alerta gerado pelo Cordova e idêntico à uma das plataformas, neste caso o iOS, é apresentado na Figura 11.

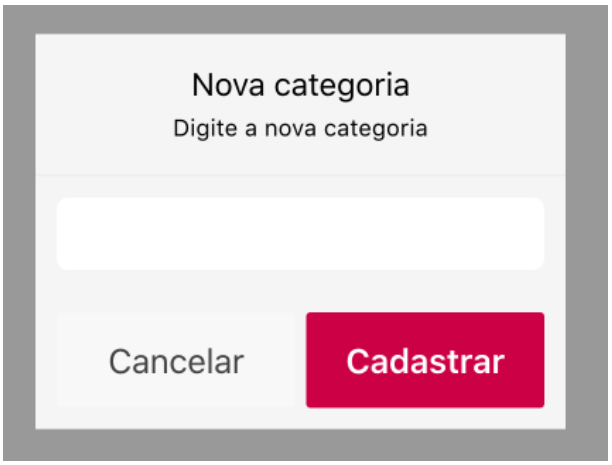

Figura 10: Tela de alerta.

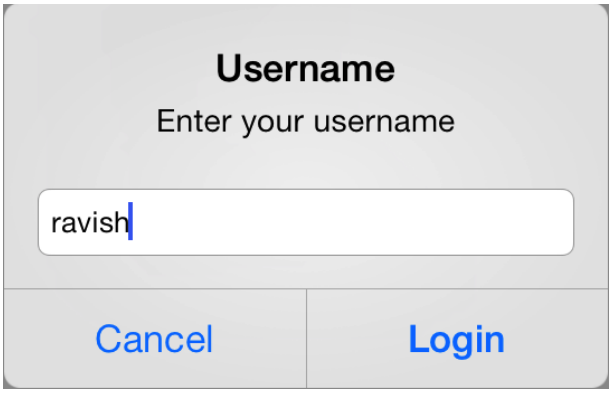

Figura 11: Alerta Cordova - iOS. Fonte: (FRAMEWORK, 2016).

– Para a seleção do local de armazenamento do remédio foi utilizado um componente inexistente no iOS, que é uma caixa de seleção. O resultado não agradou muito por ter ficado com aspecto de componente *web* e por não ter lembrado nenhum dos componentes nativos do iOS. Já para a plataforma Android o resultado foi satisfatório. O componente é apresentado na Figura 12.

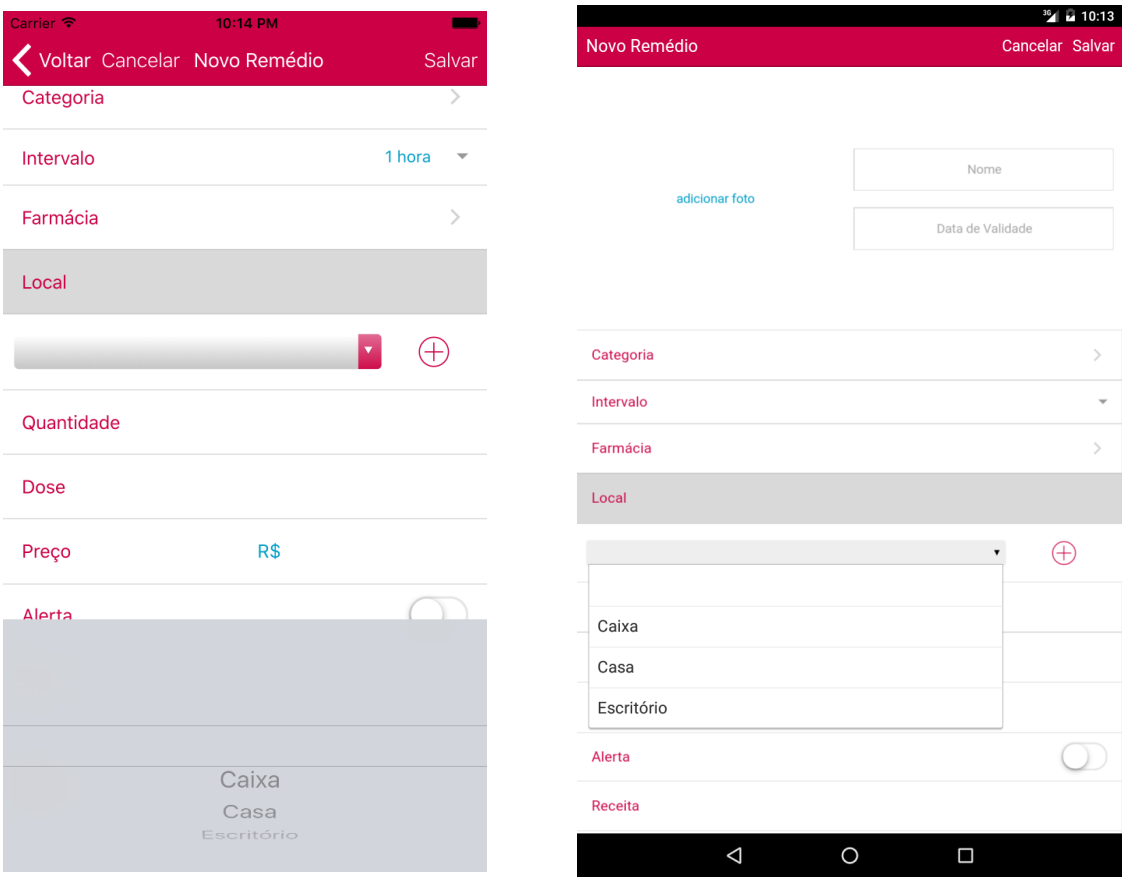

Figura 12: Tela de cadastro de local do remédio (iOS *versus* Ionic).

– Em relação ao banco de dados, utilizou-se a *API* Cordova SQLite. Como o banco de dados construído nativamente também foi feito utilizando o SQLite,

foi possível aproveitar os *scripts* de definição e manipulação de dados. Não houveram complicações no uso da *API* e facilmente foram encontrados exemplos de uso na internet que auxiliaram na estruturação da camada de comunicação do aplicativo com o banco de dados.

– De forma geral, o resultado da tela de cadastro de remédios foi satisfatório e a interface construída ficou semelhante à tela do aplicativo nativo, apresentado na Figura 13. A tela construída com Ionic é apresentada nas Figuras 8 e 12.

Figura 13: Tela de cadastro de remédio - iOS.

- **Cadastro de Farmácias**;

  – Para cadastrar uma nova farmácia, o usuário deve marcar no mapa onde a farmácia se encontra. Para a utilização do mapa no Ionic, foi utilizada a *API Maps* do Google, diferentemente da *API MapKit* do iOS. Não houveram grandes dificuldades na criação do mapa e renderização do mesmo na tela. A dificuldade de implementação é similar a do iOS nativo. Apenas realizando pesquisas e estudos na documentação da *API* do Google, já foi possível a elaboração do que precisava ser feito, no caso, mostrar o mapa e fornecer a opção de mover o pino para um local específico e capturar as coordenadas daquele ponto. No caso, foi necessário utilizar a diretiva *map* para renderização do mapa na tela

e definir o *callback* de reposicionamento do cursor e captura da localização. Na Figura 14 é possível ver as diferenças entre o mapa do *MapKit* do iOS e da *API Maps* do Google.

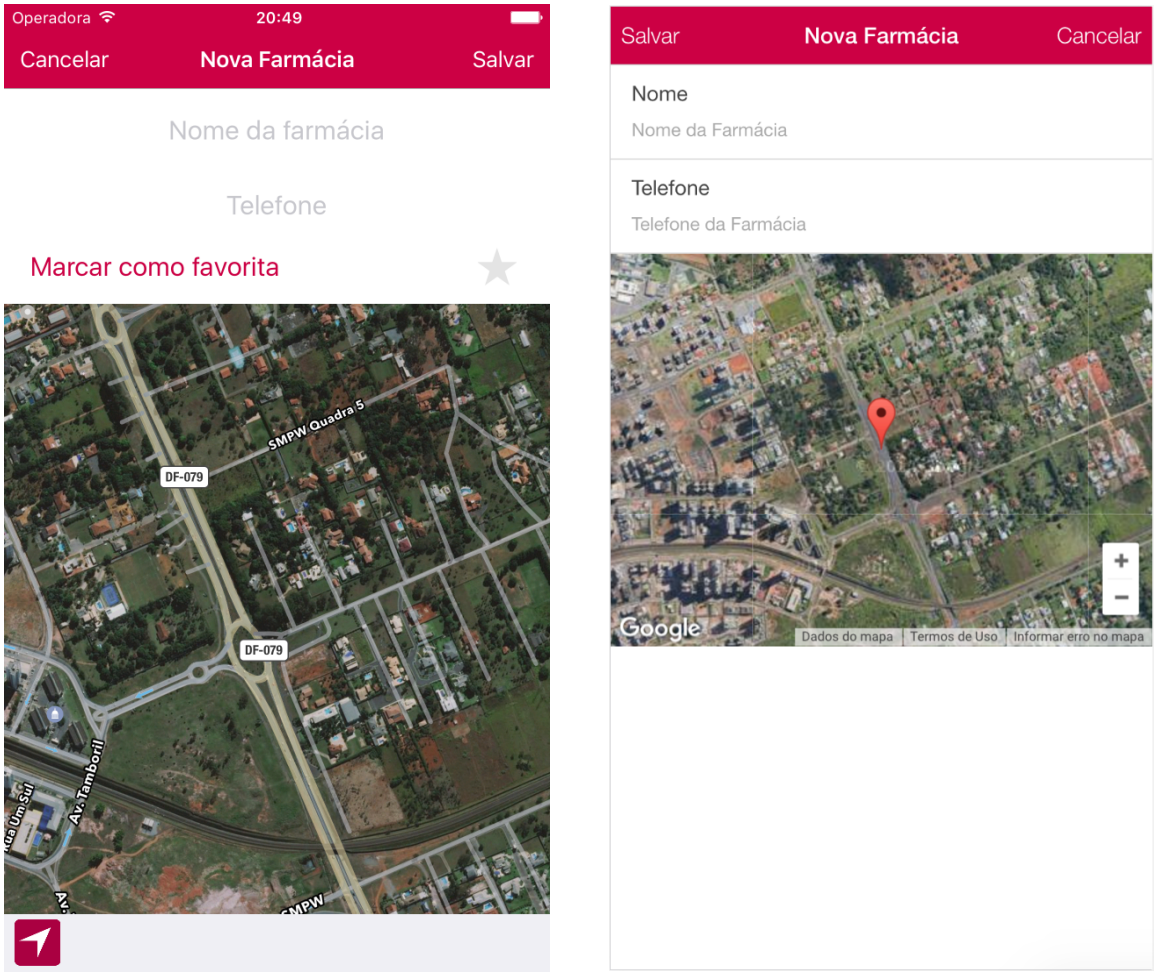

Figura 14: Tela de adicionar farmácia (iOS *versus* Ionic).

- **Cadastro de Alertas**;
  - Para o cadastro de um novo alerta, é preciso selecionar data e hora que o usuário será alertado. Para isso, o Ionic 1 não possui o componente adequado. Com isso, foi necessário procurar e utilizar um *framework* terceiro. Os *frameworks* escolhidos foram o *ionic-datepicker*[4] e *ionic-timepicker*[5], respectivamente para seleção de data e hora. Ambos foram desenvolvidos pelo mesmo usuário no Github e instalados no projeto do aplicativo via *CLI*. Nas próprias documentações dos *frameworks* existem exemplos de uso suficientes para guiar a implementação das funcionalidades. Vale ressaltar que o componente do iOS possui a seleção de data e hora juntas, enquanto nos *frameworks* utilizados, a seleção

[4] <https://github.com/rajeshwarpatlolla/ionic-datepicker>
[5] <https://github.com/rajeshwarpatlolla/ionic-timepicker>

é separada, gerando uma pequena alteração na interface gráfica da tela de cadastro de alertas quando comparada com a do aplicativo original. As diferenças entre o original e Ionic podem ser vistas na Figura 15. O mesmo *framework* para seleção de data também foi utilizado na tela de cadastro de remédios para definir a data de validade do mesmo.

– Para a criação dos alertas é preciso cadastrar notificações locais na central de notificações do dispositivo. Realizar essa tarefa teve o mesmo nível de dificuldade da implementação no iOS nativo. Apenas foi instalado um *plugin* do Cordova para gerenciar a central de notificações e agendar notificações com as datas e horas corretas para serem entregues.

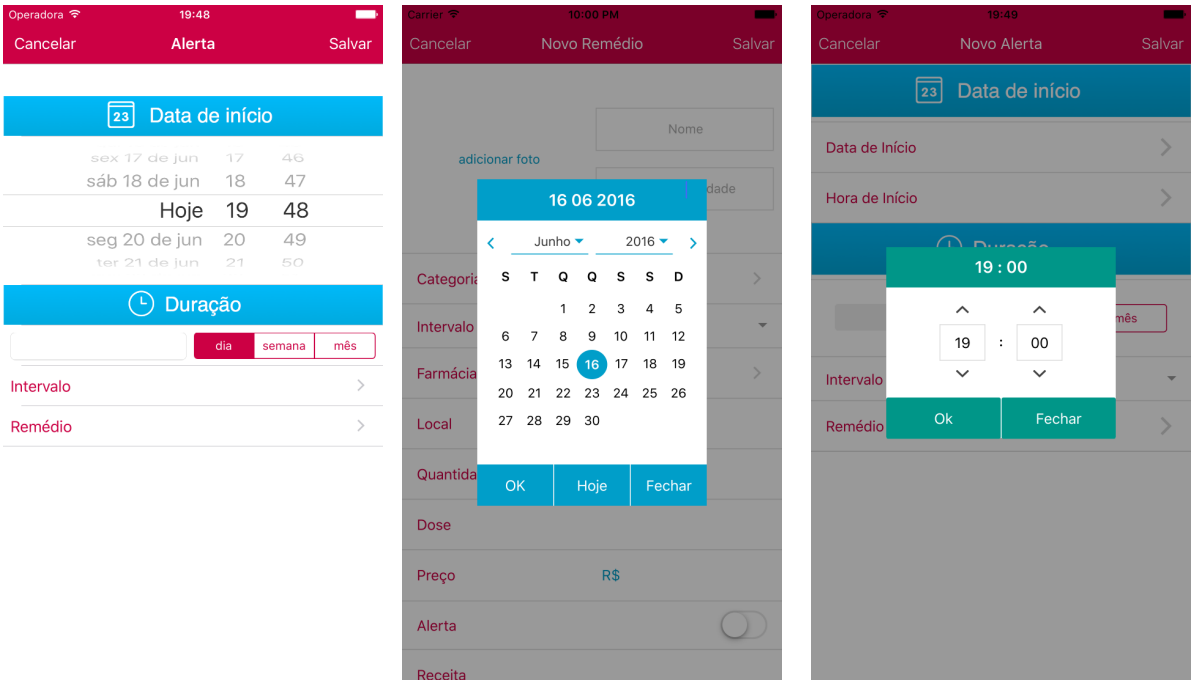

Figura 15: Seleção de data e hora dos alertas (iOS *versus* Ionic).

- **Observações gerais**;

  – Houve um problema no carregamento da lista inicial de remédios. Por ficar na primeira tela do aplicativo, acontecia de o acesso ao banco de dados para obtenção da lista ocorrer antes do total carregamento do Ionic, o que acarretava em falha na execução do aplicativo. Esse problema não existiu na criação do *app* original e para contorná-lo foram feitas pesquisas na comunidade para saber como resolver. Umas das soluções sugeridas era causar um *delay* proposital no aplicativo para que desse tempo do Ionic carregar completamente, mas foi descartada para não impactar negativamente na performance do *app*. A solução utilizada foi fazer uma verificação a cada transação realizada no banco

de dados para que só sejam efetuadas caso o Ionic esteja pronto para ser usado e apresentado.

- O Ionic apresenta uma útil funcionalidade que permite o *debug* dos aplicativos pelo terminal, porém, a partir da versão 9 no iOS não é possível realizar essa ação, enquanto que no Android continua funcionando adequadamente. Para corrigir o problema, usuários do Ionic recomendam a modificação de uma configuração no arquivo *.plist* do projeto iOS, para desabilitar a opção *App Transport Security* e permitir requisições que não sejam *HTTPS*. No entanto, ao modificar essa opção há o risco do aplicativo ser recusado no momento da sua publicação, portanto é necessário lembrar de reabilitá-la antes de submeter para a loja de aplicativos. Caso opte-se por não realizar essa alteração, para debugar é preciso um computador com *Xcode* ou é possível também utilizar os navegadores como *Google Chrome* ou *Safari* no modo desenvolvedor para debugar utilizando o console *JavaScript* dos navegadores.
- Alguns erros do *JavaScript* não são acusados pelo terminal do Ionic, dificultando a tarefa de *debug* da aplicação. Com isso, em um momento de dificuldade em achar um problema que estava acontecendo, algumas pesquisas foram realizadas na comunidade *on-line* e em um fórum de discussão sobre Ionic foi dada uma dica sobre o uso de alertas do *JavaScript* para mostrar erros para o desenvolvedor e com isso facilitar na tarefa de debugar o aplicativo.
- A diretiva *ng-class* do AngularJS não funciona em conjunto com a diretiva *ion-nav* do Ionic. No entanto, isso não foi achado nas documentações, nem do Ionic e nem do AngularJS. Novamente, recorremos à comunidade e foi dito que esses componentes não funcionam corretamente juntos.
- Por mais que os autores não possuíssem conhecimentos avançados em *HTML*, *CSS* e *JavaScript*, vale ressaltar que o conhecimento inicial nessas tecnologias ajudou muito durante o desenvolvimento do *app*.
- O Cordova e o Ionic possuem documentações bem completas, apresentando exemplos de uso de códigos. A comunidade é ativa e disponibiliza *APIs* adicionais e exemplos de códigos em repositórios online.

Apesar das desvantagens citadas na Tabela 1, não houveram limitações quanto ao uso dos recursos necessários para o projeto ou problemas de performance que impactassem nos requisitos do aplicativo. Foi possível construir apenas um código e gerar dois executáveis de plataformas diferentes, o que poderia acarretar em redução de custos e tempo de desenvolvimento. Além disso o aplicativo desenvolvido apresentou aparência e usabilidade semelhantes ao nativo, adaptando-se a cada plataforma alvo seguindo seus padrões específicos de interface de usuário.

5 Considerações Preliminares

Ao longo da execução do trabalho, foi possível perceber que as ferramentas para desenvolvimento multiplataforma evoluíram muito desde sua criação, o que as tornaram, hoje, uma opção que deve ser considerada no momento da criação de um novo *app*.

Ambas as abordagens de desenvolvimento móvel possuem suas vantagens e desvantagens, conforme apresentado nos capítulos anteriores deste trabalho. No entanto, antigamente as ferramentas *cross* apresentavam um *gap* muito grande quando comparadas às ferramentas e ambientes nativos e com isso, dificilmente eram consideradas no momento do desenvolvimento de um aplicativo.

Todas as funcionalidades do aplicativo Mini Farma, que foram planejadas para serem desenvolvidas no ambiente *cross-plataform*, puderam ser desenvolvidas, não havendo quaisquer limitações quanto ao uso dos recursos nativos do dispositivo necessários para o projeto selecionado. O *app* multiplataforma se assemelhou muito ao nativo em relação a aparência e usabilidade o que confirma a ideia de que as ferramentas multiplataforma estão cada vez mais se aproximando das nativas apresentando, com o passar do tempo, mais vantagens do que desvantagens, mostrando ainda, que os gargalos antes vistos para essa forma de desenvolvimento, não condizem mais com a realidade.

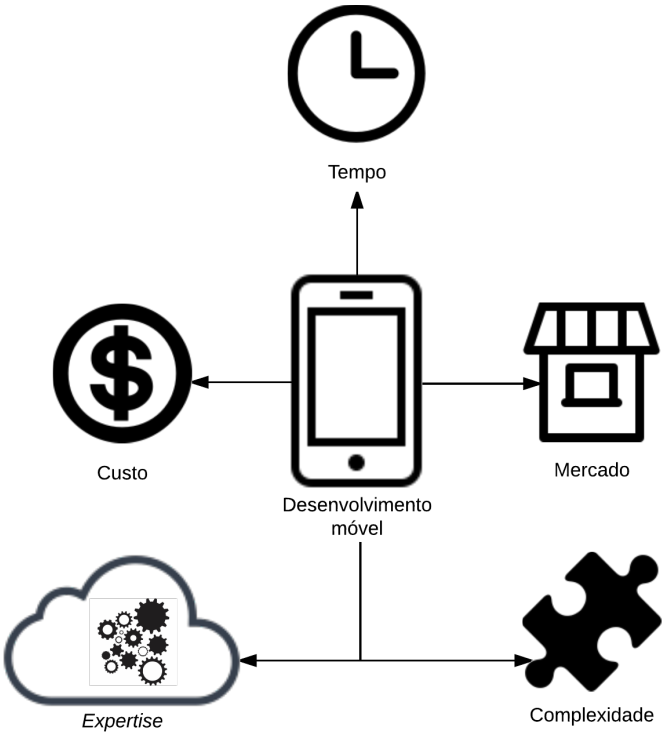

Figura 16: Fatores a serem avaliados no desenvolvimento móvel.

Com o término da primeira parte deste trabalho, pôde-se concluir que o desenvolvimento móvel requer uma análise aprofundada de uma série de fatores, como mercado, público e tecnologias para decidir qual abordagem escolher. É importante ressaltar que a abordagem nativa não é melhor que a *cross-platform* ou vice-versa, sendo apenas distinta e deve-se avaliar qual utilizar caso a caso. Para cada situação existem fatores que devem ser avaliados de uma maneira conjunta e alguns desses fatores são listados e explicados a seguir e apresentados na Figura 16.

- **Tipo e complexidade da aplicação**: cada aplicação possui requisitos diferentes e próprios que originam necessidades e dificuldades inerentes daquele aplicativo. Com isso, deve-se avaliar, com base nos requisitos da aplicação, qual abordagem suporta melhor o *app*;

- ***Expertise* da equipe nas plataformas e seus ambientes**: cada equipe possui um conjunto único de habilidades e conhecimentos. No momento da escolha de uma abordagem, esses conhecimentos devem ser levados em consideração, visto que é a equipe de desenvolvimento que irá conceber o produto final. Se a equipe possui mais conhecimentos em uma abordagem do que em outra, isso pode ser um indicativo de qual abordagem escolher;

- **Nicho de mercado que se quer atacar**: Cada plataforma móvel (iOS, Android, *Windows Phone*, etc), domina uma parcela do mercado e possui um grupo de usuários com características, opiniões, necessidades e gostos próprios inerentes à plataforma que usam. Com isso, no momento de criar um *app* deve-se pensar para quem é esse aplicativo. Se ele for concebido para suprir uma demanda de um grupo específico, talvez não haja a necessidade de criá-lo para várias plataformas;

- **Prazo de desenvolvimento**: Quanto mais plataformas para atender, maior é o tempo necessário para desenvolver a solução. Se o prazo do projeto for apertado para desenvolvimento de mais de uma solução nativa, há de considerar o desenvolvimento multiplataforma, visto que apenas será codificada uma solução que poderá atender várias plataformas diferentes;

- **Capital disponível para investimento**: Desenvolver para plataformas nativas exige ambiente, infraestrutura e conhecimentos diferentes para cada plataforma. Dessa forma, quanto mais plataformas se quer abarcar, mais custoso o projeto será. Uma solução multiplataforma pode ser mais viável economicamente dependendo da situação;

5.1 Planos Futuros

Na continuação deste trabalho, serão realizadas novas análises de exemplos de uso a fim de obter mais detalhes sobre as vantagens e desvantagens das abordagens de desenvolvimento móvel. A seguir, é apresentado, na Tabela 2, um cronograma preliminar das atividades a serem realizadas.

Atividade	**Julho**	**Agosto**	**Setembro**	**Outubro**	**Novembro**	**Dezembro**
Recriar um aplicativo nativo Android em Ionic	X	X				
Propor um fluxograma de tomada de decisão		X	X			
Investigar relação com linha de produto de *software*		X	X			
Avaliar fluxograma proposto em um projeto Ionic			X	X	X	X
Refinar análise de vantagens e desvantagens e do fluxograma					X	X

Tabela 2: Cronograma inicial para o TCC 2

Será recriado um aplicativo, originalmente feito para a plataforma Android, utilizando o *framework* Ionic a fim de aprimorar o comparativo entre o desenvolvimento nativo e multiplataforma e colher mais insumos para a proposta de um fluxograma de tomada de decisão de qual abordagem é mais adequada para um dado contexto de desenvolvimento.

Uma vez feito o novo aplicativo, será proposto um fluxograma para auxiliar desenvolvedores a escolher de forma mais assertiva qual abordagem utilizar no contexto de desenvolvimento que estiver imerso.

Realizar pesquisas na área de linha de produto de *software*, para investigar se há alguma relação com o desenvolvimento de aplicativos móveis.

A fim de validar o fluxograma proposto, o mesmo será utilizado em um projeto de evolução de um aplicativo Ionic para avaliar se o modelo de tomada de decisão está correto ou precisa de melhorias.

Ao final serão refinados a análise de vantagens e desvantagens das abordagens de desenvolvimento móvel e o fluxograma criado.

Referências

Android. *Android Interfaces and Architecture | Android Open Source Project*. 2016. Disponível em: <http://source.android.com/devices/index.html>. Citado 2 vezes nas páginas 27 e 28.

Android. *Meet Android Studio | Android Studio*. 2016. Disponível em: <https://developer.android.com/studio/intro/index.html>. Citado na página 28.

Apple Inc. News, *Apple - Hot News*. 2007. Disponível em: <http://web.archive.org/web/20071020040652/http://www.apple.com/hotnews>. Citado na página 24.

Apple Inc. *About Objective-C*. 2014. Disponível em: <https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>. Citado na página 26.

Apple Inc. *About the iOS Technologies*. 2014. Disponível em: <https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>. Citado 2 vezes nas páginas 24 e 25.

Apple Inc. *Apple Developer Program - Apple Developer*. 2016. Disponível em: <https://developer.apple.com/programs/>. Citado na página 26.

Apple Inc. *Developing for iOS 9 - Apple Developer*. 2016. Disponível em: <https://developer.apple.com/ios/>. Citado na página 24.

Apple Inc. *Submitting Your App to the Store*. 2016. Disponível em: <https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/SubmittingYourApp/SubmittingYourApp.html>. Citado na página 26.

Apple Inc. *Swift - Apple (BR)*. 2016. Disponível em: <http://www.apple.com/br/swift/>. Citado na página 26.

BARTH, N. *ANÁLISE COMPARATIVA DE FERRAMENTAS DE DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA*. UNIVERSIDADE REGIONAL DE BLUMENAU, 2014. Disponível em: <http://dsc.inf.furb.br/arquivos/tccs/monografias/2014_1_nikson-barth_monografia.pdf>. Citado na página 34.

BEZERRA, P. T.; SCHIMIGUEL, J. *Desenvolvimento de aplicaçõeses mobile cross-platform utilizando phonegap*. 2016. Disponível em: <http://eumed.net/cursecon/ecolat/br/16/phonegap.html>. Citado 3 vezes nas páginas 29, 30 e 38.

CEVALLOS, E. A. *Case Study on Mobile Applications UX: Effect of the Usage of a Cross Platform Development Framework*. Tese (Master Thesis) — UNIVERSIDAD POLITÉCNICA DE MADRID, Madrid, jun. 2014. Disponível em: <http://oa.upm.es/30422/1/EMSE-2014-05_Esteban_Angulo-1.pdf>. Citado na página 19.

CHARLAND, A.; LEROUX, B. Mobile Application Development: Web vs. Native. *Commun. ACM*, v. 54, n. 5, p. 49–53, 2011. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1941487.1941504>. Citado 2 vezes nas páginas 34 e 38.

CORRAL, L.; JANES, A.; REMENCIUS, T. Potential Advantages and Disadvantages of Multiplatform Development Frameworks - A Vision on Mobile Environments. *Procedia Computer Science*, v. 10, p. 1202–1207, jan. 2012. ISSN 1877-0509. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050912005303>. Citado 5 vezes nas páginas 19, 23, 33, 34 e 38.

DRIFTY. *About Ionic - Ionic Components*. 2016. Disponível em: <http://ionicframework.com/docs/overview/#about>. Citado na página 30.

DRIFTY. *Installing Ionic and its Dependencies - Ionic Framework*. 2016. Disponível em: <http://ionicframework.com/docs/guide/installation.html>. Citado na página 32.

DRIFTY. *Ionic: Advanced HTML5 Hybrid Mobile App Framework*. 2016. Disponível em: <http://ionicframework.com/>. Citado 2 vezes nas páginas 30 e 32.

DRIFTY. *Ionic Concepts - App Structure - Ionic Framework*. 2016. Disponível em: <http://ionicframework.com/docs/concepts/structure.html>. Citado na página 31.

DRIFTY. *Ionic Creator*. 2016. Disponível em: <http://ionic.io/products/creator>. Citado na página 32.

DRIFTY. *Ionic Lab*. 2016. Disponível em: <http://lab.ionic.io/>. Citado na página 32.

DRIFTY. *Ionic Platform*. 2016. Disponível em: <http://ionic.io/platform>. Citado na página 32.

DRIFTY. *Ionic Play*. 2016. Disponível em: <http://play.ionic.io>. Citado na página 33.

DRIFTY. *The Ionic View App | Share your apps with the world*. 2016. Disponível em: <http://view.ionic.io/>. Citado na página 33.

EL-KASSAS, W. S. et al. Taxonomy of Cross-Platform Mobile Applications Development Approaches. *Ain Shams Engineering Journal*, 2015. ISSN 2090-4479. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2090447915001276>. Citado 5 vezes nas páginas 23, 24, 28, 33 e 34.

FRAMEWORK, I. *ngCordova - Document and Examples*. 2016. Disponível em: <http://ngcordova.com/docs/plugins/dialogs/>. Citado na página 49.

GOOGLE. *AngularJS - Superheroic JavaScript MVW Framework*. 2016. Disponível em: <https://angularjs.org/>. Citado na página 30.

HEITKOTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. A. Evaluating cross-platform development approaches for mobile applications. In: *Web information systems and technologies*. Springer, 2013. p. 120–138. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-642-36608-6_8>. Citado 4 vezes nas páginas 24, 26, 28 e 29.

HEITKöTTER, H.; HANSCHKE, S.; MAJCHRZAK, T. A. Comparing Cross-platform Development Approaches for Mobile Applications. In: *Web information systems and technologies*. Springer, 2013. p. 120–138. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-642-36608-6_8>. Citado na página 19.

HOLZINGER, A.; TREITLER, P.; SLANY, W. Making Apps Useable on Multiple Different Mobile Platforms: On Interoperability for Business Application Development on Smartphones. In: QUIRCHMAYR, G. et al. (Ed.). *Multidisciplinary Research and Practice for Information Systems*. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, 7465). p. 176–189. ISBN 978-3-642-32497-0 978-3-642-32498-7. DOI: 10.1007/978-3-642-32498-7_14. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-642-32498-7_14>. Citado 2 vezes nas páginas 33 e 34.

JOBE, W. Native Apps Vs. Mobile Web Apps. *International Journal of Interactive Mobile Technologies (iJIM)*, v. 7, n. 4, p. 27, out. 2013. ISSN 1865-7923. Disponível em: <http://online-journals.org/i-jim/article/view/3226>. Citado na página 24.

MEIER, R. *Creating Better User Experiences on Google Play | Android Developers Blog*. 2015. Disponível em: <http://android-developers.blogspot.com.br/2015/03/creating-better-user-experiences-on.html>. Citado na página 28.

PAPAJORGJI, P. *Automated Enterprise Systems for Maximizing Business Performance*. 1 edition. ed. Hershey, PA: IGI Global, 2015. ISBN 978-1-4666-8841-4. Citado na página 26.

PREZOTTO, E.; BONIATI, B. Estudo de Frameworks Multiplataforma Para Desenvolvimento de Aplicações Mobile Híbridas. 2014. Citado 2 vezes nas páginas 19 e 38.

REBOUAS, M. et al. An Empirical Study on the Usage of the Swift Programming Language. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. [S.l.: s.n.], 2016. v. 1, p. 634–638. Citado na página 26.

RODRíGUEZ, A. M.; BALDRICH, R. Diseño e implementación de una aplicación multidispositivo en un entorno HTML5. 2015. OCLC: 878531227. Citado na página 30.

SHAKSHUKI, E. M. et al. Component based Framework to Create Mobile Cross-platform Applications. *Procedia Computer Science*, v. 19, p. 1004–1011, jan. 2013. ISSN 1877-0509. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050913007485>. Citado na página 33.

STARK, J. *Building iPhone Apps with HTML, CSS, and JavaScript*. O'Reilly Media, 2010. ISBN 978-0-596-80578-4. Disponível em: <http://shop.oreilly.com/product/9780596805791.do>. Citado 3 vezes nas páginas 19, 28 e 29.

URSINO, D.; CAPANNA, C. A. AngularJS - un framework di frontiera per la realizzazione di siti Web. 2015. Disponível em: <http://www.barbiana20.unirc.it/wp-content/uploads/2015/07/Tesi-Cristiano-finale-2.pdf>. Citado na página 30.