



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO  
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA  
ESCOLA DE INFORMÁTICA APLICADA

Um estudo sobre o PhoneGap e seu desempenho ante a linguagem nativa do Android.

Leonardo Moreira de Souza  
Túlio Henrique Seixas Lemes

**Orientador**  
Alexandre Correa

RIO DE JANEIRO, RJ – BRASIL  
DEZEMBRO DE 2014

Um estudo sobre o PhoneGap e seu desempenho ante a linguagem nativa do Android.

Leonardo Moreira de Souza  
Túlio Henrique Seixas Lemes

Projeto de Graduação apresentado à Escola de  
Informática Aplicada da Universidade Federal do  
Estado do Rio de Janeiro (UNIRIO) para obtenção do  
título de Bacharel em Sistemas de Informação.

Aprovada por:

---

Alexandre Correa (UNIRIO)

---

[NOME DO PRIMEIRO INTEGRANTE DA BANCA]

---

[NOME DO SEGUNDO INTEGRANTE DA BANCA]

RIO DE JANEIRO, RJ – BRASIL.

DEZEMBRO DE 2014

## **Agradecimentos**

Agradeço aos meus familiares e namorada, pelo apoio e confiança ao longo desses quatro anos.

A UNIRIO, pela oportunidade.

Ao corpo docente, responsável pela excelência do ensino.

Aos amigos, pelas alegrias, realizações e pela caminhada compartilhada.

Túlio Lemes

Meu muito obrigado vai para todos que de alguma forma participaram dessa jornada de evolução, diversão e aprendizado.

Leonardo Souza

## RESUMO

Com a venda de smartphones batendo recordes de trimestre em trimestre, o acesso da população aos aplicativos e sites móveis se torna cada vez maior. Porém, entrar para este mercado não é tão simples, além das conhecidas barreiras na área de desenvolvimento, há diversas particularidades distintas de configuração, tanto do hardware quanto do software por parte dos fabricantes. Os *frameworks* multiplataforma surgem como uma alternativa de desenvolvimento de um único código que funcione em diversos sistemas operacionais. Com aproveitamento de código, custos reduzidos e uma facilidade maior de atualizações futuras, esses *frameworks* se tornam uma opção atraente para o desenvolvimento.

Este trabalho visa analisar a estrutura, o comportamento e o desempenho do PhoneGap, *framework* multiplataforma mais utilizado no sistema operacional com maior fatia de mercado, o Android. Também apresenta um comparativo com a linguagem Java (nativa do Android), mostrando as vantagens, perdas e o perfil de aplicação mais adequado para essa estrutura de *framework*.

**Palavras-chave:** PhoneGap, Android, Comparação, Framework, Desempenho.

## ABSTRACT

By selling smartphones hitting quarter records in the quarter, the population's access to applications and mobile sites becomes increasingly larger. However, entering this market is not so simple, in addition to the known barriers in the development area, there are several distinct peculiarities configuration, both the hardware and the software for manufacturers. The cross-platform *frameworks* emerge as an alternative to develop a unique code that runs on multiple operating systems. With code reuse, low cost and ease of future upgrades, these *frameworks* became an attractive option for development.

This work aims to analyze the structure, behavior and performance of PhoneGap, multiplatform *framework* most widely used operating system more market share, Android. Also presents a comparison with Java (Android native), showing the benefits, losses and the most appropriate application profile for this *framework* structure.

**Keywords:** PhoneGap, Android, Comparison, Framework, Performance

## Sumário

<b>1. Introdução</b> .....	11
1.1 Motivação .....	11
1.2 Objetivos.....	12
1.3 Organização do texto .....	12
<b>2. Soluções de desenvolvimento para plataformas móveis</b> .....	14
2.1 Abordagens Multiplataforma.....	14
2.1.1 Solução web .....	14
2.1.2 Solução nativa .....	15
2.2 Categorias de <i>frameworks</i> .....	16
2.2.1 App Factories .....	16
2.2.2 Cross-Platform Integrated Development Environments (CP IDEs) .....	16
2.2.3 Web App Toolkits .....	17
2.2.4 CP IDEs for Enterprise (CP IDEs Enterprise) .....	17
2.2.5 CP Compilers .....	18
2.2.6 CP Services .....	19
2.3 Mercado dos <i>frameworks</i> .....	20
2.4 PhoneGap.....	22
<b>3. PhoneGap</b> .....	24
3.1 Visão Geral de Funcionamento.....	24
3.2 Estrutura da aplicação .....	26
3.3 Configuração do PhoneGap.....	26
3.3.1 Compilador local .....	26
3.3.2 Compilador online .....	27
3.3.3 Configuração da IDE.....	28
3.4 APIs do PhoneGap.....	29
3.4.1 Acelerômetro.....	30

3.4.2	Câmera.....	32
3.4.3	Bússola .....	33
3.4.4	Contatos .....	33
3.4.5	Arquivos .....	34
3.4.6	Geolocalização.....	34
3.4.7	API do Google Maps no PhoneGap .....	35
3.4.8	Mídia.....	35
3.4.9	Notificações .....	36
3.4.10	Banco de Dados .....	36
3.4.10.1	Web Storage .....	37
3.4.10.2	Web SQL .....	37
3.4.10.3	Indexed Database (Banco de Dados Indexado).....	38
3.4.10.4	API de Arquivos .....	38
<b>4.</b>	<b>PhoneGap x Android .....</b>	<b>39</b>
4.1	A aplicação .....	39
4.2	Documentação .....	40
4.3	Curva de aprendizado .....	44
4.4	Performance .....	45
4.4.1	Performance do Acelerômetro.....	45
4.4.2	Performance da Bússola .....	46
4.4.3	Performance de Geolocalização .....	48
4.4.4	Performance de Rede.....	49
4.4.5	Performance de Processamento.....	49
4.5	Reutilização de código no desenvolvimento de aplicativo para outra plataforma	51
<b>5.</b>	<b>Conclusão .....</b>	<b>52</b>
5.1	Considerações finais .....	52
5.2	Limitações do projeto .....	53

5.3 Projetos futuros.....	53
---------------------------	----

## Índice de Figuras

Figura 1: Mercado global dos sistemas operacionais entre abril de 2011 e junho de 2014	11
Figura 2: Estrutura de funcionamento do Xamarin	15
Figura 3: Estrutura do AnyPresence	18
Figura 4: Estrutura do PhoneGap	19
Figura 5: Plataformas nas quais são publicadas aplicativos de CTP	20
Figura 6: Plataformas que os desenvolvedores mais sentem falta nos <i>frameworks</i> utilizados	21
Figura 7: Número de plataformas alcançadas por um aplicativo do <i>framework</i>	22
Figura 8: Os três <i>frameworks</i> preferidos para as sete maiores plataformas	23
Figura 9: Arquitetura de um aplicativo PhoneGap	25
Figura 10: Upload através do Adobe PhoneGap Build	27
Figura 11: Compilação através do Adobe PhoneGap Build	28
Figura 12: Interface Ripple	29
Figura 13: Relação de APIs disponíveis no PhoneGap, de acordo com funcionalidade e sistema operacional.	30
Figura 14: Comportamento dos eixos de um acelerômetro	31
Figura 15: Tela inicial aplicativo PhoneGap	40
Figura 16: Documentação Android	41
Figura 17: Tela inicial da guia de APIs	42
Figura 18: Documentação PhoneGap	43
Figura 19: Tela que explica o funcionamento de um plugin do PhoneGap	43
Figura 20: Página inicial da documentação do PhoneGap	44
Figura 21: Tempo de acesso ao Acelerômetro pelo aplicativo Android	46
Figura 22: Tempo de acesso ao Acelerômetro pelo aplicativo PhoneGap	46
Figura 23: Tempo de execução para retorno de informação da bússola no Android	47
Figura 24: Tempo de execução para retorno de informação da bússola no PhoneGap	47
Figura 25: Tempo de execução para retorno de dados de Geolocalização no Android	48
Figura 26: Tempo de execução para retorno de dados de Geolocalização no PhoneGap	48
Figura 27: Tempo de acesso as informações de rede no Android	49
Figura 28: Tempo de acesso as informações de rede no PhoneGap	49

Figura 29: Tempo de execução para achar o valor um bilhão da sequência de Fibonacci  
no Android 50

Figura 30: Tempo de execução para achar o valor um bilhão da sequência de Fibonacci  
no PhoneGap 50

# 1. Introdução

## 1.1 Motivação

Em 2010, a venda de smartphones ultrapassou a venda de computadores (WEINTRAUB, 2011), e esse volume de vendas continuou crescendo até os dias de hoje. Esse fato fez com que o mercado de desenvolvimento de aplicações para dispositivos móveis crescesse, com o surgimento de muitas opções de ferramentas para realizar este desenvolvimento.

A Figura 1 apresenta a divisão do mercado de sistemas operacionais. Para cada sistema existe uma linguagem nativa de desenvolvimento, fazendo com que as empresas que queiram atender a todos os consumidores tenham que desenvolver o mesmo aplicativo diversas vezes, utilizando diferentes linguagens e ambientes de desenvolvimento.

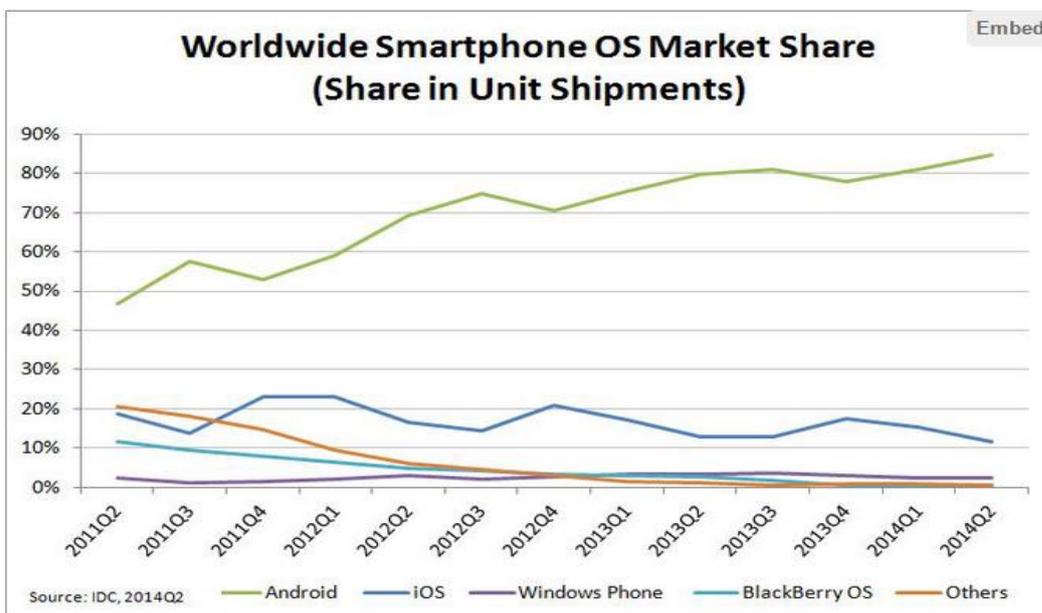


Figura 1: Mercado global dos sistemas operacionais entre abril de 2011 e junho de 2014 (IDC, 2014)

Como alternativa, surgiram diversos *frameworks* multiplataforma que permitem que a aplicação seja desenvolvida em uma única linguagem, e executada em diferentes sistemas operacionais. Entretanto, apesar dessas soluções para múltiplas plataformas possibilitarem o desenvolvimento em uma única linguagem e ambiente, elas podem afetar negativamente o desempenho da aplicação resultante, impossibilitar a utilização de algum recurso, ou até mesmo, não aproveitar toda a capacidade do hardware.

## 1.2 Objetivos

Nesse contexto, torna-se importante analisar a estrutura desses *frameworks* de desenvolvimento multiplataforma, com o objetivo de definir as suas possibilidades e limitações.

Desta forma, o objetivo deste trabalho é analisar o *framework* PhoneGap em relação à sua estrutura de funcionamento, documentação disponível, sua curva de aprendizado e seu desempenho, em comparação com o desenvolvimento nativo em Android utilizando Java.

## 1.3 Organização do texto

O presente trabalho está estruturado em capítulos e, além desta introdução, será apresentado da seguinte forma:

- O capítulo 2 justifica a escolha do PhoneGap, apresenta os diversos tipos de solução, uma breve comparação das suas estruturas, suas formas de utilização, sua posição no mercado, além de apresentar a opinião dos desenvolvedores sobre essas soluções.
- O capítulo 3 descreve detalhes do PhoneGap, isto é, como é realizado o desenvolvimento com esse *framework*, suas limitações, vantagens e APIs disponíveis para acesso aos recursos de hardware e banco de dados.
- O capítulo 4 apresenta um mesmo aplicativo desenvolvido tanto em PhoneGap quanto em Java, com o objetivo avaliar os prós e contras de cada solução. É apresentada uma comparação entre essas soluções, considerando aspectos como

documentação, curva de aprendizado, performance da aplicação resultante, e possibilidades de reutilização de código entre plataformas distintas.

- O capítulo 5 reúne as considerações finais, assinala as contribuições do trabalho e sugere possibilidades de posterior evolução.

## 2. Soluções de desenvolvimento para plataformas móveis

Para evitar a necessidade de escrever códigos distintos de uma mesma aplicação, desenvolvedores, bem como empresários, estão tentando buscar soluções que os permitam implementar soluções para diferentes plataformas móveis com um único conjunto de código fonte.

Seguindo essa nova tendência, diversos *frameworks* multiplataforma foram lançados a fim de suprir a demanda e conquistar a comunidade de desenvolvedores. Dada a ampla oferta de soluções disponíveis nesse mercado, a escolha do *framework* mais adequado para atender às demandas pode tornar-se uma tarefa não trivial.

Este capítulo apresenta as abordagens multiplataforma existentes, o mercado dessas soluções a partir de uma pesquisa realizada com desenvolvedores e o posicionamento do PhoneGap nesse contexto.

### 2.1 Abordagens Multiplataforma

Existem essencialmente duas abordagens de soluções para desenvolvimento para múltiplas plataformas móveis atualmente: (a) solução web - aplicativos web tratados como aplicativos nativos, como o PhoneGap; (b) solução nativa - *frameworks* que permitem o desenvolvimento de aplicativos nativos, como o Xamarin (SHIELDS, 2014).

#### 2.1.1 Solução web

Esta abordagem consiste em *frameworks* web que fazem com que aplicativos desenvolvidos em HTML5 e JavaScript sejam construídos e instalados como se tivessem sido desenvolvidos em linguagem nativa, ou seja, seu resultado final não apresenta diferenças para outros aplicativos provindos da linguagem nativa. Essas soluções empacotam as aplicações de acordo com o sistema alvo, para que possam ser publicadas e instaladas diretamente das lojas oficiais de cada plataforma.

Quando os recursos web não atendem completamente as necessidades de desenvolvimento do aplicativo, por exemplo, por não conseguirem acessar recursos dos dispositivos, alguns *frameworks* oferecem a possibilidade de desenvolvimento de *plugins* em linguagem nativa que atendam essa demanda e se comuniquem com o restante da aplicação. Neste caso, o *framework* é chamado de plataforma híbrida de desenvolvimento.

### 2.1.2 Solução nativa

Esta abordagem consiste em *frameworks* que buscam fazer com que os aplicativos desenvolvidos apresentem performance próxima à obtida com o desenvolvimento nativo. O desenvolvimento é feito em apenas uma linguagem, sendo que a camada de interface do usuário é implementada com os elementos nativos de cada plataforma alvo. Isso faz com que seja necessário o conhecimento das plataformas alvo, mas, ao mesmo tempo, garante aos usuários uma experiência mais parecida com os aplicativos nativos, por não se limitar aos recursos disponíveis em interfaces web. Esses *frameworks* são compilados em linguagem binária, não interpretados, aumentando seu desempenho em relação a outras soluções multiplataforma.

O Xamarin, por exemplo, oferece o desenvolvimento em C#, onde o código é executado por uma máquina virtual Xamarin presente em cada plataforma alvo. A *Figura 2* apresenta a estrutura do Xamarin.

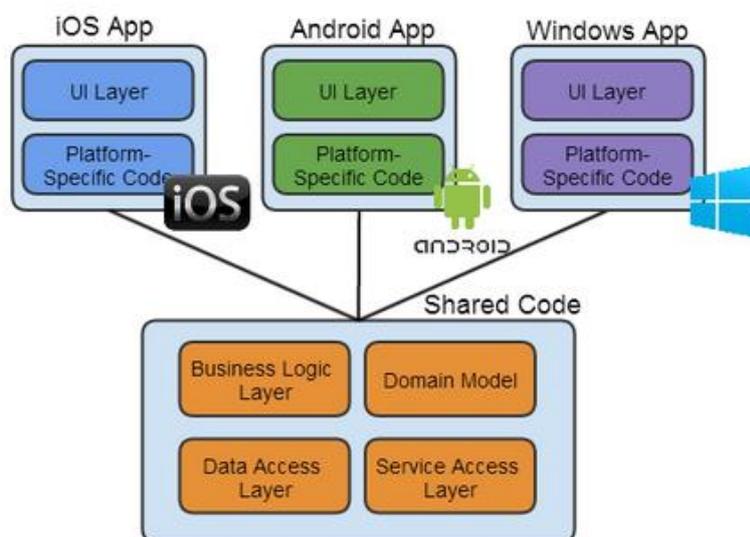


Figura 2: Estrutura de funcionamento do Xamarin

(SHIELDS, 2014)

A camada de interface do usuário é implementada com os elementos específicos de cada plataforma, enquanto as demais camadas de desenvolvimento da aplicação são desenvolvidas em um único código compartilhado C#. Com essa estrutura, a reutilização de código pode chegar a até 75% (SHIELDS, 2014).

## **2.2 Categorias de *frameworks***

O mercado de *frameworks* multiplataforma (também chamado de Cross Platform Tools, CP Tools ou até mesmo CPT) vem se tornando cada vez mais competitivo, tendo crescido 97% do ano de 2012 para o de 2013, com uma expectativa de crescimento de 14% para o ano de 2014 (Research and Markets, 2014). A pesquisa definiu seis categorias neste mercado, onde as classificações foram feitas de acordo com o próprio posicionamento das empresas (Research2guidance, 2014).

### **2.2.1 App Factories**

Estes *frameworks* têm um ambiente de desenvolvimento no estilo *drag & drop* (arrastar e soltar), ou seja, grande parte das necessidades do aplicativo já está pré-desenvolvida, bastando apenas que se “monte” o aplicativo. São voltados para aplicativos de baixo e médio custo, e que não requerem grandes habilidades de desenvolvimento.

Exemplos: AppShed, AppYourself, Como, GameBuilder Studio, GameSalad, iGenApps, Magmito, MobAppCreator, MobiForms, Vizi Apps, Weever Apps.

### **2.2.2 Cross-Platform Integrated Development Environments (CP IDEs)**

As ferramentas dessa categoria oferecem uma estrutura de desenvolvimento sólida, oferecendo um SDK próprio. São voltadas principalmente para a abordagem de solução nativa, porém algumas também suportam soluções web. Tem um nível de complexidade mais alto do que as demais.

Exemplos: Appcelerator, Embarcadero Appmethod, Marmalade, NeoMAD, Qt, Smartface App Studio, Titanium, Unity, V-Play, Xamarin.

### **2.2.3 Web App Toolkits**

Esta categoria mescla características dos App Factories com os CP IDEs, porque permite tanto o desenvolvimento por pessoas sem experiência através de templates pré-instalados, quanto de desenvolvedores mais experientes por meio de diversas linguagens web. Ela disponibiliza diversas bibliotecas, teste automatizados, prototipação e suporte para diversos formatos de tela.

Exemplo: Vaadin, que oferece uma interface intuitiva de criação, mas também permite desenvolvimento Java para os mais avançados.

### **2.2.4 CP IDEs for Enterprise (CP IDEs Enterprise)**

Voltadas para o desenvolvimento de aplicações que precisam integrar dados e outros sistemas corporativos, possuindo flexibilidade para adaptação ao ambiente da empresa. A quantidade de APIs pré-instaladas é maior se comparada a outras alternativas, tais como ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) ou sistemas de compras. Tem um grau de complexidade maior do que os demais.

Exemplos: AnyPresence, AppConKit, Appear IQ, EachScape, Feed

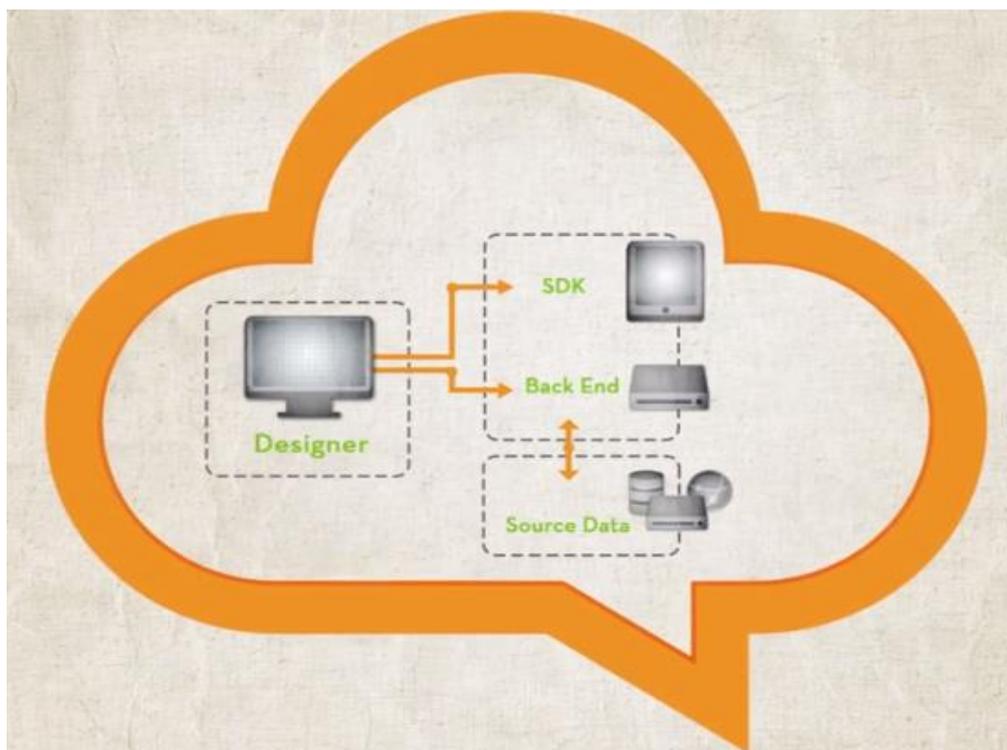


Figura 4: Estrutura do AnyPresence  
(AnyPresence, 2014)

A Figura 3 apresenta a estrutura do AnyPresence, que oferece uma solução que não se importa somente com os diversos ambientes em que deve rodar, mas também na flexibilidade de funcionar com os diferentes tipos de dados que sua empresa já utiliza.

### 2.2.5 CP Compilers

Ferramentas que utilizam um único código fonte e os transformam em aplicações nativas. Assim é possível acessar as APIs das plataformas com mais facilidade, visto que a linguagem no baixo nível é a própria linguagem da plataforma. Algumas IDEs dessas ferramentas fazem uso destes compiladores, e, portanto, não precisam criar uma solução própria.

Exemplos: Alchemo, PhoneGap Build, Cocoon.

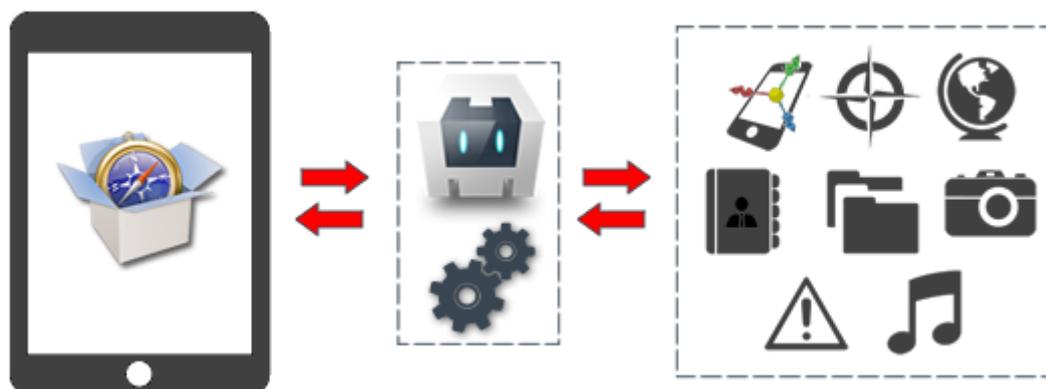


Figura 5: Estrutura do PhoneGap  
(PhoneGap, 2014)

A estrutura da Figura 4 ilustra a estrutura de funcionamento do PhoneGap. A interface com o usuário é construída em HTML5 e CSS. A lógica do aplicativo é implementada em JavaScript, que a API do PhoneGap interpreta e executa na plataforma nativa. Os acessos aos recursos de hardware são feitos através de *plug-ins* desenvolvidos na linguagem nativa.

### 2.2.6 CP Services

Voltada para não desenvolvedores, esta categoria de solução também oferece a montagem de aplicativos, com o diferencial de oferecer diversos serviços na nuvem, tais como ambientes de testes e análise de dispositivos. Os serviços em nuvem também facilitam a integração em diferentes plataformas mobile, compras dentro do aplicativo (in-app-purchases), propaganda, notificações, entre outras.

Como todos os recursos necessários para o desenvolvimento da aplicação estão disponíveis em servidores, as máquinas do desenvolvedor não precisam ter uma estrutura que suporte esses sistemas, sendo necessário apenas o acesso aos servidores, economizando assim dinheiro em infraestrutura.

Exemplos: Appurify, Capriza, Codename One, MobileSmith

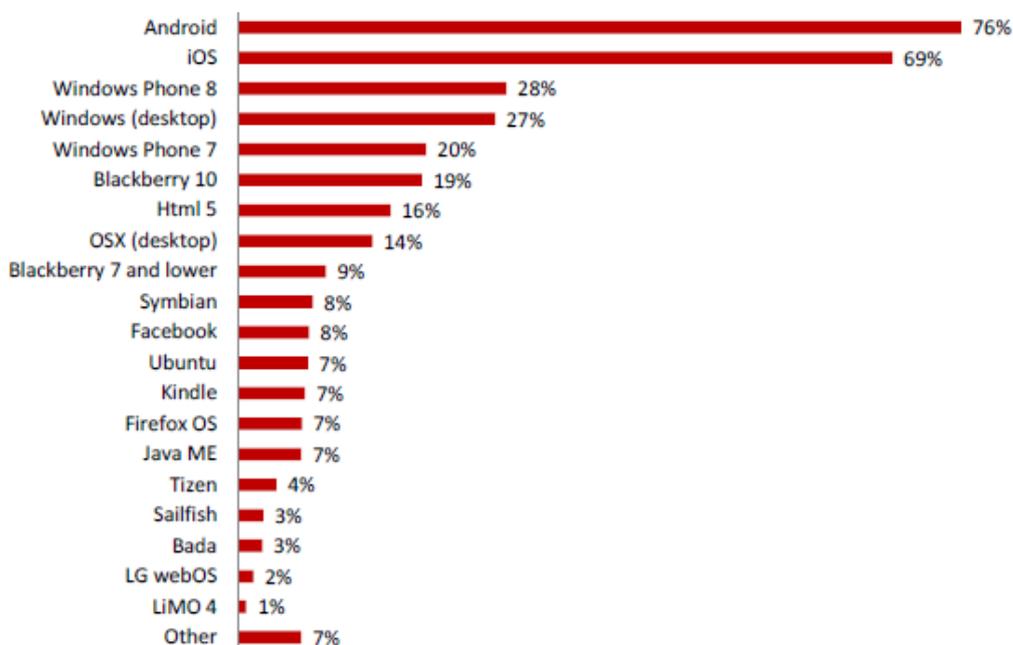
## 2.3 Mercado dos *frameworks*

Para identificar o *framework* com maior impacto no universo de desenvolvimento, é preciso ter acesso a dados que mostrem a dimensão e o crescimento de cada um, seus pontos fortes e fracos e as plataformas mais populares.

Uma pesquisa publicada em julho de 2014 (Cross Platform Tool Benchmarking, 2014) realizada pela research2guidance, uma empresa que presta serviços de consultoria e pesquisa de mercado, apresentou a opinião e experiência de 2188 desenvolvedores sobre 40 *frameworks* multiplataforma. A pesquisa aborda diferentes tópicos, desde a capacidade de utilização de um mesmo código fonte para diferentes plataformas, até o tempo necessário para se tornar um especialista, além de apresentar as plataformas que com a maior quantidade de aplicativos publicados.

Para entender quais são os *frameworks* mais utilizados, é preciso avaliar as maiores demandas que os desenvolvedores possuem. A Figura 5 apresenta as plataformas onde os desenvolvedores de *frameworks* multiplataforma publicam seus aplicativos.

research2guidance 24: Platforms CPT users publish apps on



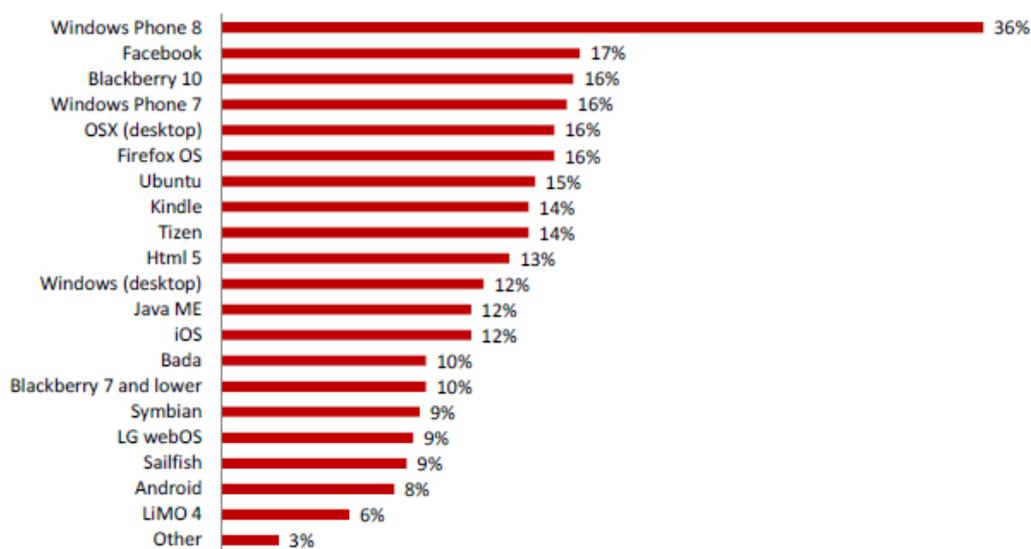
Source: research2guidance, CPT Benchmarking Study 2014, n=2,188

Figura 6: Plataformas nas quais são publicadas aplicativos de CTP

Assim como no gráfico da Figura 1 apresentado no capítulo 1, o Android também lidera o ranking de sistemas operacionais que são alvos dos desenvolvedores. A diferença do segundo colocado, o iOS, para o terceiro, o Windows Phone 8 e seus sucessores é grande, e o motivo vai além da popularidade dos sistemas operacionais.

A Figura 6 os sistemas nos quais os desenvolvedores mais sentem falta de suporte nas soluções multiplataforma, o que tem impacto direto no oferecimento dos aplicativos para esses ambientes.

#### research2guidance 25: Platforms missing



Source: research2guidance, CPT Benchmarking Study 2014, n=2,188

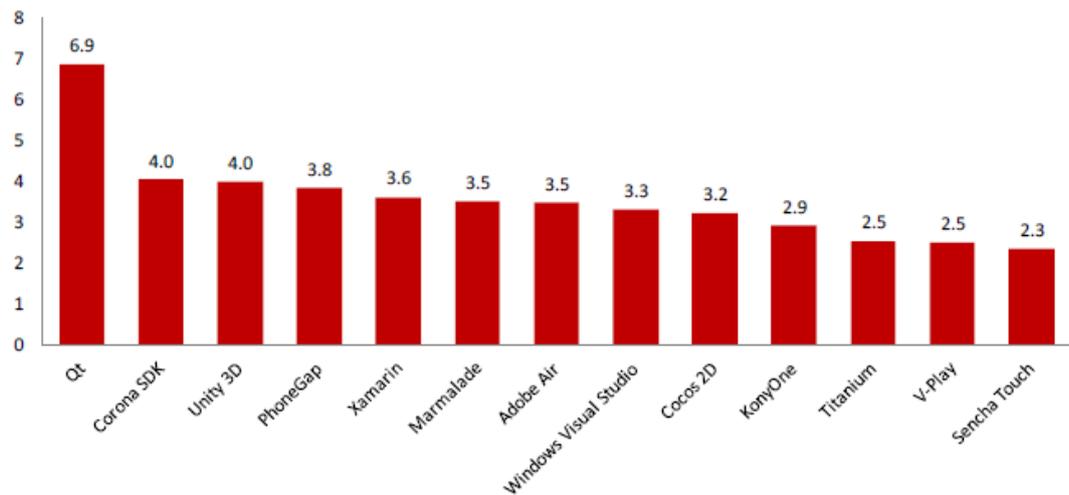
Figura 7: Plataformas que os desenvolvedores mais sentem falta nos *frameworks* utilizados

Mais de um terço dos entrevistados sentem falta do suporte ao Windows Phone 8 no *framework* utilizado, um número bastante expressivo por se tratar do terceiro sistema com maior mercado. Isso mostra que as soluções disponíveis ainda têm lacunas a preencher neste mercado multiplataforma. Outro ponto que vale ser ressaltado é que até mesmo os líderes do mercado são citados nessa pesquisa. O Android com 8% e o iOS com 12% mostram que, mesmo com um grande número de aplicativos publicados nesses sistemas, ainda há muito espaço para crescimento.

Para analisar a portabilidade oferecida por um *framework* multiplataforma, podemos avaliar o número de plataformas em que os aplicativos desenvolvidos com

esses *frameworks* são tipicamente publicados. A Figura 7 apresenta o número médio de plataformas que recebem um mesmo aplicativo.

**research2guidance 26: Average number of platforms which users publish their apps developed with a CP Tool on**



Source: research2guidance, CPT Benchmarking Study 2014, n=2188

Figura 8: Número de plataformas alcançadas por um aplicativo do *framework*

Quase sete plataformas, em média, recebem um aplicativo desenvolvido com o *framework* Qt, enquanto a média dos treze melhores colocados é de 3,53 plataformas por aplicativo. Entre os seis primeiros, a única solução de código aberto é o PhoneGap.

## 2.4 PhoneGap

Além de ser um *framework* de código aberto, o que facilita a busca de conteúdo para estudo, o PhoneGap é listado entre as três soluções preferidas nas seis das sete plataformas com maior fatia do mercado. A Figura 8 mostra que, apenas para o Windows Phone 8, o PhoneGap não está entre os três *frameworks* preferidos.

research2guidance 42: Top 10 most used, platform-specific CP Tools

Platform/OS	Mobile platform share	Preferred CPT	CPT share
Android 	77%	PhoneGap Corona SDK Unity	9% 9% 9%
iOS 	69%	Xamarin Corona SDK PhoneGap	10% 9% 9%
Windows Phone 8 	28%	Marmalade KonyOne Xamarin	13% 11% 10%
Windows Phone 7 	20%	Qt Windows Visual Studio PhoneGap	20% 13% 8%
BlackBerry 10 	20%	Marmalade Qt PhoneGap	21% 16% 8%
HTML 5 	16%	PhoneGap KonyOne Sencha Touch	17% 14% 8%
BlackBerry 7 	9%	KonyOne PhoneGap NeoMAD	24% 12% 10%

Figura 9: Os três *frameworks* preferidos para as sete maiores plataformas

A Figura 8 também reitera a preferência dos desenvolvedores multiplataforma pelo Android, o que explica a utilização da linguagem nativa desse sistema operacional na comparação entre um *framework* e uma linguagem nativa no capítulo quatro.

Na pesquisa, o PhoneGap também ficou em décimo lugar entre os *frameworks* mais recomendados, que otimizam o desenvolvimento (por causa da reutilização de código) e oferecem a melhor estrutura. Nono lugar nos rankings custo/performance e nível de compreensão por alterações de funcionalidades, além do quinto lugar na lista das soluções de mais baixa complexidade. Esses dados justificam a escolha do PhoneGap como *framework* alvo do estudo deste trabalho.

## 3. PhoneGap

Este capítulo apresenta as principais características do PhoneGap, i.e., seu funcionamento, sua estrutura, configuração necessária, e explica as principais APIs de acesso já existentes. Explica também as melhores opções de banco de dados para serem utilizadas com o PhoneGap.

### 3.1 Visão Geral de Funcionamento

O PhoneGap é um *framework* de licença livre onde a interface dos aplicativos é desenvolvida em HTML5, CSS e JavaScript. A camada que o usuário tem acesso é exibida através de uma *Web Browser View*, ou seja, uma espécie de navegador web. A diferença é que não existe barra de ferramentas ou qualquer outra característica de um browser padrão, e.g. Internet Explorer, Google Chrome, oferecendo 100% do display existente para a aplicação.

Vale ressaltar que uma aplicação feita em PhoneGap que seja executada em um navegador ocupará apenas o espaço oferecido pelo navegador como tela. A barra de ferramentas e outros recursos que possam existir continuam aparecendo, visto que, nesse caso, as dimensões da aplicação ficam limitadas ao ambiente na qual está sendo executada.

A utilização da *Web View* facilita o oferecimento da aplicação em diversas plataformas, visto que todos os sistemas operacionais que o PhoneGap suporta também usam a mesma *Web View* em suas linguagens nativas. A linguagem nativa do iOS, Objective-C, a utiliza na classe *UIWebView*, enquanto no Android o pacote responsável por utilizá-la é o *android.webkit.WebView*. As poucas diferenças existentes são a renderização das imagens e o acesso a informações e recursos de hardware.

Em baixa plataforma, o código HTML5, CSS e JavaScript se torna binário, para assim ser distribuído para as plataformas. Isso é necessário porque o formato de pacote de uma aplicação é diferente para cada sistema operacional. Enquanto o Android aceita aplicações em formato APK, o iOS aceita apenas aplicações em formato IPA.

A arquitetura típica de uma aplicação PhoneGap é que ela funciona como um cliente que interage diretamente com o usuário, ao mesmo tempo em que ela se comunica com um servidor para envio e recebimento de dados.

O servidor da aplicação é, em sua maioria, um servidor web que pode ser desenvolvido em diversos tipos de linguagem, desde que ele utilize os protocolos padronizados para a web. O servidor da aplicação também é o responsável por qualquer comunicação com servidores de dados.

A Figura 9 mostra como o PhoneGap atua somente na parte do cliente. Sua comunicação com o servidor ocorre por protocolos nos padrões web e o servidor é que tem a tarefa de comunicação com o banco de dados.

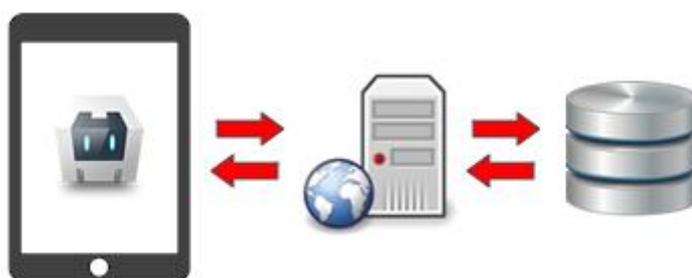


Figura 10: Arquitetura de um aplicativo PhoneGap  
(PhoneGap, 2014)

A arquitetura no lado cliente também procura manter o modelo *Single-page application*, ou seja, toda a lógica da aplicação está sob uma única página HTML que nunca sai da memória do aparelho e os dados são atualizados através do DOM, Document Object Model, uma API para HTML que permite com que programas e scripts acessem e atualizem o conteúdo e a estrutura de uma página (W3C, 2004).

Apesar de não ser muito comum, modelos *Multi-page application*, onde se utilizam diversas páginas para tratar a lógica da aplicação, também são suportados. O problema dessa solução é justamente a perda de variáveis gravadas na memória, o que aumenta a necessidade de comunicação com o lado servidor, impactando assim na performance.

## 3.2 Estrutura da aplicação

As configurações do aplicativo, tais como suas informações e parâmetros, ficam em um arquivo chamado `config.xml`. Baseado nas especificações de XML da W3C, (World Wide Web Consortium, comunidade que define padrões web) o arquivo `config` pode controlar alguns aspectos de comportamento da aplicação, como orientação de tela, cor do background e nome do aplicativo.

A aplicação em si é feita como se fosse uma página web, onde um arquivo chamado `index.html`, por exemplo, contém todo o desenvolvimento feito em HTML5, além de referenciar o CSS, o JavaScript, as imagens e os arquivos de mídia utilizados. Em aplicações Multi-page, os arquivos `.html` podem tanto referenciar os mesmos arquivos, quanto cada um fazer referências diferentes, o que varia de acordo com as necessidades e quantidades de configurações diferentes para as diversas páginas, assim como a estrutura feita no projeto.

## 3.3 Configuração do PhoneGap

O PhoneGap pode ser compilado de duas maneiras: no próprio computador do usuário ou utilizando um compilador online, o Adobe PhoneGap Build.

### 3.3.1 Compilador local

Como pré-requisito para se instalar o PhoneGap em uma máquina é necessário possuir o Node JS, encontrado em <http://nodejs.org/>. Realizado o download, basta executar o arquivo.

Para instalar o PhoneGap Framework, com o prompt do Node JS aberto (baixado no passo anterior), abra o prompt de comando. Através do comando “`npm install -g phonegap`”, o *framework* será instalado. O “Hello Word” é criado pelo comando “`phonegap create <nome_projeto>`”.

Já a compilação do aplicativo é realizada através do comando “`phonegap build android`” e depois “`phonegap install android`”, neste passo, um dispositivo deve estar conectado ao computador, para que seja realizada a instalação. Caso queira realizar a compilação para outro sistema, basta substituir o final “`android`” pelo sistema desejado.

### 3.3.2 Compilador online

O compilador online é um serviço em nuvem que busca facilitar o *build* de seu aplicativo, nele é possível fazer upload do código de maneira mais simples do que com o Compilador Local, sem ser necessário utilizar linhas de código via prompt de comando. Para isso, basta realizar o upload da pasta “www” zipada, pelo site <https://build.phonegap.com/>.

Após concluído o upload, que não leva mais que 1 minuto, o código começa a ser compilado para iOS, Android e Windows Phone (por volta de 1 minuto). Concluído, os pacotes dos respectivos sistemas são disponibilizados para download, sendo o iOS restrito, para este é necessário fornecer a chave de assinatura. Após realizar o download dos pacotes, basta instalá-los em seu dispositivo.

Na Figura 10 e na Figura 11 mostram o processo de compilação online. Na primeira, é feito o upload do arquivo, enquanto na segunda é possível verificar os downloads do aplicativo já compilado para iOS, Android e Windows Phone.

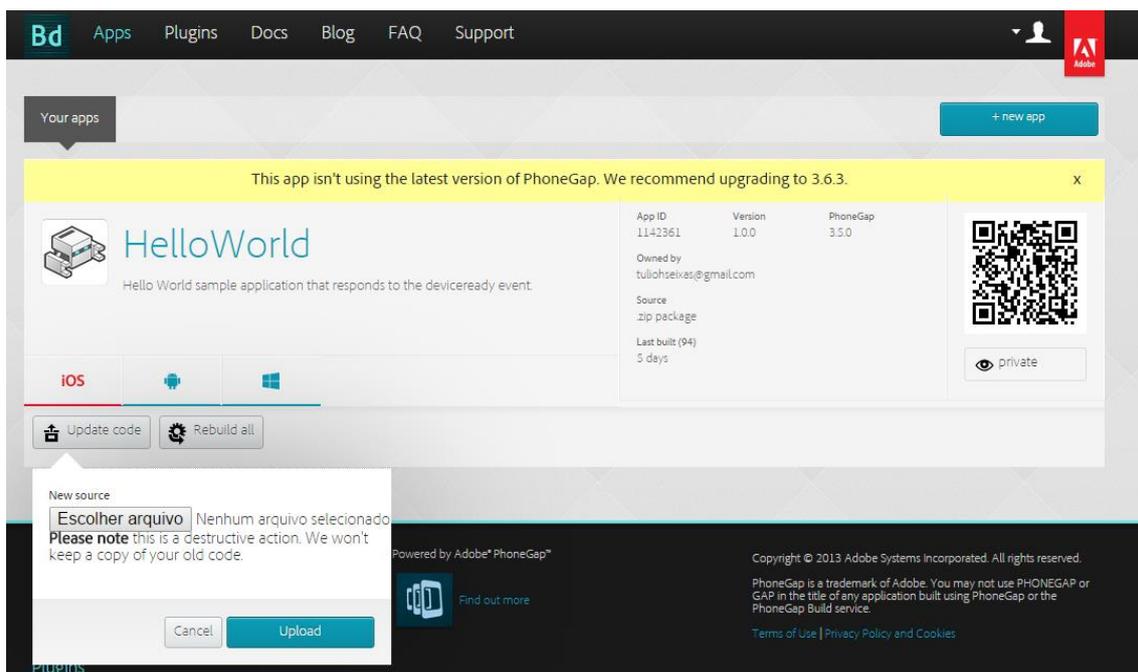


Figura 11: Upload através do Adobe PhoneGap Build  
(PhoneGap, 2014)

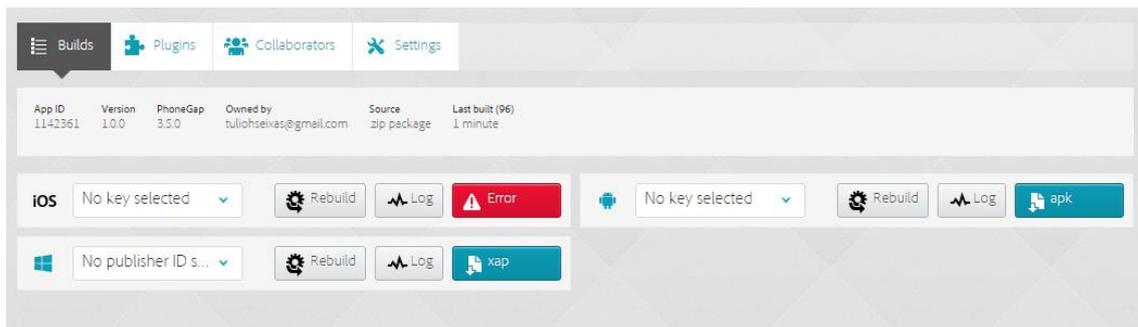


Figura 12: Compilação através do Adobe PhoneGap Build  
(PhoneGap, 2014)

### 3.3.3 Configuração da IDE

O PhoneGap pode ser configurado para rodar no próprio Eclipse. Para isso, foi utilizado o Android Developer Tools, um plugin que capacita o eclipse para desenvolver aplicações Android, encontrado em <http://developer.android.com/tools/sdk/eclipse-adt.html#downloading>.

Para começar um novo projeto, basta criar um projeto PhoneGap, como mostrado no tópico 3.3.1, e importa-lo para o Eclipse (File > Import > Android > Existing Android Code Into Workspace).

No que diz respeito à parte de emulação, existem inúmeras possibilidades, como o próprio emulador do Eclipse, sendo esta limitada, visto que não emula diversos comportamentos, como acelerômetro e bateria. Para a aplicação desenvolvida no projeto, em um primeiro momento tentamos utilizar o Ripple, “um emulador de ambiente móvel de várias plataformas, feito sob medida para o desenvolvimento de aplicativos móveis HTML5 e testes” (BlackBerry Academic). No entanto, o uso desse emulador não é intuitivo e não funcionou como esperado, o que nos forçou a procurar outro recurso. A Figura 12 mostra a interface do emulador Ripple. Como se pode ver, é possível emular o acelerômetro, geolocalização e bússola.

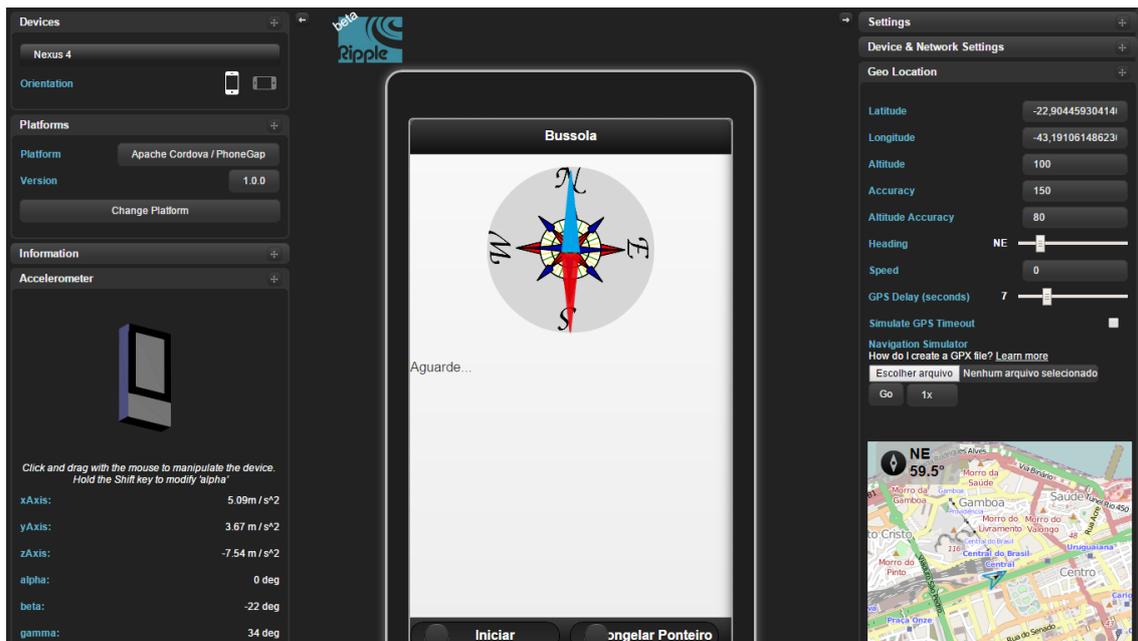


Figura 13: Interface Ripple  
(Ripple, 2014)

Dentre os recursos de emulação estudados, sendo estes o Ripple, emulador do Eclipse e a emulação feita no próprio dispositivo móvel, o último é disparado o mais rápido e eficiente, sendo o escolhido para a continuidade da emulação.

### 3.4 APIs do PhoneGap

Para acessar os diversos recursos de hardware disponíveis nos aparelhos, o PhoneGap oferece APIs (sigla para Application Programming Interface, ou seja, é uma interface de programação de aplicativos, onde um software oferece por meio de padrões suas funcionalidades para que um desenvolvedor possa usá-las sem ter que se preocupar como ela funciona internamente) que variam conforme os sistemas operacionais. A Figura 13 lista os recursos disponíveis nas APIs, assim com o sistema operacional correspondente.

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 6.0+	Blackberry 10	Windows Phone 8	Ubuntu	Firefox OS
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	✓	✓	✓	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	✓	✓	✓	X
Network	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓

✓ - supported feature  
X - unsupported feature due to hardware or software restrictions

Figura 14: Relação de APIs disponíveis no PhoneGap, de acordo com funcionalidade e sistema operacional.

(PhoneGap, 2014)

As APIs ficam disponíveis para uso através de *plug-ins* que provém as permissões, os acessos e os monitoramentos. Dependendo do sistema operacional, pode haver algumas peculiaridades que diferenciam o funcionamento do plug-in. Em seguida, veremos para que servem e como funcionam as principais funcionalidades desses recursos de hardware.

O comportamento dessas APIs quando o aplicativo é executado em um browser varia de acordo com o aplicativo e com o browser, podendo gerar valores aleatórios, como no caso do acelerômetro, ou simplesmente não exibir nada, como no caso da câmera.

### 3.4.1 Acelerômetro

O acelerômetro é um dispositivo comum nos smartphones atuais e serve para medir forças de aceleração e alteração de sua orientação. Essas forças podem ser tanto estáticas, sendo afetadas pela gravidade, quanto dinâmicas, quando a aceleração é gerada através de movimentos manuais no aparelho.

São monitorados três eixos, chamados de X, Y e Z, e de acordo com os valores de cada um é possível saber em qual posição o smartphone está. A Figura 14 apresenta o rumo que o valor de cada eixo toma dependendo da posição do aparelho, valores esses que variam normalmente entre 10 e -10. Por exemplo, um smartphone que está “em pé”, como da Figura 14, está com toda a gravidade sobre o eixo Y, fazendo com que seu valor se aproxime de 10, enquanto os eixos X e Z, não impactados, se aproximem de zero.

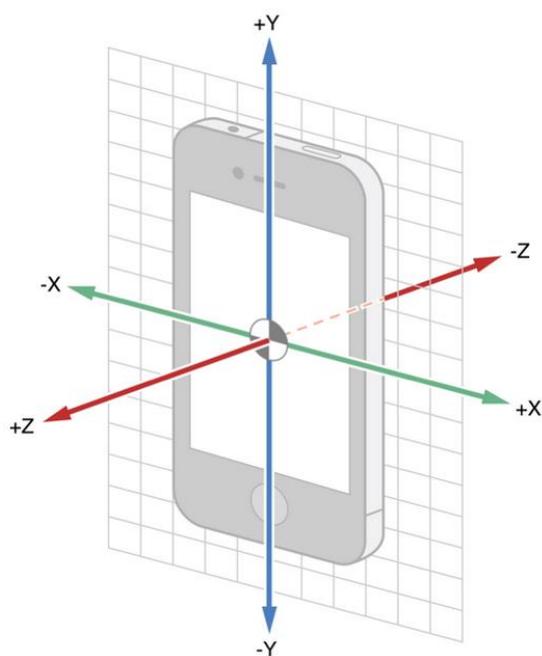


Figura 15: Comportamento dos eixos de um acelerômetro  
(Robotix, 2014)

Com isso, a principal funcionalidade do acelerômetro é justamente identificar a posição que está sendo utilizada pelo usuário, para automaticamente adaptar a apresentação da tela da melhor forma possível, ou seja, caso o aparelho mude do estado “em pé” (com base no eixo Y) para “deitado”, (com base no eixo X) a aplicação altera o formato de retrato para paisagem. Ele também é muito utilizado em jogos que interagem com a movimentação do aparelho.

No PhoneGap o acelerômetro pode ser utilizado através do objeto `navigator.accelerometer` encontrado no plug-in `org.apache.cordova.device-motion`. O método chamado `getCurrentAcceleration` retorna os valores atuais para cada eixo,

dinamicamente. Esse método funciona para todos os sistemas operacionais, menos para o iOS.

A peculiaridade do iOS é que, diferentemente dos outros, ele não consegue captar os valores dos eixos a qualquer momento, precisando de um objeto que monitore a atividade do acelerômetro em determinado intervalo de tempo. A frequência desse intervalo é de milissegundos e é restringida pelo iOS no intervalo de 40ms e 1000ms, ou seja, se a aplicação for programada para monitorar o acelerômetro apenas a cada 5000ms, a atualização será exibida apenas a cada 5000ms, mas a API atualizará internamente os valores a cada 1000ms, fazendo com que a performance de qualquer monitoramento acima de 1000ms seja igual ao de 1000ms.

### 3.4.2 Câmera

Muitos aplicativos requerem o uso da câmera disponível no aplicativo, desde os de cunho social, como os leitores de QR Code. Para atender essa demanda, o PhoneGap oferece o objeto *navigator.camera*, encontrado no plug-in *org.apache.cordova.camera*, que oferece uma API para se tirar fotos e escolher imagens em uma galeria.

O método *getPicture* é o responsável por efetuar a captura de imagens e também por permitir a escolha de uma imagem na galeria. Sua função depende da configuração do parâmetro *SourceType*.

Se ele for omitido ou definido como “câmera”, o método abrirá o aplicativo nativo do aparelho responsável por tirar fotos. Uma vez que a foto seja tirada, o aplicativo nativo fecha, voltando para a aplicação PhoneGap.

Caso o *SourceType* esteja definido como “photolibrary” o método apresentará todos os álbuns que possuam imagens do dispositivo, tanto o álbum de imagens da câmera, como o álbum de imagens recebidas em um grupo do aplicativo WhatsApp, por exemplo. Se a aplicação desejar apenas exibir o álbum de fotos padrão da câmera, o valor “savedphotoalbum” é o que deve ser utilizado no parâmetro *SourceType*.

A qualidade das fotos tiradas depende exclusivamente do hardware e todas as imagens exibidas pela aplicação serão apresentadas na melhor qualidade possível, mesmo que exista um parâmetro de qualidade que tente diminuir a qualidade da imagem.

Quando executado no Android, o aplicativo PhoneGap pode parar de funcionar caso o aparelho tenha pouca memória. Isso acontece porque quando o aplicativo nativo

da câmera é solicitado, o Android inicializa as atividades da câmera que podem consumir toda a memória disponível, fazendo com que o PhoneGap tenha sua execução interrompida.

### 3.4.3 Bússola

Para facilitar a orientação do usuário, os smartphones modernos têm uma bússola, geralmente localizada na parte superior do aparelho. O plug-in que oferece acesso ao sensor da bússola, que aponta a direção para qual o aparelho aponta, é o *org.apache.cordova.device-orientation*, que fornece o objeto *navigator.compass* que é capaz de medir em graus o apontamento do aparelho. Essa medição vai de 0 a 359,99, onde zero é o norte.

Como o aparelho pode ficar em diversas posições, a bússola precisa de auxílio do acelerômetro para que sua medição seja corretamente interpretada independentemente da posição do smartphone.

Por exemplo, caso o aparelho esteja em pé, a bússola apresenta o valor zero. Se o aparelho for virado de cabeça para baixo, ele apresentará o valor aproximado de 180, mas para o usuário o norte continuará no mesmo lugar, uma vez que apenas o dispositivo se moveu. Com o auxílio do acelerômetro é possível monitorar a inversão de valores do eixo y, assim interpretando que o valor de norte para o usuário agora é justamente 180.

Esse recurso não está disponível nas versões antigas do iOS e do BlackBerry devido a restrições de hardware, que não existem nos modelos mais recentes.

### 3.4.4 Contatos

Com o objeto *navigator.contacts* encontrado no plug-in *org.apache.cordova.contacts*, é possível acessar a lista de contatos existentes. A lista de contatos é uma informação muito sensível, por questões de privacidade, por isso é aconselhado informar ao usuário sempre que se for utilizar esse recurso.

No Firefox OS solicitar a permissão ao usuário é obrigatório, sendo necessário realizar a chamada do *permissions*, método no qual se solicita a permissão de leitura e/ou escrita. Já nos sistemas Windows, o objeto *contacts* apenas consegue ler as informações dos contatos, não sendo possível alterá-los.

### 3.4.5 Arquivos

Para leitura e escrita de arquivos no dispositivo, o PhoneGap disponibiliza o plug-in *org.apache.cordova.file* baseado na API de tratamento de arquivos do HTML5, o que facilita o trabalho dos desenvolvedores web.

Nesse recurso, a reutilização de código é mais complicada, visto que o comportamento das pastas dos arquivos varia entre os sistemas operacionais. Apesar dos parâmetros para se acessar os diretórios serem os mesmos, as permissões de leitura e escrita não são, e precisam de tratamento diferenciado para cada sistema. Além disso, algumas pastas são limpas automaticamente pelo sistema, então os diretórios nos quais o aplicativo irá gravar arquivos precisam ser analisados para cada sistema.

Essa API também pode ser utilizada como uma opção para armazenamento de dados no lado do cliente. Mais detalhes sobre essa opção são descritos na seção 3.4.10.

### 3.4.6 Geolocalização

Um dos recursos mais requisitados no desenvolvimento de aplicativos, a geolocalização oferece os dados de localização do dispositivo, como latitude e longitude. Podem ser utilizadas as informações de GPS, dados de localização descobertos através de rede, como endereço de ip, wifi, ou até mesmo de bluetooth. Informações de chip GSM também podem ser utilizadas.

Essa API é baseada na API de geolocalização da W3C. O plug-in de instalação é *org.apache.cordova.geolocation* e o objeto base de utilização é o *cordova.geolocation*.

Além de longitude e latitude, outras propriedades que podem ser acessadas pela API são: altitude, accuracy (retorna o nível de precisão entre latitude e longitude, não está disponível para o Amazon Fire e Android), altitudeAccuracy (retorna o nível de precisão da altitude), heading (direção de movimento, que indica os graus no sentido horário do atual posicionamento em relação ao norte) e speed (mede a velocidade do dispositivo, retornando valores em metros por segundo).

O monitoramento dos dados pode ser feito quando solicitado com o método *getCurrentPosition*, quando uma alteração de posição for detectada pelo aparelho ou em um determinado intervalo de tempo. Para os dois últimos, o método *watchPosition* deve ser utilizado.

### 3.4.7 API do Google Maps no PhoneGap

É possível utilizar os dados de geolocalização obtidos para exibi-los no Google Maps. Para isso, é preciso criar uma API key, ou seja, uma chave de acesso para os recursos do Google Maps. Essa é uma medida de segurança que garante que apenas aplicativos registrados tenham acesso a API. Essa chave pode ser obtida no site <https://code.google.com/apis/console/b/0/> e deve ser colocada em uma tag JavaScript no arquivo.html correspondente ao seu uso. Uma vez que os servidores do Google verificam a autenticidade da chave, é possível utilizar o objeto Maps que, com os parâmetros da geolocalização, será capaz de apresentar no aplicativo do Google Maps a localização correspondente.

Outra alternativa é utilizar as coordenadas de latitude e longitude para montar um link que direcione para o site do Google Maps. A URL possui um padrão que permite essa montagem e faz com que se exiba exatamente o local desejado. O padrão é composto da string “<http://maps.google.com/maps?z=18&q=>” concatenada com a latitude, mais o caractere vírgula, mais a longitude. Por exemplo, o link <http://maps.google.com/maps?z=18&q=-22.9538908,-43.1697161> exibirá nos mapas do Google o campus da UNIRIO, localizada na avenida Pasteur, 458. O problema desta opção é que os valores decimais das coordenadas precisam ser arredondados, porque a vírgula como separador de casas decimais é entendida como um separador de parâmetros no link, fazendo com que o resultado esperado não seja entendido. Apenas o Android consegue interpretar os decimais de forma correta.

### 3.4.8 Mídia

O plug-in *org.apache.cordova.media* permite utilizar os recursos dos dispositivos para gravar e reproduzir arquivos de áudio. O objeto *Media* é o responsável por controlar a gravação e reprodução de arquivos de mídia, podendo definir volume, alterar posicionamento de reprodução, além de poder retornar a duração e tamanho dos arquivos. Para o iOS, essa API oferece o recurso de loops de reprodução, além da opção de reprodução de áudio com a tela bloqueada.

Essa API não está de acordo com as normas de especificação da W3C, mas a documentação do PhoneGap informa que uma próxima implementação passará a estar em conformidade com tais normas, descontinuando a atual.

### 3.4.9 Notificações

Para acessar algumas funcionalidades nativas de comunicação com usuário, como as notificações, é possível utilizar o objeto *Notification* encontrado no plug-in *org.apache.cordova.dialogs*.

É possível exibir um alerta ou tela de mensagem através do método *alert*, porém a escolha de qual exibir não depende do desenvolvedor e sim da plataforma. Algumas usarão a tela de notificação nativa do dispositivo, enquanto outras farão o tratamento igual ao de um *browser* e utilizarão um alerta.

O método *prompt* dessa API oferece uma caixa de diálogo nativa com mais opções de customização de layout do que o método *prompt* padrão de desenvolvimento web. Na plataforma Android, existe um limite máximo de três botões.

Para emitir sons de notificação basta utilizar o método *beep*, passando como parâmetro o número de vezes que o som será emitido. As plataformas emitem o som de notificação configurado nos sistemas, com exceção do Windows Phone 7, 8 e o Tizen. No caso dos sistemas da Microsoft, é emitido o som de um arquivo que existe no próprio plug-in, enquanto o Tizen executa através da API de mídia um arquivo de áudio (este arquivo deve estar no diretório *sounds* e precisa ter o nome de *beep.wav*).

Também é possível utilizar a tela padrão de confirmação utilizando o método *Confirm*.

### 3.4.10 Banco de Dados

O PhoneGap, por se tratar de desenvolvimento em *web view*, tem como principais opções APIs de armazenamento do lado cliente. Na prática, isso significa que os dados passados na API são armazenados no próprio dispositivo, no mesmo lugar onde se salvam as preferências do usuário. Através da API, também é possível recuperar, manipular e realizar a pesquisa dos dados (MAHEMOFF, 2010).

Todas as APIs possuem em comum o fato de limitarem o acesso de informações armazenadas ao mesmo domínio e porta que salvou os dados. Elas também sofrem restrições de tamanho permitido por domínio, que variam de acordo com o sistema.

Existem duas formas de operação das tarefas de banco de dados. As síncronas, que só executam as próximas atividades de JavaScript depois que as operações de banco

terminam, e as assíncronas, que não interrompem as execuções de JavaScript, realizando as operações de banco em paralelo.

#### **3.4.10.1 Web Storage**

Também é chamada de Local Storage, por ser basicamente um único objeto com esse nome. É uma API síncrona que possibilita gravar e recuperar dados no objeto, mesmo após o dispositivo ser reiniciado.

Suas vantagens são que, além de ser uma API muito simples para desenvolvimento, ela é aceita por todos os sistemas operacionais e browsers, possuindo eventos de atualização de janelas.

Sua grande desvantagem é o desempenho de execução no armazenamento, procura e recuperação de dados complexos. Isso ocorre pela falta de indexação da API e pelo fato da serialização dos dados ser manual, uma vez que o banco não é estruturado.

#### **3.4.10.2 Web SQL**

É um banco de dados estruturado com todas as funcionalidades de um banco de dados relacional baseado em SQL. Com a possibilidade de indexar dados de acordo com palavras chave de busca, sua velocidade de busca é muito superior ao Web Storage.

Por ser assíncrona, melhora a performance do aplicativo em geral, já que sua atividade não interrompe a utilização pelo usuário. Ela também garante a integridade dos dados devido a sua robusta estrutura.

Sua principal desvantagem é que ela corre o risco de ser descontinuada, uma vez que as novas versões do Internet Explorer e do Firefox não aceitam mais essa API, os outros sistemas podem ir pelo mesmo caminho. O principal motivo é o fato do esquema do banco de dados ter que ser feito antes do recebimento de dados, que por sua vez tem que respeitar as definições das tabelas. Isso atrapalha o dinamismo do tipo de dado que pode ser armazenado, o que se torna um grande problema se tratando de banco de dados do lado cliente.

### **3.4.10.3 Indexed Database (Banco de Dados Indexado)**

Essa API tenta reunir as vantagens das opções anteriormente apresentadas sem manter suas fraquezas. Ela funciona como uma coleção de objetos de armazenamento, onde cada objeto se assemelha a uma tabela de um banco SQL, sem ter que respeitar qualquer tipo de restrição. Sua grande vantagem em relação ao Web Storage é que suas operações são assíncronas, melhorando o desempenho geral do aplicativo, além de ser possível a indexação dos dados, melhorando o desempenho de operações de busca.

Suas desvantagens são a complexidade da API e o fato de muitos sistemas móveis não a suportarem ainda (dos três principais, apenas o Windows Phone 8 suporta o Indexed Database).

### **3.4.10.4 API de Arquivos**

Essa API, por ter a capacidade de armazenar arquivos no hd do dispositivo, é utilizada como opção para tratamento de grandes arquivos (de áudio e vídeos, por exemplo). Por ser assíncrona, não atrapalha o desempenho da aplicação.

Suas desvantagens são a impossibilidade de indexação de dados para facilitar buscas e a falta de suporte a transições de dados cliente-servidor.

Essa opção deve ser evitada para desenvolvimento de aplicativos que também vão ser disponibilizados para browsers, visto que apenas o Google Chrome e o Opera são compatíveis.

## 4. PhoneGap x Android

Para avaliar o PhoneGap com o sistema de maior mercado, o Android, foram desenvolvidos dois aplicativos iguais em suas respectivas linguagens. Para analisar algumas das APIs de acessos a recursos do smartphone, as aplicações foram implementadas baseadas no aplicativo Sensor Sense, que tem como objetivo justamente apresentar os resultados de momento dos sensores disponíveis no Android. Foram escolhidos quatro recursos do Android para comparação: acelerômetro, bússola, geolocalização e rede.

Para comparar o desempenho de ambos quando não há necessidade de acesso a API, foi desenvolvida uma funcionalidade que busca o valor de um bilhão da sequência de Fibonacci, utilizando o mesmo algoritmo de recursividade em ambos.

As comparações foram divididas em: documentação, curva de aprendizado, performance, reutilização de código em uma migração de sistema e publicação da aplicação.

### 4.1 A aplicação

Após feita a escolha dos sensores a serem desenvolvidos (acelerômetro, bússola, geolocalização e rede), a aplicação Android foi tomada como ponto de partida. Como os testes não podiam influenciar uns nos outros, a estrutura da aplicação colocou na tela inicial um botão de acesso para cada recurso testado, com a execução da busca pelo número um bilhão da sequência de Fibonacci ocorrendo após a abertura da aplicação.

O aplicativo feito no PhoneGap utilizou a mesma estrutura, com a diferença de haver um botão para executar a busca na sequência de Fibonacci. Como apresentado anteriormente no capítulo 3, essa estrutura multi page não é recomendada, mas foi utilizada justamente para isolar os testes de acesso e evitar impacto de umas

funcionalidades nas outras. A Figura 15 exibe a tela inicial do aplicativo feito no PhoneGap.



Figura 16: Tela inicial aplicativo PhoneGap

Vale ressaltar que as aplicações foram feitas para verificar seus desempenhos, não sendo voltados para o público em geral, não levando em consideração questões de usabilidade, ergonomia, design e outros fatores importantes na criação de um aplicativo comercializado.

## 4.2 Documentação

A documentação oficial do Android é encontrada no site <https://developer.android.com/> e é estruturado em diversos menus que vão orientando o usuário. O menu principal oferece as opções de design, desenvolvimento e distribuição. A área destinada ao design oferece informações referentes à parte visual do aplicativo,

apresentando conceitos, informando funcionalidades e exibindo as diferenças de estrutura entre os aparelhos (relógios, smartphones, televisores e tablets). O menu de desenvolvimento apresenta informações de download e configuração, suporte de características de acordo com cada versão, guia de APIs, exemplos de código, além de um item de menu voltado para iniciantes, com primeiros passos e boas práticas de desenvolvimento para cada tipo de hardware. A opção de distribuição contém informações necessárias para a distribuição de aplicativos no Google Play, as formas de cobrança, como obter usuários e ferramentas de auxílio ao marketing de aplicativos. No desenvolvimento da aplicação Android, todas as informações necessárias foram encontradas nessa documentação.

Como se pode ver pela Figura 16, a documentação do Android é muito bem estruturada, com as APIs listadas no menu esquerdo e detalhadas no centro. Também é possível notar o sistema de buscas no campo direito superior, que é de grande ajuda.

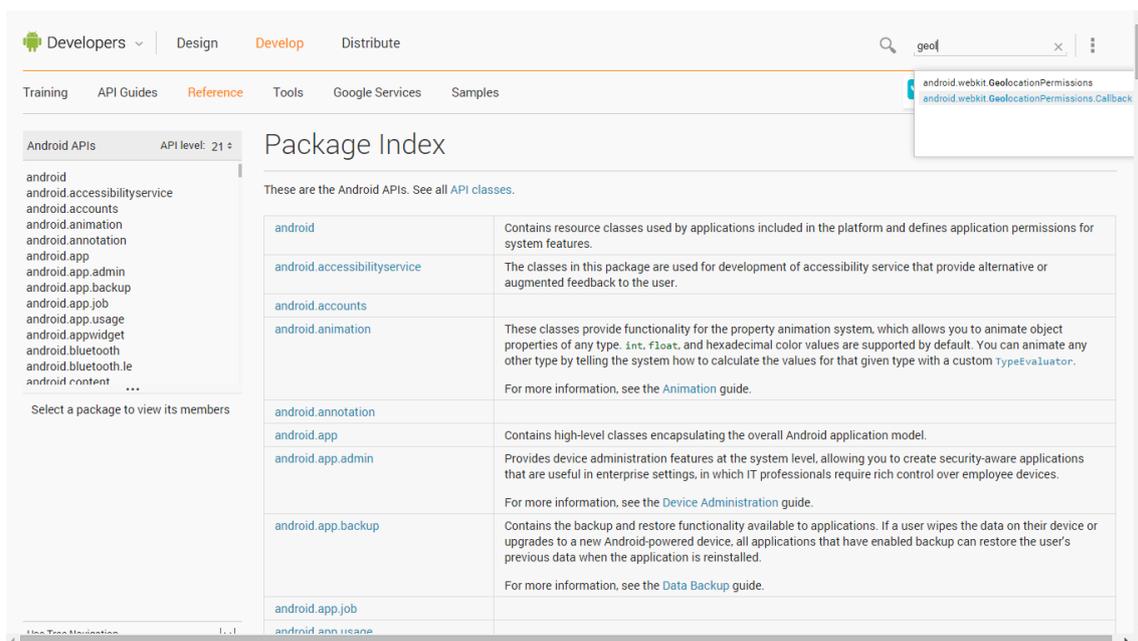


Figura 17: Documentação Android

(Android, 2014)

A Figura 17 apresenta a tela inicial do guia de APIs do Android. Os menus superiores também podem ser observados.

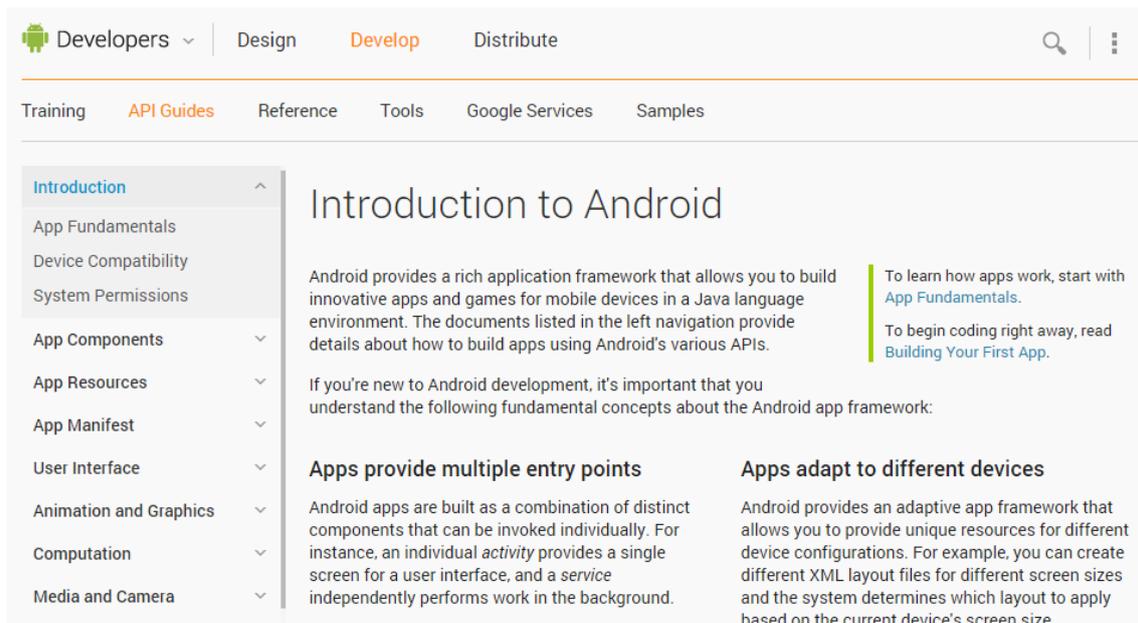


Figura 18: Tela inicial da guia de APIs  
(Android, 2014)

A documentação do PhoneGap está no site <http://docs.phonegap.com/>. Estruturado por um único menu a esquerda, sua apresentação não é tão intuitiva quanto a do Android e para informar sobre suas APIs ela redireciona o usuário para o site [github.com/apache](http://github.com/apache). A falta de um mecanismo de pesquisa também faz falta. Mesmo com esses problemas, tudo que era necessário foi encontrado na documentação do PhoneGap.

Conforme a Figura 18 ilustra, o site possui um menu lateral esquerdo com guias e referências. No centro, são listadas todas as APIs com um pequeno resumo sobre cada uma delas. Também é possível detalhá-las, ao clicar sobre elas.

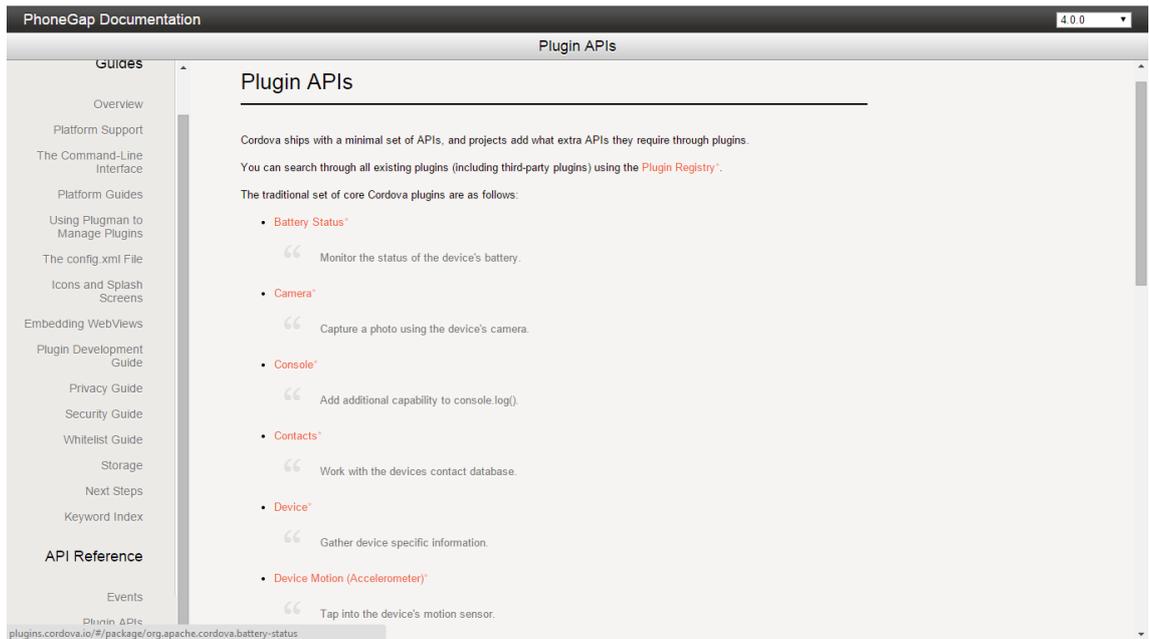


Figura 19: Documentação PhoneGap

A Figura 19 mostra uma das APIs detalhadas. O detalhamento apresenta informações sobre as plataformas suportadas, exemplos e eventos existentes nessa API.

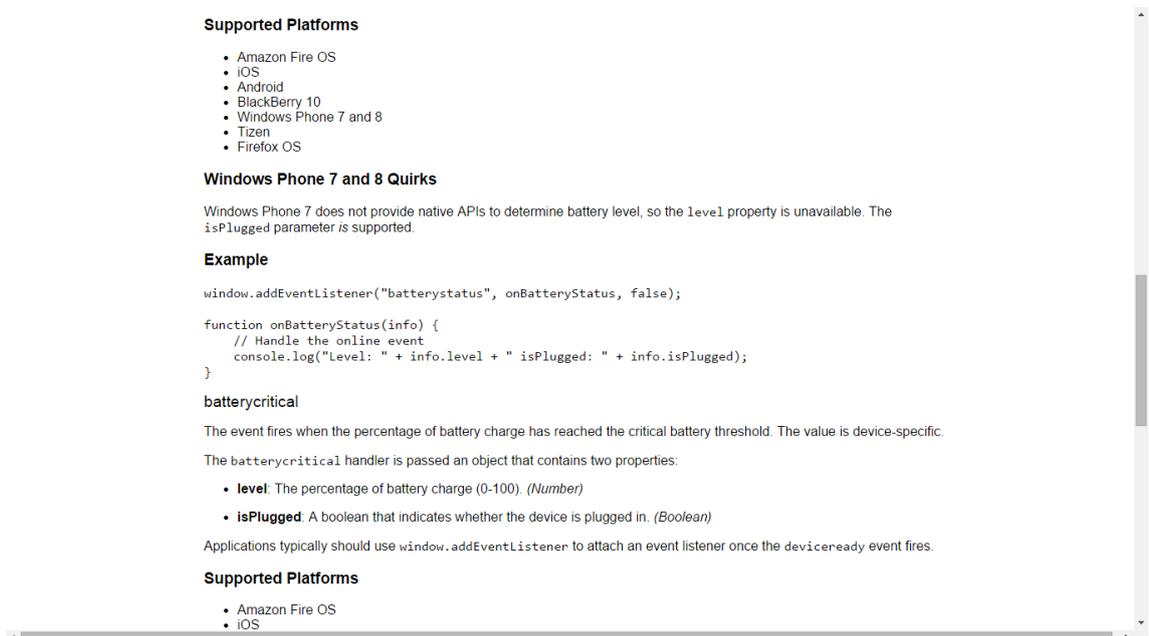


Figura 20: Tela que explica o funcionamento de um plugin do PhoneGap (PhoneGap, 2014)

A Figura 20 apresenta a tela inicial da documentação do PhoneGap.

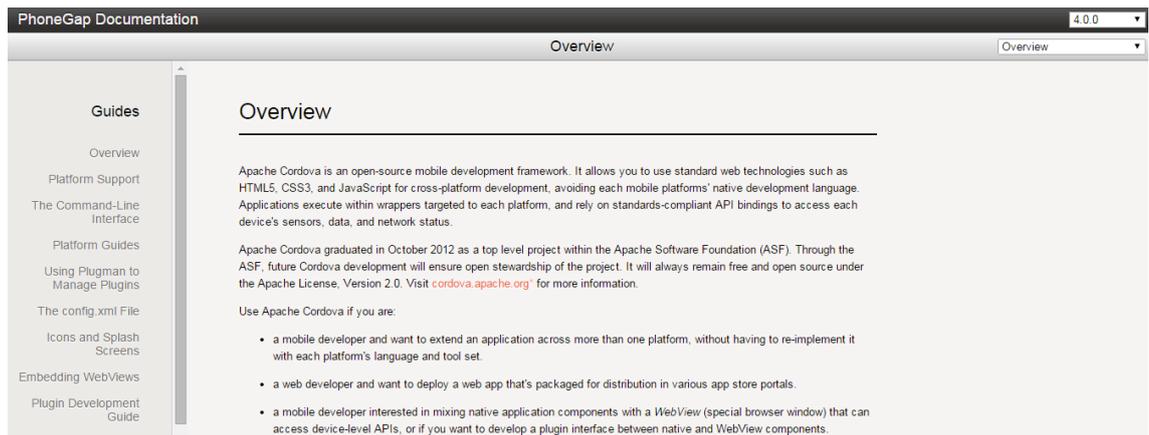


Figura 21: Página inicial da documentação do PhoneGap  
(PhoneGap, 2014)

Ambas as documentações estão em inglês, dificultando o acesso de quem tem apenas o português como idioma.

### 4.3 Curva de aprendizado

Na pesquisa da Research2Guidance o PhoneGap ficou com a quinta colocação no ranking de soluções multiplataforma com menor complexidade, com apenas 14% dos desenvolvedores considerando alta ou muito alta a sua complexidade. Como os desenvolvedores da aplicação já tinham experiência em HTML5 e JavaScript, o aprendizado foi bem rápido. As dificuldades registradas foram na instalação, configuração, *build* e teste do aplicativo. As experiências como simuladores de recursos de um smartphone (geolocalização, por exemplo) para testes foram decepcionantes, pelo fato da lentidão e dificuldade de configuração, sendo mais rápido testar no próprio dispositivo.

Apesar dos desenvolvedores também terem uma experiência em Java, o início do desenvolvimento da aplicação Android foi mais complicada. Isso aconteceu pelo fato da estrutura da aplicação ser bem diferente, levando um tempo de adaptação. Em compensação, após esse período o desenvolvimento foi muito mais ágil, por causa do conhecimento prévio da linguagem assim como da IDE de desenvolvimento, o Eclipse. Assim como no caso do PhoneGap, o emulador do Android e os simuladores de recursos foram decepcionantes por serem muito lentos quando comparados aos smartphones. A vantagem do Eclipse é que ele permite debug da aplicação no dispositivo enquanto se monitora no computador.

Dados os níveis diferentes de experiência dos desenvolvedores entre as linguagens, não é possível realizar uma comparação direta da curva de aprendizado entre ambas. O que se pode afirmar é que, em ambos os casos, sem o dispositivo alvo para qual se está desenvolvendo o processo de desenvolvimento é mais lento e trabalhoso.

#### **4.4 Performance**

Para mensurar uma possível perda de performance de um aplicativo feito no PhoneGap para um feito no Android, foram comparados os tempos de retorno dos métodos de acesso aos recursos de hardware, assim como o tempo de processamento para encontrar o valor um bilhão da sequência de Fibonacci.

Todos os testes foram feitos no smartphone LG Optimus G, que possui um processador Quad Core de 1.5GHz com 2Gb de memória RAM, rodando no sistema Android 4.2. Apenas o aplicativo era executado no momento do teste, (com exceção dos aplicativos de background) existindo sempre pelo menos 1Gb de memória RAM livre no momento da execução.

No PhoneGap, o tempo de execução foi apresentado na tela, com exceção da geolocalização, que foi exibida através de um Alert. Isso porque essa funcionalidade redireciona para o Google Maps, saindo da tela de exibição do aplicativo.

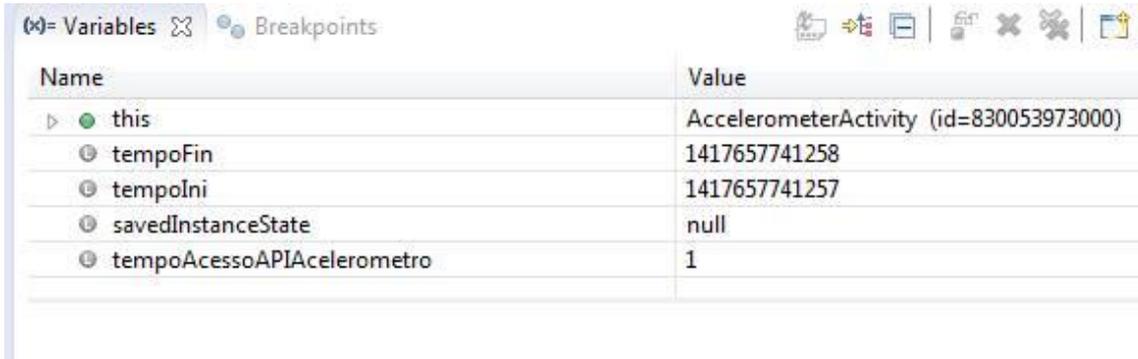
No Android, os tempos foram monitorados com ajuda do Eclipse, que exibia os valores das variáveis, sem alterar a apresentação da aplicação. O teste de Fibonacci foi o único caso em que a apresentação dos resultados foi feita na tela justamente por não ter uma funcionalidade para o usuário.

Foram feitas cinco etapas de testes de cada item e foi considerado o valor intermediário, obtendo assim uma mediana do tempo de processo, ou seja, um valor que não representa um caso extremo, seja de lentidão ou de alta velocidade de processamento. A unidade de tempo está em milissegundos, a não ser que outra referência seja feita.

##### **4.4.1 Performance do Acelerômetro**

Os testes do acelerômetro mostraram que o aplicativo Android é oito vezes mais rápido no acesso desses dados do que o aplicativo PhoneGap. A Figura 21 e a Figura 22

mostram os valores encontrados, no app Android e PhoneGap, respectivamente.



Name	Value
▶ this	AccelerometerActivity (id=830053973000)
⌚ tempoFin	1417657741258
⌚ tempoIni	1417657741257
⌚ savedInstanceState	null
⌚ tempoAcessoAPIAcelerometro	1

Figura 22: Tempo de acesso ao Acelerômetro pelo aplicativo Android

```
Eixo X: -0.767181396484375  
Eixo Y: 4.678436279296875  
Eixo Z: 8.930145263671875  
Timestamp: 1417665083809
```

```
Tempo total de comunicacao com a API, em  
milisegundos: 8
```

Figura 23: Tempo de acesso ao Acelerômetro pelo aplicativo PhoneGap

Como podemos observar, a mesma informação que o app Android obtém em apenas 1 milissegundo, o app PhoneGap precisa de 8 milissegundos para obter. Como os dados são atualizados de acordo com a mudança de valores, o aplicativo Android é oito vezes mais preciso ao retornar as informações para a aplicação.

Vale ressaltar, porém, que como se tratam de milissegundos, apenas aplicativos que necessitam de muito poder de processamento ou uma precisão muito acurada sentiriam efeitos nessa diferença. No primeiro caso porque esses valores podem subir se a necessidade de processamento da aplicação como um todo for alta e o poder de processamento do dispositivo for baixo. No segundo caso, apenas aplicativos que necessitam saber imediatamente a alteração de posição do aplicativo sentiriam alguma diferença.

#### 4.4.2 Performance da Bússola

Nessa comparação foi avaliado o tempo de retorno de informação do sensor

bússola. Não foi considerado o tempo gasto na busca de dados do acelerômetro, também utilizado para fazer a bússola. Na Figura 23 e na Figura 24 observamos os tempos gastos em ambos os aplicativos.

Name	Value
▶ this	MagneticActivity (id=830035681088)
⌚ tempoFin	1417666430213
⌚ tempoIni	1417666430213
⌚ tempoAcessoBussola	0

Figura 24: Tempo de execução para retorno de informação da bússola no Android

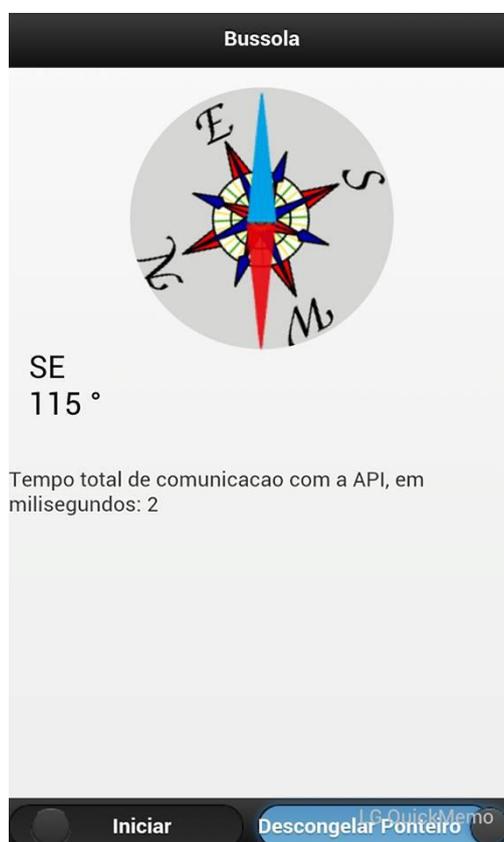


Figura 25: Tempo de execução para retorno de informação da bússola no PhoneGap

Nessa comparação a diferença é menor, enquanto o app PhoneGap precisa de dois milissegundos para atualizar os dados da bússola, o app Android precisa de menos de um milissegundo. Essa diferença não é perceptível na comparação do funcionamento das bússolas em si, mas em aplicativos com grande necessidade de processamento, o desempenho pode ser prejudicado, afetando negativamente a experiência do usuário.

#### 4.4.3 Performance de Geolocalização

Name	Value
▶ this	GPSFacade (id=830035772192)
⌚ tempoFin	1417666655244
⌚ tempoIni	1417666655240
⌚ tempoAcessoLocalizacao	4

Figura 26: Tempo de execução para retorno de dados de Geolocalização no Android



Figura 27: Tempo de execução para retorno de dados de Geolocalização no PhoneGap

Diferente dos anteriores, na geolocalização o tempo de resposta do app PhoneGap é mais rápida do que a do app Android. Essa diferença pode se dar pelo fato de que no caso do aplicativo em Android, primeiro é feito uma busca pelos dados de geolocalização do GPS e, caso esse não seja encontrado, busca os dados a partir da rede. Como no caso do aplicativo desenvolvido através do PhoneGap apenas um método é chamado, não dá para saber qual sua lógica de funcionamento, nem o porquê de sua

maior eficiência em relação ao Android.

#### 4.4.4 Performance de Rede

A velocidade de acesso à informação de qual tipo de rede o dispositivo está conectado é a que mais se assemelha em ambas as linguagens. Tanto no aplicativo feito no Android, quanto no aplicativo feito no PhoneGap levam menos de um milissegundo, mostrando que em ambos os casos são indiferentes para este recurso. Os valores obtidos são apresentados na Figura 27 e na Figura 28.

Name	Value
▶ this	MagneticActivity (id=830035681088)
⌚ tempoFin	1417666430213
⌚ tempoIni	1417666430213
⌚ tempoAcessoBussola	0

Figura 28: Tempo de acesso as informações de rede no Android

Tipo de conexao: WiFi connection

Tempo total de comunicacao com a API, em milisegundos:0

Figura 29: Tempo de acesso as informações de rede no PhoneGap

#### 4.4.5 Performance de Processamento

Na comparação de execução pura do HTML5 e JavaScript do PhoneGap com o Java do Android, o tempo de processamento entre ambos tem a maior diferença dentre os testes realizados. Os dois aplicativos executaram uma função que possui a mesma lógica de recursividade, para encontrar o valor de um bilhão da sequência de Fibonacci. A Figura 29 e a Figura 30 apresentam os resultados.



Figura 30: Tempo de execução para achar o valor um bilhão da sequência de Fibonacci no Android



Figura 31: Tempo de execução para achar o valor um bilhão da sequência de Fibonacci no PhoneGap

A velocidade de processamento do app Android é quase meio segundo superior a do app PhoneGap, o que reforça o fato de aplicativos com grande necessidade de processamento serão mais lentos no PhoneGap do que no Android.

## **4.5 Reutilização de código no desenvolvimento de aplicativo para outra plataforma**

No aplicativo desenvolvido para Android a reutilização de código é nula, visto que ele tem que ser reescrito por completo, independente do sistema alvo. Isso exige amplo conhecimento de uma nova linguagem e uma nova estrutura de sistemas, além de um maior tempo de desenvolvimento para publicar a aplicação para todos os sistemas que se deseja.

No caso do PhoneGap a história é diferente. O décimo colocado na pesquisa da Research2guidance no quesito de tempo salvo, com 79% dos entrevistados informando que salvam, pelo menos, 10% de tempo em sua utilização, tem na reutilização de código um amplo alcance a diversos sistemas. Como todas as plataformas que tem suporte ao PhoneGap entendem o HTML5 e o JavaScript, toda a parte da aplicação que não se comunica com recursos do hardware são 100% reutilizáveis.

Com relação às APIs, grande parte do código também é reutilizado, porque os métodos são comuns independente do sistema. O cuidado que deve ser tomado nesse caso são em relação às peculiaridades que podem ocorrer em alguns casos, como os apresentados no capítulo 3.

Para otimizar a reutilização do código, essas exceções precisam ser conhecidas no início do desenvolvimento, porque assim pode-se ser desenvolvido um método já adaptado a divergências de sistemas. Por exemplo, no caso de acesso ao acelerômetro, o iOS, diferentemente dos demais, não consegue obter as informações dos eixos assim que os valores são alterados precisando assim de um objeto que verifique periodicamente essas informações. Todos os sistemas aceitam ter um método de observação dos dados do acelerômetro da forma que o iOS precisa, então se essa for a forma de desenvolvimento para acesso, a peculiaridade do sistema da Apple não seria percebida e a reutilização de código nesse caso também chegaria 100%.

Porém é importante ressaltar que essa forma de desenvolvimento aumenta o tempo salvo, mas afeta a performance dos sistemas que tem uma forma mais rápida de obtenção de dados. No caso do acelerômetro, uma informação que o Android obtém em oito milissegundos, passaria a ser obtido em quarenta milissegundos, mínimo do iOS.

## 5. Conclusão

### 5.1 Considerações finais

Esse projeto apresentou as opções de *frameworks* multiplataforma que existem, uma pesquisa de mercado que mostra as percepções de desenvolvedores sobre os mesmos, além de explicar o porquê de eleger o PhoneGap como ferramenta de estudo e o Android como ferramenta de comparação.

Apresentou também a estrutura de funcionamento do PhoneGap, suas plataformas alvo, suas APIs de acesso a recursos de hardware e as peculiaridades que ocorrem ao utilizá-lo no desenvolvimento de alguns sistemas.

Por fim, foi feita uma análise comparativa entre o Android e o PhoneGap, desenvolvendo-se para cada, uma mesma aplicação. Essa comparação avaliou documentação, desempenho, curva de aprendizado e reutilização de código. Os resultados obtidos mostraram que a opção multiplataforma tem a reutilização de código como sua grande vantagem, uma boa curva de aprendizado, apesar de apresentar uma documentação confusa e um desempenho geral pior do que o do Android.

Assim, conclui-se que *frameworks* multiplataforma são uma boa alternativa quando não se tem condições, sejam estas financeiras ou de conhecimento técnico, de se desenvolver a mesma aplicação para todas as plataformas alvo desejadas.

O PhoneGap é uma boa opção para desenvolver aplicativos em que os recursos web, somados a recursos de hardware sem necessidade de grande poder de processamento, sejam suficientes. Aplicativos de uma única tela, com banco de dados no lado do cliente, sem necessidade de muita comunicação com o lado servidor, têm a estrutura ideal para este tipo de desenvolvimento. O fato de ser código aberto e 100% gratuita colabora para sua utilização.

Para aplicativos mais complexos, com muitas telas, utilizando banco de dados relacionais e que necessitam de grande comunicação com o servidor, o recomendado é utilizar uma solução que tenha uma performance mais próxima das linguagens nativas.

## 5.2 Limitações do projeto

As comparações do PhoneGap foram feitas apenas em relação ao Android, não sendo possível garantir que os resultados encontrados se apliquem a outras plataformas.

Os testes foram realizados em apenas um dispositivo, não garantindo assim que os resultados serão os mesmos, caso os testes sejam efetuados em smartphones de configurações diferentes, sejam melhores ou piores.

Por se tratar de uma aplicação simples, que tinha como objetivo apenas a execução de testes, o desenvolvimento do aplicativo no PhoneGap não abrangeu a utilização de IDEs nem formas de debug para testes em tempo de execução.

## 5.3 Projetos futuros

Dar continuidade à pesquisa sobre o PhoneGap, comparando sua utilização nos outros sistemas de maior mercado, iOS e Windows Phone 8. Pode-se também avaliar o processo de criação de novos *plug-ins* para acesso a recursos que ainda não sejam atendidos.

Avaliar o porquê dos desenvolvedores citarem o Windows Phone 8 como o sistema em que eles mais sentem falta de suporte pelos *frameworks* que eles utilizam. Verificar as soluções necessárias para acabar com essa lacuna do mercado.

Realizar um estudo sobre o *framework* multiplataforma mais utilizado, o Xamarin, no segundo sistema de maior mercado, o iOS.

Após o lançamento do estudo de mercado de 2015, comparar o crescimento e utilização dos *frameworks* multiplataforma, analisando as possibilidades, novas tecnologias e adesão do mercado.

## Referências Bibliográficas

Research2Guidance (2014) “Cross Platform Tool Benchmarking 2014”, disponível em: <http://research2guidance.com/cross-platform-tool-benchmarking-2014>.

Research and Markets (2014) “Cross Platform Mobile Development Tools: Market Analysis & Forecast - Third Edition”, disponível em: <http://www.researchandmarkets.com/reports/2896434/cross-platform-mobile-development-tools-market>.

Mahemoff, Michael (2010). “Client-Side Storage”, <http://www.html5rocks.com/en/tutorials/offline/storage/>, acessado em novembro de 2014.

Jiang, Jeniffer (2012). “PhoneGap Compass sample”, <https://github.com/kublaj/sample-phonegap-compass>, acessado em outubro de 2014.

Weintraub, Seth (2011). “Industry first: Smartphones pass PCs in sales”, <http://fortune.com/2011/02/07/industry-first-smartphones-pass-pcs-in-sale/>, acessado em setembro de 2014.

Documentação oficial do PhoneGap. Disponível em: <http://docs.phonegap.com/>, acessado em setembro de 2014.

Documentação oficial do Android. Disponível em: <http://developer.android.com/guide/index.html>, acessado em setembro de 2014.

W3C Recommendation (2004). Disponível em: <http://www.w3.org/TR/DOM-Level-3-Core/introduction.html>, acessado em dezembro de 2014.

Trice, Andrew (2012). “PhoneGap Explained Visually”, <http://phonegap.com/2012/05/02/phonegap-explained-visually/>, acessado em dezembro de 2014.

Trice, Andrew (2012). “Getting started with PhoneGap in Eclipse for Android”, <http://www.adobe.com/devnet/archive/html5/articles/getting-started-with-phonegap-in-eclipse-for-android.html>, acessado em dezembro de 2014.

Shield, Justin (2014), “Cross-Platform Mobile Development: PhoneGap vs Xamarin”, <http://www.justinshield.com/2014/05/cross-platform-mobile-development-phonegap-vs-xamarin/>, acessado em dezembro de 2014.