UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

FÁBIO CHASSOT DO ROSÁRIO

DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2015

FÁBIO CHASSOT DO ROSÁRIO

DESENVOLVIMENTO DE APLICATIVOS MÓVEIS MULTIPLATAFORMA

Trabalho de Conclusão de Curso apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – COADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof Márcio Angelo Matté.

MEDIANEIRA

2015

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Desenvolvimento de aplicativos móveis multiplataforma

Por

Fábio Chassot do Rosário

Este Trabalho de Diplomação (TD) foi apresentado às 14:50 h do dia 16 de novembro de 2015 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Medianeira. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Márcio Angelo Matté.
UTFPR – Câmpus Medianeira
(Orientador)

Prof. Dr. Evando Carlos Pessini
UTFPR – Câmpus Medianeira
(Convidado)

Prof. Me. Ricardo Sobjack
UTFPR – Câmpus Medianeira
(Convidado)

Prof. Me. Jorge Aikes Junior
UTFPR – Câmpus Medianeira
(Responsável pelas atividades de TCC)

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

RESUMO

ROSARIO, C. FÁBIO, 2015. Desenvolvimento de aplicativos móveis multiplataforma. 2015. 39 f. Trabalho de Diplomação (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas). Universidade Tecnológica Federal do Paraná – UTFPR. Medianeira. 2015

Com o crescente aumento na utilização de dispositivos *smartphones* seguida pela constante evolução de suas plataformas e aplicações, surgem diferentes arquiteturas e *frameworks* para suprir as necessidades no desenvolvimento de novas aplicações, o objetivo deste trabalho é apresentar o *framework Cordova* e sua arquitetura de comunicação com a plataforma móvel *Android*. Apresentando um estudo comparativo na invocação de recursos nativos Android através dos *plugin cordova-plugin-contacts* disponibilizados no framework *Cordova*.

Palavras-chave: *Cordova, cordova-plugin-contacts, Android*.

RESUMO EM LINGUA ESTRANGEIRA

ROSARIO, C. FÁBIO, 2015. Developing mobile applications multi-platform. 39 f. Trabalho de Diplomação (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas). Universidade Tecnológica Federal do Paraná – UTFPR. Medianeira. 2015

With Ascending increased use of smartphones followed by a constant evolution of mobile platforms and applications , there are different architectures and structures to meet such needs on developing a new application, the objective is to show the Cordova framework communication architecture with Android platform. Featuring a comparative study in the invocation of native Android Features through the cordova-plugin-contacs available on Cordova framework.

Keywords: *Cordova, cordova-plugin-contacs, Android*.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Os dispositivos móveis vem proporcionando uma revolução na forma de comunicação e interação na sociedade. Através do crescente avanço na disponibilização de acesso a Internet, estes aparelhos ganharam destaque no cotidiano de uma boa parcela da população. A universalização de padrões técnicos e a ampliação de conteúdos e acessos à rede internet se intensifica, resultando num aumento significativo de aquisições de *smartphones* para a conexão. Até a metade de 2014, dos 7 bilhões de habitantes do planeta, pouco mais de 40% tinham acesso à internet (WASHINGTON POST, 2014).

Com esta popularização, houve um grande investimento na melhoria dos dispositivos e suas funcionalidades. Fomentados por uma indústria tecnológica composta principalmente por grandes fabricantes como Samsumg, Apple, Google, Nokia e BackBerry, desenvolveram sistemas operacionais diferentes focados no seu próprio hardware. Segundo pesquisas da fundação *StatCounter Global Stats* os sistemas operacionais que possuem maior parte dos usuários são *Android* e iOS. Como pode-se observar pela figura 1 esses dois nomes ocupam mais de 80% dos sistemas operacionais utilizados atualmente.

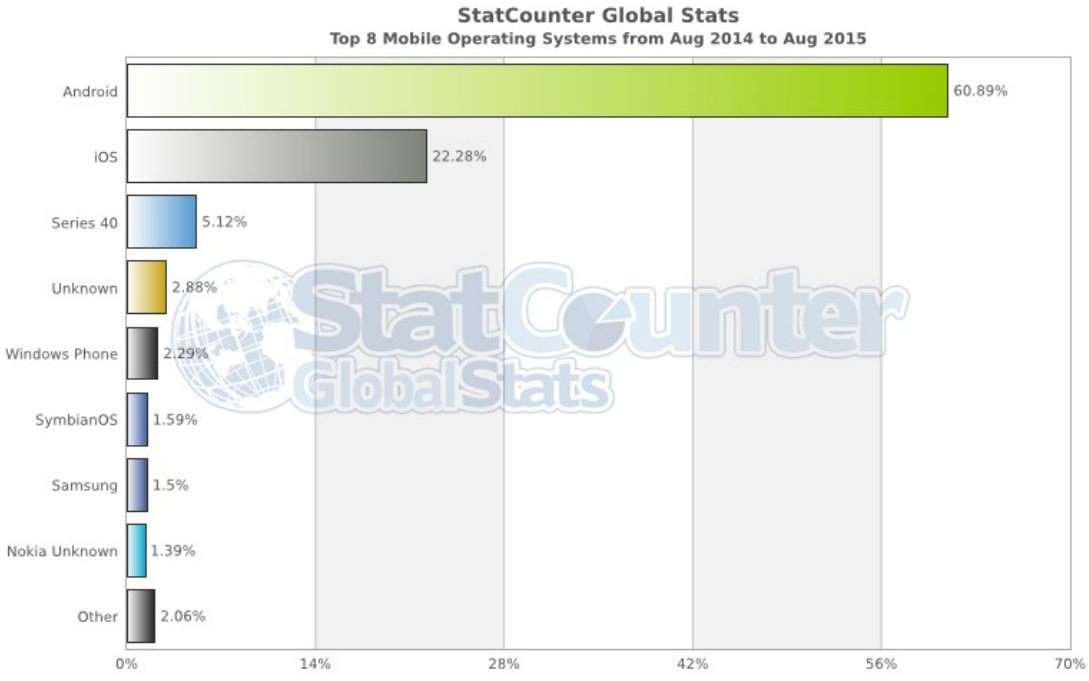

Figura 1 – Gráfico dos sistemas operacionais utilizados por *tablets* e aparelhos de telefonia móvel. Fonte: Stat Counter – Global Stats.

Atraídos pela necessidade em disponibilizar conteúdos e serviços aos consumidores, grandes empresas focam seus esforços de publicidade e marketing em aplicativos que sejam acessíveis ao público possuidor de *smarthphones*. Essa rápida evolução tem oferecido um campo muito atrativo para desenvolvedores de *software*, que podem oferecer soluções de forma ágil e eficientes para o mercado. Como esse mercado é voltado a consumidores que utilizam diferentes sistemas operacionais, surge a problemática de como atingir efetivamente o maior público possível com um solução robusta. Existem três opções para desenvolver um software voltado a aparelhos *smarthphones* e *tablets*, utilizando uma plataforma nativa, utilizando *frameworks Web* e HTML5, neste modelo, o aplicativo é executado pelo navegador desta forma o desenvolvedor deve ter em mente as diferenças entre cada navegador, ou utilizar tecnologias de desenvolvimento híbridas com *frameworks* como o Cordova.

Neste trabalho será apresentado o funcionamento do *framework* Cordova através de um estudo sobre sua arquitetura e testes comparativos entre o acesso de recursos nativos do sistema operacional Android. O recurso escolhido para este trabalho é o acesso a contatos do dispositivo. Desta forma serão comparados a chamada deste recurso através de aplicações construídas com o *Cordova* e *SDK Android*.

1.1 OBJETIVO GERAL

Este trabalho tem por objetivo a criação de aplicações com o *framework Cordova* e com o ambiente de desenvolvimento *Android*, com intuito de comparar o funcionamento do acesso a recurso de contatos de dispositivos *smartphones*.

1.2 OBJETIVOS ESPECÍFICOS

- Apresentar a arquitetura, modelo de programação e funcionamento do *framework* Cordova;
- Implementar aplicativo para acesso de recurso de contatos com o *framework Cordova*;

- Implementar aplicativo para acesso de recurso de contato com o SDK *Android*;
- Comparar a performance entre as chamadas dos aplicativos desenvolvidos;

1.3 JUSTIFICATIVA

Mais de 345 milhões de *smarphones* foram vendidos até o segundo trimestre de 2015 de acordo com *International Data Corporation* (IDC), onde 4 sistemas operacionais dominam a maior fatia do mercado. Atualmente a loja da Google contabiliza mais de 1,43 milhões de aplicações enquanto a *Apple Store* tem 1,21 milhões.

O modelo padrão de desenvolvimento de aplicativos para dispositivos móveis é a criação de uma aplicação nativa para cada SO (Sistema Operacional), porém isto tem um custo elevado de projeto (MARTIN FOWLER, 2012).

O crescimento do mercado de aplicações leva a necessidade de atender rapidamente o tempo de mercado. Assim como as oportunidades do mercado levaram ao surgimento de aplicações multi plataformas nos computadores pessoais nos anos 90, aplicativos móveis são mais frequentemente disponibilizados para múltiplas plataformas (ALLEN SARAH, 2010).

Em função disso faz-se necessário aprofundar a discussão a cerca do *Cordova*, um *framework* que possibilita a criação de aplicações multi plataformas com apenas um código fonte. Com tecnologias largamente utilizadas como HTML, *JavaScript* e CSS esse *framework* propõe um modelo de desenvolvimento que não há duplicação de esforços para suprir as diferentes plataformas móveis (SHOTTS KERRI, 2014).

1.4 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos:

- O primeiro capítulo apresenta a introdução, objetivos (geral e específicos) e justificativa do trabalho;

- O segundo capítulo aborda conceitos de desenvolvimento para dispositivos móveis, *frameworks* e plataformas (revisão de literatura).
- O terceiro capítulo trata sobre os materiais e métodos.
- O quarto capítulo trata sobre resultados e discussões finais

2 REFERENCIAL TEÓRICO

2.1 FRAMEWORKS

São soluções genéricas para problemas comuns no processo de desenvolvimento de *software*. É uma abstração que une códigos comuns entre vários problemas comuns. Através de configurações o *framework* atinge uma funcionalidade específica. Diferente de bibliotecas é capaz de ditar o fluxo da aplicação, através das configurações pré estabelecidas. Permite o reuso de uma arquitetura de aplicação pré definida e prove componentes que auxilia na criação de um *software (SOMMER, 2012)*.

A utilização de um *framework* enfatiza a reutilização de projetos em oposição a reutilização de código, desta forma compreende de um conjunto de classes implementadas em uma linguagem específica.

Neste contexto, os *frameworks* são divididos em dois grupos: verticais que são aplicados em domínios comuns entre diversos projetos, como persistência e interfaces gráficas. E horizontais, são chamados de especialistas, gerados a partir de um problema em domínio especifico, provê funcionalidades específicas de um domínio.

Existem diversos projetos de *frameworks* disponíveis, provendo ferramentas que auxiliam nas atividades do processo de desenvolvimento de *software*, é importante citar a contribuição das comunidades criadas em torno destes projetos, que contribuem compartilhando e disseminando informações, fomentando e expandindo mais a comunidade e consequentemente os produtos e serviços produzidos. Como consequência o desenvolvedor possui um vasto número de ferramentas para auxiliar na construção de aplicativos.

2.2 PLATAFORMAS MÓVEIS

Uma plataforma computacional consiste em uma coleção de recursos de *hardware*, como o processador, memória principal, módulos de entrada e saída, *timers*, *drives* de disco e

recursos de *software* que se comunica com o *hardware* e provê uma interface com o usuário (STALLINGS, 2012). Desta forma afirma-se que é uma estrutura composta por *softwares* e *hardwares*, com uma arquitetura preestabelecida.

As plataformas móveis são um conjunto de tecnologias que envolvem sistemas operacionais, linguagens de programação e ferramentas de desenvolvimento. Os aparelhos celulares necessitam de uma plataforma para funcionarem. Estas plataformas são responsáveis por gerenciar os recursos dos aparelhos como o banco de dados, câmera, GPS e áudio. Também são responsáveis pela interação com o usuário, através de aplicativos e recursos multimídia.

Durante a última década, 2005 em diante, dezenas de plataformas móveis surgiram, como já apresentada na Figura 1, mas não conseguiram manter-se no mercado por muito tempo. São poucas as plataformas que possuem uma parcela relevante no mercado mundial (VISION MOBILE, 2011).

2.2.1 ANDROID

Android é um sistema operacional móvel desenvolvido pela Google que roda sobre um núcleo Linux. O Android SDK fornece as ferramentas e API's (*Application Programming Interface*) necessárias para começar o desenvolvimento de aplicativos na plataforma *Android* usando a linguagem de programação Java. (ANDROID DEVELOPER, 2015).

Essa plataforma é resultado dos esforços da OHA (*Open Handset Alliance)* um grupo formado por empresas que atuam no mercado de telefonia de celulares, sendo liderados pela Google. Os principais integrantes do grupo são: HTC, LG, Motorola, Samsung, Sony Ericsson, Toshiba, Sprint Nextel, China Móbile, T-Mobile, ASUS, Intel e Garmin. O objetivo deste grupo é padronizar uma plataforma única e aberta para celulares e aumentar a satisfação dos consumidores em relação ao produto final. O objetivo secundário é criar e manter uma plataforma moderna e flexível para o desenvolvimento de aplicações corporativas (LECHETA, 2010).

Algumas das características do *Android* apontadas por ALLEN (2010) são: a capacidade de reproduzir gráficos 2D e 3D, através do *OpenGL ES*, suporte para arquivos de áudio, vídeo e imagem, ser multitarefa e possuir uma máquina virtual *JavaScript* responsável

por interpretar e executar *javascripts*. Por utilizar as linguagens de programação Java e C/C++ para o desenvolvimento de aplicativos, o *Android* abriu novas possibilidades para os desenvolvedores destas linguagens. A arquitetura de desenvolvimento proposta pelo *Android* é visualizada na figura 2.

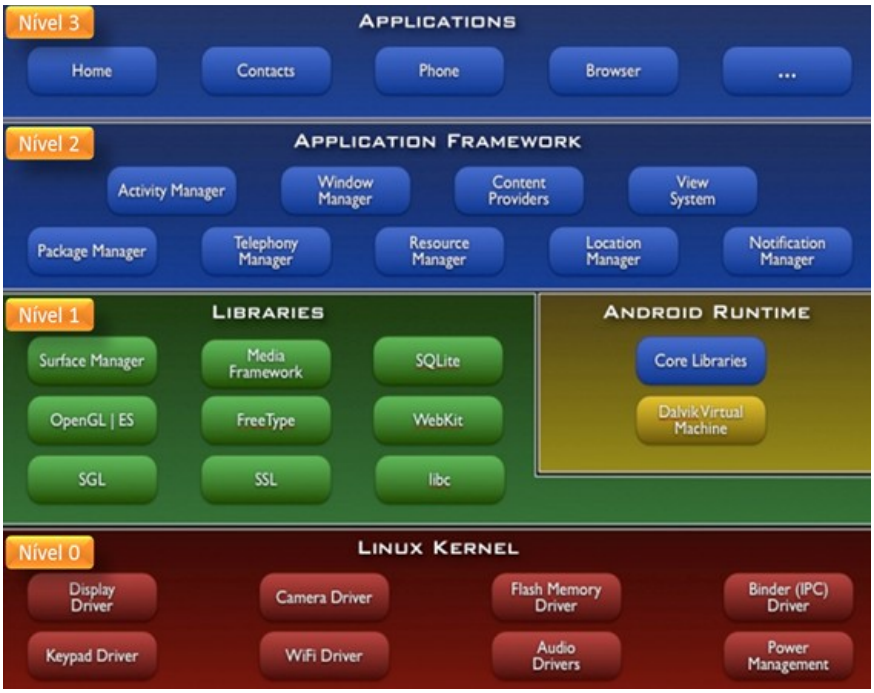

Figura 2 - Arquitetura Android.
Fonte: Android Developer, 2015

As definições de cada camada da arquitetura apresentada na figura 2 são descritas a seguir:

- Nível zero: *Linux Kernel* , encontraremos os programas de gerenciamento de memória, configurações de segurança e *drivers* de *hardware*.
- Nível um: bibliotecas e tempo de execução *Android*. A camada de biblioteca é um conjunto de instruções que dizem ao dispositivo como lidar com diferentes tipos de dados, incluindo um conjunto de biblioteca C/C++ usadas por diversos componentes do sistema e são expostas a desenvolvedores através da estrutura de aplicativo *Android*. A camada de tempo de execução inclui um conjunto de bibliotecas do núcleo Java. Para desenvolver aplicações para o *Android*, os programadores utilizam a linguagem de programação Java, nesta camada encontraremos a Máquina Virtual

Dalvik (DVM). O *Android* usa esta máquina virtual para rodar cada aplicação com seu próprio processo. Isso é importante por algumas razões: nenhuma aplicação é dependente de outra e se uma aplicação parar, ela não afeta quaisquer outras aplicações rodando no dispositivo e isso simplifica o gerenciamento de memória, pois a máquina virtual está baseada em registradores e desenvolvida de forma otimizada para requerer pouca memória e permitir que múltiplas instâncias executem ao mesmo tempo (BURNETTE, et All, 2010).

- Nível dois: Camada de *framework* de aplicação, programas que gerenciam as aplicações básicas do telefone. Os desenvolvedores têm acesso total ao *framework* como um conjunto de ferramentas básicas com o qual poderá construir ferramentas mais complexas.
- Nível três: Camada de aplicações e as funções básicas do dispositivo. Esta é a camada de interação entre o usuário e o dispositivo móvel, nela encontramos aplicativos cliente de e-mail, programa de SMS, calendário, mapas, navegador, contatos entre outros.

2.2.1.1 ANDROID SDK

É o conjunto de ferramentas utilizado para desenvolver aplicações para o sistema operacional *Android*, oferece um emulador para simular aparelhos celulares, ferramentas utilitárias e uma API para a linguagem Java, com todas as classes necessárias para desenvolver aplicações (LECHETA, 2009).

O SDK é gratuito sendo possível realizar o seu *download* no portal de desenvolvimento do *Android*. O SDK oferece a possibilidade de utilizar a IDE (*Integrated Development Enviroment*) *Android Studio* desenvolvida no IntelliJ IDEA, além de oferecer um editor de código permite o gerenciamento de emuladores com formatos e configurações comuns de dispositivos do mercado. Por meio do gerenciador de pacotes incluído no SDK, é possível escolher as diferentes versões disponíveis do Android e outros pacotes adicionais contendo exemplos de aplicativos, ferramentas para *debug,* pacotes de suporte a desenvolvimento de aplicações para diferentes modelos de processadores.

Este modelo de desenvolvimento seria o de construções de aplicações nativas, ou seja, seria impossível desenvolver uma aplicação neste modelo que seja portável para outro sistema operacional como iOS ou *Windows Phone*.

2.3 TECNOLOGIAS WEB

Desde o surgimento da Internet, seu principal objetivo é a publicação e troca de informações. Esta proposta é pregada por meio da navegação em documentos virtuais dispostos na rede de forma descentralizada. A Internet priorizava acesso à publicações científicas. Com o passar do tempo, este cenário foi mudando e hoje percebe-se que a Internet hospeda além de publicações científicas, guias, notícias, tutoriais, imagens, vídeos, serviços, entre outros.

De acordo com o W3C (*World Wide Web Consortium*) a *Web* é baseada em 3 pilares

- Um esquema de nomes para localização de fontes de informação na *Web*, esse esquema chama-se URI.
- Um protocolo de acesso para acessar estas fontes, o HTTP.
- Uma linguagem de Hipertexto, para a fácil navegação entre as fontes de informação: o HTML.

Na camada do cliente encontra-se o usuário final, através de um navegador é possível acessar *sites* e aplicações *Web* fazendo requisições HTTP (*HyperText Transfer Protocol*). O protocolo HTTP especifica as mensagens que os clientes podem enviar aos servidores e que respostas receberão. Cada interação consiste em uma solicitação e uma resposta desta solicitação, esta troca de informação seguem especificações pré-definidas (TANENBAUM, 2003). Conforme a figura 3 o servidor recebe estas requisições, processa e retorna o resultado para o cliente, desta forma tornando possível a troca de conteúdos.

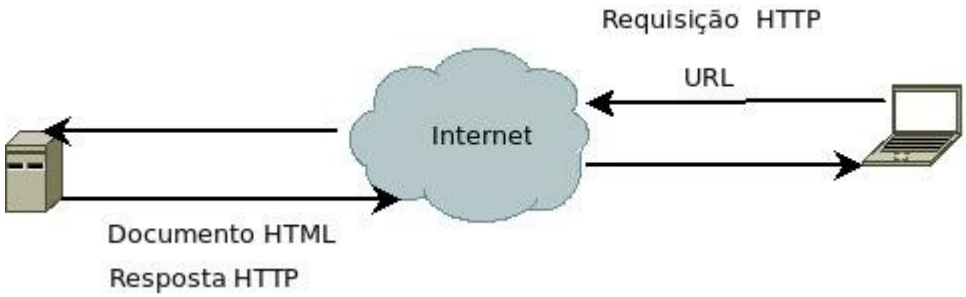

Figura 3 - Requisição e resposta protocolo HTTP.

Também fazem parte da Internet diversas linguagens de programação. Estas se dividem basicamente em duas categorias: linguagens *client-side* e *server-side.* Linguagens *client-side* são executadas do lado do cliente, através do navegador, geralmente são utilizadas para obter uma melhor experiência do usuário, fornecendo recursos que melhoram a usabilidade e navegação, validação de formulário e outras funcionalidades. JavaScript é a linguagem *client-side* mais utilizada para criação de conteúdos dinâmicos e ricos em recursos (HAVERBEKE, 2014). Já as linguagens *server-side* são executadas no servidor e fornecem a lógica principal da aplicação, processando dados mais complexos e persistindo informações em banco de dados.

A interface gráfica é responsável pela comunicação direta com o usuário final, fornecendo ferramentas como formulários, tabelas e gráficos que permitem a interação e manipulação das informações por parte do usuário. Toda interface gráfica é criada pelo navegador, ou seja, no lado do cliente. As páginas de Internet são feitas utilizando HTML, para estruturar e armazenar o conteúdo, e CSS (*Cascading Style Sheets*), para estilização visual das páginas. Essas linguagens e seus padrões de implementações permitem a criação de *frameworks* e bibliotecas que aumentam significativamente a velocidade de desenvolvimento para *Web* (ALLEN, 2010).

2.3.1 HTML

As páginas da *Web* são escritas em uma linguagem denominada HTML (HyperText Markup Language). O HTML permite que os usuários produzam páginas da Web que incluem

texto, gráficos e ponteiros para outras páginas da Web. A HTML é uma linguagem de marcação, ou seja, uma linguagem para descrever como os documentos devem ser formatados. O termo "marcação" surgiu devido ao modelo de trabalho de editores de jornais e revistas impressos onde realmente marcavam-se os documentos para informar ao impressor (uma pessoa) que fontes usar, espaçamento entre linhas, realce de palavras e outras informações de formatação necessárias. Portanto, estas linguagens contêm comandos explícitos de formatação. Por exemplo, em HTML, o significa início do modo negrito**,** e significa fim do modo negrito. A vantagem deste tipo de linguagem sobre outra sem marcação explícita é a maior facilidade para criar um navegador destinado a interpretar apenas estas marcações (TANENBAUM, 2008).

O HTML está presente em toda página de Internet, ele é a estrutura básica da mesma. Derivado do SGML (*Standard Generalized Markup Language*) o HTML foi criado no início da década de 1990 com o objetivo de tornar possível o acesso e troca de informações e pesquisas entre universidades, em seu surgimento a principal inovação foi a implementação da *tag* <a>, permitindo a referencia de links entre os documentos.

Desde 1999, o desenvolvimento da linguagem HTML ficou estacionado na versão 4. De lá pra cá, a W3C esteve focada em linguagens como XML e SVG. Enquanto isso, os navegadores estiveram preocupados em desenvolver suas funcionalidades, como exibir páginas em abas e oferecer a integração com leitores de RSS (SARTI, 2009). 19 Em dezembro de 1999, foi publicada a versão 4.01 do HTML como uma recomendação do W3C. E só em janeiro de 2008 o HTML5 foi publicado como um novo projeto do W3C.

Em sua última versão, o HTML5, é uma especificação avançada em termos de simplicidade, extensibilidade e foco na aplicação (KOSMACZEWSLI 2012). Fornece recursos para o uso do CSS e JavaScript, novas *tags* foram incluídas e algumas tiveram suas funções modificadas. Através de suas novas API é possível manipular as características destes elementos de forma que o *Web* site, ou a aplicação, continue leve e funcional. *Canvas*, *localstorage*, geolocalização e *WebGL* são alguns dos componentes que acompanham o HTML5. Outra funcionalidade inserida é a possibilidade de tornar o conteúdo das páginas mais semântico, permitindo aos motores de busca, compreenderem este conteúdo e o indexarem corretamente. As versões anteriores ao HTML5 não possuíam um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, *sidebar* e menu, também não havia um padrão de nomenclatura de IDs, classes ou *tags*. Com o uso das novas

tags *header*, *footer*, *section*, *aside*, *nav* e *article* é possível tornar o código HTML mais semântico e "legível" aos mecanismos de busca, permitindo que a indexação seja feita de uma forma mais inteligente e efetiva (PILGRIM, 2010).

No Figura 4 é possível verificar algumas mudanças entre as versões 4 e 5 do HTML.

Nome e descrição do elemento	HTML 4.01	HTML 5
DOCTYPE: Declarar ao navegador qual tipo de documento será mostrado e quais regras deverão ser utilizadas.	`<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd >`	`<!DOCTYPE html>`
METATAGS: marcação utilizada para descrever o contexto da página e sua codificação	`<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />`	`<meta charset="UTF-8" />`
SCRIPT: utilizar arquivos de scripts externos	`<script type="text/javascript" src="file.js"></script>`	`<script src="file.js"></script>`
LINK: utilizar arquivos externos	`<link rel="stylesheet" type="text/css" href="file.css" />`	`<link rel="stylesheet" href="file.css" />`
CANVAS: renderizar conteúdo gráfico	Não consta	`<canvas> ... </canvas>`
AUDIO: executar conteúdo de áudio	Não consta	`<audio src="file.mp3" controls></audio>`
VIDEO: executar conteúdo de vídeo	Não consta	`<video src="file.ogg" controls></video>`
HEADER: elemento responsável por definir cabeçalhos de conteúdo no contexto da página	Não consta	`<header> ... </header>`
ARTICLE: elemento responsável por definir artigos de conteúdo no contexto da página	Não consta	`<article> ... </article>`
SECTION: define uma seção de conteúdo	Não consta	`<section> ... </section>`

Figura 4 - Quadro comparativo entre versões 4 e 5 do HTML.
FONTE: CARVALHO, 2011

2.3.2 CSS

O CSS (*Cascading Style Sheets*) é uma linguagem de estilização responsável por definir a apresentação visual de um documento HTML, adicionando elementos como cores, sombras e animações (STARK, 2010). O CSS possui seletores que estão associados às *tags* do HTML. Por meio dos seletores é possível referenciar trechos específicos do documento HTML, estilizando-os individualmente ou coletivamente. O CSS possui seletores pré-definidos como *div*, *span*, *table* e p, também é possível criar novos seletores, estes são conhecidos como *id* e *class*. Os seletores id devem ser utilizados por elementos HTML que

não se repetem na página, ou seja, elementos únicos. Os seletores *class* podem ser atribuídos a mais de um elemento HTML (W3C , 2015).

O CSS possui propriedades que são pré-definidas e não podem ser adicionadas e/ou removidas. Cada propriedade espera um valor apropriado, algumas suportam diversos valores e formatos.

O CSS3 é a especificação mais nova da linguagem. Se comparado a versão anterior, CSS 2.1, o CSS3 adicionou diversas propriedades que facilitam a estilização de *Web* sites, tornando o conteúdo muito mais rico em recursos visuais. O CSS3 implementou os perfis (profiles) para plataforma mobile, TVs e impressões, estes permitem definir estilos individuais para cada perfil. Novas propriedades também foram adicionadas, dentre elas destacam-se *border-radius*, *box-shadow*, *opacity*, *transform*, *transition*, *resize*, *@font-face*, dentre outras.

2.3.3 JAVASCRIPT

É uma linguagem de programação baseada em scripts e padronizada pela ECMA *International* (associação especializada na padronização de sistemas de informação). Foi criada por Brendan Eich (Netscape) e surgiu em 1995 como linguagem de script *client-side* de páginas *Web*. Inicialmente introduzida em navegadores para permitir a execução de scripts sem a necessidade de passar pelo servidor, controlando o navegador, realizando comunicação assincronia e alterando o conteúdo do documento exibido (HARVEBERKE, 2014).

Com esta finalidade se popularizou em inúmeros navegadores como, Internet Explorer, Firefox, Chrome, Safari. Tornando-se padrão para muitos deles. Atualmente é utilizado também para criar aplicações *server-side*. Algumas das API que elevaram o status do JavaScript tornando-o essencial no dias atuais são o AJAX, JQuery, ExtJS, Node.js, GWT e MooTools. Ferramentas como o FireBug e JSLint também ajudaram em sua popularização.

2.4 APLICATIVOS MÓVEIS MULTIPLATAFORMA

Diferentes linguagens de programação, ferramentas de desenvolvimento e API's impedem a interoperabilidade e o reuso do código entre as plataformas. Aplicativos para Android são feitos utilizando Java, no iOS é o Objective-C, no Windows Phone utilizasse a plataforma .NET (C# e VB.NET são as principais linguagens). Algumas plataformas, como iOS e Android, permitem o uso do C/C++ para o desenvolvimento de aplicativos, mas cada uma possui API's distintas que impossibilitam o reuso de boa parte do código. Para desenvolver uma aplicação nativa que rode em diversas plataformas seria necessário criar uma aplicação para cada plataforma, isso exige mais investimento, tempo e uma equipe com bastante experiência nas plataformas para qual o aplicativo será desenvolvido. Conforme afirma KESSIN, 2008, a internet é multiplataforma, é possível escrever páginas que irão funcionar no Windows, Mac OS X, Linux, iPhone/iPad, Android e diversas outras plataformas. Toda plataforma *mobile* possui navegadores que são capazes de interpretar códigos JavaScript e reproduzir documentos HTML estilizados com CSS.

Decorrente da enorme diversificação das características dos dispositivos móveis, que envolvem aspectos como variações nos tamanhos de tela e diferentes sistemas operacionais, bem como versões, surge a necessidade do desenvolvimento de aplicações multiplataforma, que atualmente é bastante discutido e aconselhável por diversas organizações, tais como a Gartner, a W3C. Para auxiliar aos desenvolvedores, empresas como a Intel disponibilizam ambientes integrados para a distribuição dos aplicativos. Um exemplo é o Intel XDK que possibilita o desenvolvimento de aplicações em HTML5 onde irá atender aos quesitos de responsividade, que se refere ao fato de projetar uma interface adaptativa a diversos tamanhos de telas, e gerar um executável para diversas plataformas como: iOS, *Android*, *Windows Phone*, dentre outros (CORRAL *et all*, 2011).

O desenvolvimento de aplicações móveis envolve diversos aspectos que diferem das aplicações *desktops*, tais diferenças inferem diretamente no sucesso e no fracasso dos aplicativos em seus números de usuários e na satisfação dos mesmos. É preciso uma análise profunda do usuário, no seu perfil e nas suas possíveis limitações, sendo esses os fatores em que se devem ter como premissas na elaboração do escopo das aplicações, fazendo com que tal ideia inovadora, disponibilizada no ambiente móvel, possa atender aos requisitos de

usabilidade e acessibilidade, fazendo assim de extrema importância a definição do público alvo para com aquela solução pretende atingir. O uso de *smartphones* estende-se a diversos lugares e situações, trazendo assim muitos desafios na elaboração dos aplicativos, tais situações envolvem o uso do *smartphone* com apenas uma mão, a luminosidade em determinados lugares, sons podem desagradar ou atrapalhar os usuários de escutarem músicas, o aplicativo requerer acesso à internet, dentre outros aspectos que fazem com que tornem os *apps* de sucesso ou não no número de usuários.

A experiência do usuário deve ser reaproveitada por meio do uso comum dos componentes da interface que envolve as funcionalidades da solução, facilitando a familiaridade da mesma, como por exemplo, podemos citar o facebook, em que é facilmente identificado pelo usuário os componentes que envolvem a todas as funcionalidades básicas e mais utilizadas, em comparação com a página quando acessada pelo *desktop*.

2.5 FRAMEWORKS PARA APLICATIVOS MÓVEIS MULTIPLATAFORMA

Com o desenvolvimento de tecnologias multiplataformas, diversos *frameworks* surgiram no mercado para auxiliar o desenvolvedor. Segue uma descrição de alguns dos principais *frameworks* do mercado.

- **Cordova/PhoneGap**: Permite a criação de aplicações híbridas para as plataformas iOS, Android, Blackberry, Symbian, Bada e Windows Phone. Através do componente *webview* nativo de cada plataforma, o *PhoneGap* consegue executar a aplicação que foi desenvolvida utilizando tecnologias *Web*. Ele oferece diversas API *Java*S*cript* que possibilitam o uso de recursos nativos dos dispositivos, como bússolas escrita e leitura de arquivos na unidade de armazenamento e geolocalização;
- **Ionic**: Assim como o *PhoneGap* fornece suporte ao desenvolvimento de aplicações híbridas com as mesmas características, faz uso por padrão do *framework* Angular, através do ngCordova, que fornece suporte a criação de aplicações *JavaScript*;
- **Sencha Touch**: Baseado nas bibliotecas JavaScript ExtJS, jQTouch e Raphaël, o Sencha Touch é um dos *frameworks* mais utilizados. Permite a criação de *webapps* para as plataformas Android, iOS e Blackberry, possui sua própria IDE, o Sencha Architect, e a extensão Sencha Animator para criação de animações utilizando CSS3;

- **Mono**: MonoTouch e Mono for Android são dois produtos da empresa Xamarim. Estas duas ferramentas possibilitam a criação de native *apps* para as plataformas iOS e Android utilizando a plataforma .NET, mais precisamente a linguagem C#;
- **Titanium Mobile**: Criado pela Appcelerator e com sua própria IDE, o Titanium Studio, o Titanium Mobile possibilita a criação de *webapps* e aplicações híbridas para as plataformas *iOS, Android e Blackberry.* Além de utilizar *JavaScript* como linguagem principal, também permite a utilização de PHP, Python e Ruby no desenvolvimento dos aplicativos. Também possui suas próprias API *JavaScript* que possibilitam o uso de recursos nativos que as plataformas oferecem;
- **Adobe AIR**: É a versão *desktop* do Adobe Flash Player, que passou a ser suportada pelas plataformas mobile iOS, Android e Blackberry. O Adobe Flex é um *framework* que permite a criação de aplicativos AIR utilizando *Action Script* e MXML;

2.6 CORDOVA

Inicialmente criado pela empresa Nitobi Software em 2006 o *framework* sofreu uma constante evolução porém seu grande salto foi em outubro de 2011 quando a empresa Adobe comprou a Nitobi tendo controle sobre o projeto *PhoneGap*, com intuito de garantir uma padronização do código, governança sobre a documentação e entendimento dos conceitos, foi doado para incubação para a *Apache Software Foundation*(ASF). Sob a licença da ASF houve um grande crescimento da comunidade por trás do projeto, trazendo uma evolução continua do *framework*. Atualmente o *PhoneGap* é uma distribuição do *Apache Cordova*, fazendo uso de toda sua infraestrutura de *plugins* e ferramentas desenvolvidas pela comunidade ativa do projeto *Cordova* (WARGO, 2015).

Cordova é *framework* de código livre que permite o desenvolvimento de aplicações móveis utilizando APIs *Web* padronizadas para uma plataforma alvo desejada. Ela acessa funções nativas de dispositivos móveis como, acelerômetro, câmera, geolocalização e permite que a aplicação seja desenvolvida sem utilização de uma API específica, resultando em aplicações híbridas compatíveis com plataformas como Android, iOS e Windows Phone.

São chamados de híbridos pois não são aplicativos nativos da plataforma e nem puramente uma aplicação *Web*. Ela faz uso de ambos recursos, utiliza tecnologias *Web* para interação com o usuário e através do *JavaScript* faz uso de recursos nativos da plataforma.

A Figura 5 mostra quais API's nativas são possíveis manipular através do *Cordova*. Para cada API (linha) são marcadas as plataformas que a suportam.

	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	✓	✓	✓	✓	✓	✓	✓	✓

Figura 5 - APIs disponíveis por plataforma.
Fonte:CORDOVA

2.6.1 Plugins

Para cada API nativa como acelerômetro e câmera, são construídos *plugins*. Estes *plugins* podem ser adicionados e removidos conforme as necessidades, citaremos os principais (APACHE CORDOVA).

2.6.1.1 Acelerômetro

O acelerômetro é um dispositivo que mede as forças de aceleração e alteração de sua orientação. Essas forças podem ser tanto estáticas, sendo afetadas pela gravidade, quanto dinâmicas ou geradas pela aceleração através de movimentos manuais no aparelho. São monitorados os eixos X, Y e Z, e de acordo com os valores de cada um é possível saber em qual posição o *smartphone* está.

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 vertical (com base no eixo Y) para horizontal, (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.

O cordova-plugin-device-motion oferece acesso ao acelerômetro através do objeto navigator.accelerometer. Este objeto oferece 3 métodos para interação com o dispositivo nativo.

- navigator.accelerometer.getCurrentAcceleration;
- navigator.accelerometer.watchAcceleration;
- navigator.acceleroeter.clearWatch;

2.6.1.2 Câmera

As aplicações móveis demandam do recurso da câmera para fornecer recursos como gravar e transmitir imagens, ou mesmo utilizar imagens para detecção de padrões e URL's como o QRCode. O Cordova oferece o *plugin* cordova-plugin-camera e através do objeto navigator.camera fornece acesso aos métodos.

- camera.getPicture;
- camera.cleanup;

Através desta API pode-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. Possui duas funções, tirar uma foto, ou buscar uma imagem da galeria. Sua função depende da configuração do parâmetro *SourceType*. Conforme sua configuração ele abrirá o aplicativo nativo do aparelho responsável por tirar fotos após a foto tirada o aplicativo nativo fecha retornando para a tela da aplicação *Cordova*. A outra forma é buscar todas imagens e álbuns disponíveis no aparelho, incluindo os álbuns criados por outras aplicações. A qualidade das fotos tiradas depende exclusivamente do hardware e todas as imagens exibidas pela aplicação serão apresentadas na melhor qualidade possível.

Existem alguns *plugins* criados pela comunidade para reconhecimento de faces, identificando padrões de olhos e boca de uma imagem.

2.6.1.3 Contatos

Com o objeto navigator.contacts encontrado no *plugin* cordova.plugin.contacts, é possível acessar a lista de contatos existentes. A lista de contatos é uma informação muito sensível, por questões de privacidade sempre é notificado ao usuário quando uma aplicação acessa este tipo de informação, isto quando a politica da aplicação nativa não o faça.

2.6.1.4 Arquivos

Para leitura e escrita de arquivos no dispositivo, o *Cordova* disponibiliza o *plugin* org.apache.cordova.file baseado na API de tratamento de arquivos do HTML5, o que facilita o trabalho dos desenvolvedores *Web*. Nesse recurso existe uma particularidade entre os sistemas que não pode ser evitado, visto que o comportamento das pastas e 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. Essa API também pode ser utilizada como uma opção para armazenamento de dados no lado do cliente.

2.6.1.5 Geolocalização

Prove informações como latitude e longitude referente a localização do aparelho. Fontes comuns de localização incluem *Global Positining System*(GSP) e localização inferida através de sinais de rede como endereço IP, RFID, endereço MAC WiFi e Blutooth e códigos de célula GSM/CDMA.

Essa API é baseada na API de geolocalização da W3C. O *plugin* 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.

2.6.1.6 Persistência

O *Cordova* fornece opções de banco de dados baseadas em duas especificações da W3C.

- **Web Storage API Specification**: permite acesso de forma simplificada passando apenas pares de chave e valor.
- **Web Sql Database Specification**: possui mais funcionalidades permitindo acesso via SQL queries.

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.

3 MATERIAIS E MÉTODOS

Neste capítulo estão descritos os métodos e materiais utilizados para o desenvolvimento das aplicações que servem de base comparativa para o acesso do recurso nativo de contatos da plataforma Android, para esta comparação foram desenvolvidas aplicações utilizando o *framework Cordova* e o SDK *Android*.

A metodologia utilizada para o desenvolvimento do trabalho envolveu primeiramente uma revisão bibliográfica das linguagens e tecnologias aplicadas no desenvolvimento do estudo experimental. Observa-se que para o sucesso do trabalho é uma etapa importante, pois segundo SILVA E MENEZES (2001), "a revisão de literatura é fundamental, porque fornecerá elementos para evitar a duplicação de pesquisas sobre o mesmo enfoque do tema, e favorecerá a definição de contornos mais precisos do problema a ser estudado".

A pesquisa das tecnologias e linguagens foi de natureza aplicada e com objetivo exploratório, sendo a que mais se aplicava, pois segundo SILVA E MENEZES (2001), a pesquisa aplicada "objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos", e a pesquisa exploratória "visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses, envolve levantamento bibliográfico".

3.1 FERRAMENTAS

Para o desenvolvimento e testes das aplicações foram usadas as seguintes ferramentas:

- Cordova 5.0.0: *framework* para acesso a funcionalidades de plataformas nativas;
- Android SDK: API para desenvolvimento de aplicações Android Nativas;
- IDE Brackets: codificação JavaScript e HTML;
- IDE Studio Android: codificação Java e XML;
- Ambiente de desenvolvimento Ubuntu Linux 12.04
- Ambiente de testes *smartphone* Samsung Galaxy S4 mini;
- AppInstaller: Aplicativo *Android* para instalação de novas apk's;

3.2 ACESSO A RECURSOS COM CORDOVA

Para avaliar o *Cordova* como *framework* para desenvolvimento de aplicações híbridas foi desenvolvido uma aplicação utilizando o *plugin-cordova-contacs* e mensurando o tempo gasto para acesso e exibição dos contatos para o usuário.

3.2.1 Configuração do ambiente

Como ponto inicial da configuração do ambiente foi necessário a instalação de uma SDK *Android*, pois este será utilizado para a criação de emuladores e sua API se faz necessária no momento da criação de aplicações utilizando o *Cordova.* Isto devido ao fato de serem utilizadas algumas bibliotecas do SDK para a geração do arquivo contendo a instalação da aplicação, o artefato gerado é um arquivo com extensão *apk*, este arquivo é utilizado para realizar a instalação da aplicação no dispositivo alvo.

Após isto foi necessário a instalação da ferramenta *nodeJs*, para que as outras aplicações escritas na linguagem JavaScript pudessem ser utilizadas. Através da central de pacotes Ubuntu ela foi facilmente instalada sem maiores necessidades de configurações. Em ambiente Windows é possível a instalação baixando os pacotes do site do produto.

Em seguida foi instalado um gerenciador de pacotes NPM, responsável pela instalação e configuração do *framework Cordova*. O NPM oferece um *promt* de comando o qual permite a instalação de pacotes disponíveis em seu repositório. Com isto pode ser feita a instalação do *framework Cordova* através do comando *npm cordova install*.

Para que a execução do *Cordova* por linha de comando fosse possível, foi necessária a configuração de algumas variáveis de ambiente, desta forma os diretórios contidos no *Android* SDK, *tools* e *plartorm-tools* foram adicionadas no PATH do sistema operacional. Vale ressaltar que uma JDK (*Java Development Kit*) deve estar configurada no ambiente.

3.2.2 Codificação

Após ambiente configurado é possível a execução de uma série de comandos utilizando o *Cordova* conforme demonstrados nos próximos passos. Para criação de projetos

foi invocado o comando: *cordova create APP_NAME.* Com isso é gerado um novo diretório com o nome informado no APP_NAME, neste diretório encontra-se todos arquivos básicos para a execução de uma aplicação *Web* porém ainda não existe a estrutura de arquivos necessárias para execução de uma aplicação na plataforma Android. Para que isto fosse possível foi necessária a adição desta plataforma alvo com o comando *cordova plarform add android.*

Para evitar que todos os testes da aplicação fossem executados no emulador foi adicionada a plataforma *browser,* possibilitando a execução da aplicação diretamente no navegador do ambiente de desenvolvimento, sem que seja necessária carregar a aplicação em um emulador android, para isto foi executado o comando *cordova platform add browser.*

O *cordova-plugin-contacs* é utilizado para a criação e consulta de contatos. Ele trabalha com um objeto padrão *Contact*. Os métodos expostos serão abordados na sequência.

O método *contacts.find* é utilizado para consultar os contatos de um dispositivo, o retorno deste método em caso de sucesso na consulta é uma lista de objetos *Contact*, contendo os campos desejados e informados por argumento, segue abaixo a descrição de cada argumento deste método:

- contactFields : Informa quais campos deverão ser retornados pela consulta, o objeto de retorno será um *Contact* contendo apenas os campos passados neste argumento, este argumento é obrigatório;
- conctacSuccess: função de *calback* que será invocada se a chamada for executada com sucesso, através deste *callback*, será retornada uma lista de contatos com os resultados que satisfizeram os filtros, este argumento é obrigatório;
- contactError: funcção de *callback* que será invocada caso a chamada gera algum erro, este argumento não é obrigatório;
- contactFindOptions: informações dos filtros que serão aplicados na consulta dos contatos, este argumento não e obrigatório;

O método *contacts.create()* , retorna um objeto Contact, ele pode ou não receber um objeto como argumento, este objeto deve seguir a padronização do objeto Contact, para que assim o método possa retornar um objeto populado com as propriedades passadas no argumento da invocação. A chamada deste método não fará com que o contato seja salvo, para

que isso ocorra será necessário invocar o método *save()* que estará exposto no objeto Contact retornado pela invocação do *contacts.create()*.

Para que fosse possível uma comparação entre as chamadas *Cordova* e *Android* nativa, foi informado o tempo em milissegundos antes da consulta e após a exibição dos contatos pela aplicação. Na figura 6 temos mais detalhes do código.

```
findContact : function() {
    var that = this,
        //cria um objeto para setar as opções
        //objeto padrão do cordova para este plugin.
        options = new ContactFindOptions(),
        // Quais campos queremos que sejam populados no objeto de retorno
        fields = ["displayName"];

    // filter=* para retornar todos registros
    // multiple=true para retormar uma lista de registros
    options.filter = "*";
    options.multiple = true;
    //exibe timestamp inicio
    that.setIni(new Date().toJSON());
    navigator.contacts.find(fields, function() {
        that.onFindSuccess.apply(that, arguments);
        //exibe timestamp fim
        that.setEnd(new Date().toJSON());
    }, function() {
        that.onFindFailure.apply(that, arguments);
    }, options);
},
```

Figura 6 - Trecho de código onde é feita a consulta de todos contatos do dispositivo utilizando Cordova.

3.3 ACESSO A RECURSOS COM ANDROID SDK

Para avaliar o *Android* SDK para desenvolvimento de aplicações móveis nativas foi desenvolvido uma aplicação que consulta os contatos cadastrados no dispositivo, medindo o tempo gasto para acessar a informação e exibi-la para o usuário.

3.3.1 Configuração do ambiente

Com o SDK Android já configurado para o desenvolvimento da aplicação em *Cordova*, não foi necessário realizar novas configurações. A aplicação foi criada utilizando o *Android Studio*, o qual se encarrega de criar toda estrutura do projeto.

3.3.2 Codificação

A API do *Android* possui a classe ContentResolver que gerencia o acesso a um conjunto estruturado de dados, encapsula os dados e fornece mecanismos para definir a segurança. Desta forma prove acesso a conteúdos do dispositivo como informações dos contatos cadastrados. Utilizado em conjunto com constantes declaradas na própria API, é possível informar qual tipo de conteúdo está sendo solicitado.

Para acessar as informações dos contatos do dispositivo, utilizamos a constante definida na API, *ContactsContract.Contacts.CONTENT_URI,* passando ela como argumento para o *ContentResolver* conforme demonstra a figura 7.

```
public void fetchContacts() {
    Integer cont=0;
    String phoneNumber = null;
    String email = null;
    //Define qual sera conteudo acessado
    //Utilizamos a constante declara na API para acesso a contatos
    Uri CONTENT_URI = ContactsContract.Contacts.CONTENT_URI;
    //Definimos as constantes de acesso as propriedades
    String _ID = ContactsContract.Contacts._ID;
    String DISPLAY_NAME = ContactsContract.Contacts.DISPLAY_NAME;
    //Recebera o resultado dos contatos retornados
    StringBuffer output = new StringBuffer();
    // Recupera a instancia do ContentResolver
    ContentResolver contentResolver = getContentResolver();
    //Recupera o timestamp para inicio da busca de contatos
    iniTime.setText(new Date().getTime()+"");
    //Executa a consulta de conteúdos utilizando a constante para acesso a contatos
    Cursor cursor = contentResolver.query(CONTENT_URI, null, null, null, null);


    if (cursor.getCount() > 0) {
        //itera o cursor com os resultados
        while (cursor.moveToNext()) {
            cont++;
            String contact_id = cursor.getString(cursor.getColumnIndex( _ID ));
            String name = cursor.getString(cursor.getColumnIndex(DISPLAY_NAME));
            output.append("Id:"+ contact_id+ " - Nome:" + name);
            output.append("\n");
        }
        //popula o objeto TextVIew para exibir ao usuário
        outputText.setText(output);
    }
    //recupera o timestamp para marcar o fim da execução
    endTime.setText(new Date().getTime()+"");
    total.setText(cont+"");
}
```

Figura 7 - Trecho de código onde é feita a consulta de todos contatos do dispositivo utilizando API Android.

4 RESULTADOS E DISCUSSÕES

4.1 RESULTADOS

Nesta seção são apresentados os resultados obtidos com as aplicações desenvolvidas.

4.2 RESULTADOS CORDOVA

Como base comparativa deste trabalho, a aplicação desenvolvida foi distribuída e instalada através do artefato gerado pelo *build* utilizando o comando *cordova build android*.

Este artefato possui 1,76MB o qual foi copiado através de uma conexão USB para *smartphone*. Para realizar sua instalação foi utilizada a aplicação AppInstaller disponível na loja de aplicativos da Android, A instalação foi executada sem problemas.

A figura 8 mostra o resultado obtido da consulta de todos contatos do dispositivo.

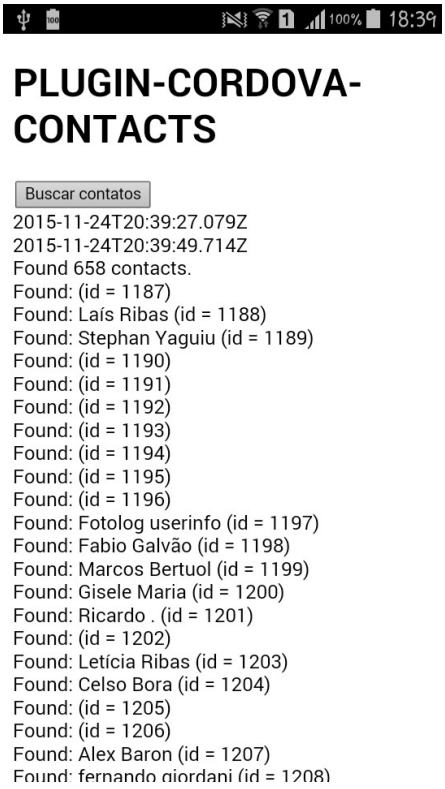

Figura 8 - Resultado consulta todos contatos Cordova.

Para uma maior precisão no tempo gasto nesta consulta, foram realizadas 5 execuções, e o tempo médio foi de 23 segundos e 235 milésimos de segundo para a consulta de 658 contatos.

4.3 RESULTADOS ANDROID

A aplicação desenvolvida foi distribuída e instalada através do artefato gerado pelo *build* realizado no *Android Studio*. O artefato gerado possui 245KB o qual foi copiado através de uma conexão USB para *smartphone*. A instalação seguiu o mesmo procedimento da aplicação desenvolvida com *Cordova*. A figura 9 mostra o resultado da execução da aplicação em um *smartphone*.

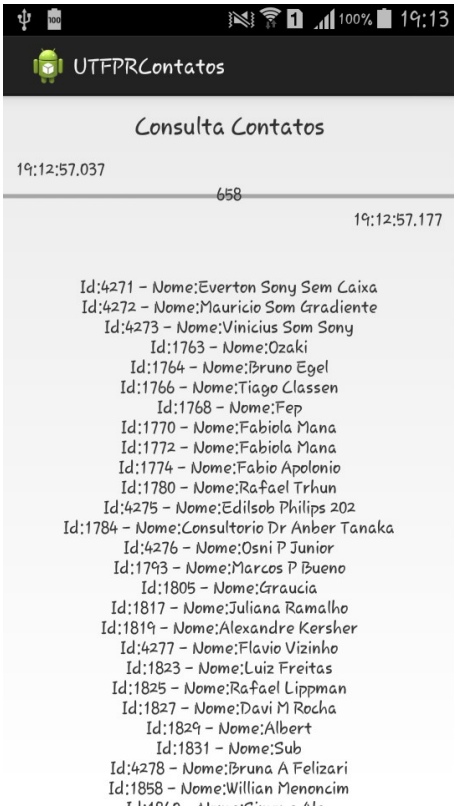

Figura 9 - Resultado consulta todos contatos Android.

Conforme feito com a aplicação desenvolvida com o *Cordova*, foram feitos 5 execuções e o tempo médio gasto pela aplicação nativa Android foi de 150 milissegundos para o total de 658 contatos existentes no aparelho.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Através deste estudo foi possível avaliar a performance para utilização de recursos de contatos da plataforma *Android*. Através dos resultados obtidos pode-se perceber que utilização do SDK nativo *Android* é muito superior a utilização do *framework Cordova*, para a funcionalidade avaliada. Vale ressaltar que apesar desta performance de acesso a conteúdos do dispositivo ser baixa, é uma tecnologia que proporciona uma facilidade no desenvolvimento de aplicações móveis devido a utilização de tecnologias *Web*.

Outro ponto observado neste estudo, é a diferença no tamanho dos artefatos gerados, onde o artefato *Cordova* ficou sete vezes maior que o artefato *Android*. Isto deve-se ao fato que para o *Cordova*, é necessário passar ao artefato todos *scripts javascript* utilizados, enquanto com o *Android*, toda API já está disponível nativamente no Sistema Operacional.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Este trabalho focou apenas uma entre várias funcionalidades disponíveis pela plataforma *Android* e *Cordova*, novas comparações entre outras funcionalidades podem ser úteis para projetistas e desenvolvedores na hora de decidir uma abordagem para desenvolver aplicações móveis. Pois a performance das aplicações em dispositivos móveis afeta diretamente a satisfação dos usuários.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ALLEN, S. **Pro Smartphone Cross-Platform Development**. Editora Apress 2010.

BURNETTE, E. Hello Android – **Introducing Google's Mobile Developing Platform.** <http://csis.pace.edu/~benjamin/teaching/cs122/webfiles/helloandroid.pdf>. Acesso em 26 set 2015.

FOWLER, M. **Developing Software for Multiple Mobile Devices.** Disponível em <http://martinfowler.com/articles/multiMobile/>. Acesso em 26 set 2015.

FOWLER, M. **Selecting a Mobile Implementation Strategy.** Disponível em <http://martinfowler.com/articles/mobileImplStrategy.html>. Acesso em 26 set 2015.

LINWOOD, J. **Creating iOS applications with jQuery Mobile, PhoneGap, and Drupal 7.** Disponível em <http://www.jefflinwood.com/wp-content/uploads/2012/02/PhoneGapChapterV1.pdf>. Acesso em 26 set 2015.

MEIR, J.D. HOMER, A. HILL, D. TAYLOR, J. BANSODE, P. WALL, L. BOUCHER, R. BOGAWAT, A. **Mobile Application Architecture Guide.** Disponível em <http://robtiffany.com/wp-content/uploads/2012/08/Mobile_Architecture_Guide_v1.1.pdf>. Acesso em 26 set 2015.

HAVERBEKE, M. **Eloquent JavaScript.** Disponível em <http://eloquentjavascript.net/Eloquent_JavaScript.pdf>, Acesso em 26 set 2015.

SARTI, Erika. 2009. Introdução ao HTML 5. Disponível em < http://www.infowester.com/introhtml5.php>. Acesso em 26 de Setembro de 2015.

SALEH, H. **JavaScript Mobile Application Development.** Disponível em < https://www.geekbooks.me/>. Acesso em 26 SET 2015.

SILVA, L. MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. 3ª edição. Laboratório de Ensino a Distância da UFSC. 2001.

SOOMER,A. **Comparison and evaluation of cross-platform frameworks for development of mobile business applications.** <https://wwwbruegge.informatik.tu-muenchen.de/lehrstuhl_1/research/paper/sommer2013crossplatform.pdf>, Acesso em 26 set.

SHOOTS, Kerri. PhoneGap 3.x Mobile Application Development HotSpot. Editora Packet 2014.

Stat Counter – Global Stats. Disponível em <http://gs.statcounter.com/#mobile+tablet-os-ww-monthly-201408-201508-bar> Acesso em 26 SET 2015.

TANENABAUM, A. S. WETHEHALL, D. J. **Computer Network – Fifth Edition.** Editora Pearson Education Inc.

TI Selvagem. **Desenvolvendo para Android : Arquitetura Android** Disponível em <http://www.tiselvagem.com.br/geral/desenvolvendo-para-android-arquitetura-android/>. Acesso em 26 set 2015.

Vision Mobile – disponível em <http://www.visionmobile.com/blog/2011/11/new-report-mobile-platforms-the-clash-of-ecosystems/>. Acesso em 26 set 2015.

ZANETI, L. A. (2003). **Sistemas de informação baseados na tecnologia web: um estudo sobre seu desenvolvimento.** Disponível em <http://www.teses.usp.br/teses/disponiveis/12/12139/tde-14082003-104928/>. Acesso em 26 set 2015.

ZIGURD, M. DORNIN L. MEIKE, G. NAKAMURA M. **Programing Android.** Editora O'Reilly.

WARGO, J. - Apache Cordova Programming 4. Editora Pearson - Adisson Wesley 2015

WASHINGTON POST. **4.4 billion people around the world still don't have Internet. Here's where they live. 02 de outubro de 2014.** Disponível em <http://www.washingtonpost.com/blogs/wonkblog/wp/2014/10/02/4-4-billion-people-around-theworld-still-dont-have-internet-heres-where-they-live/> Acesso em 26 set 2015.