

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

STEFANO WALKER PEREIRA PONTES

DESENVOLVIMENTO DE APLICATIVOS MÓVEIS HÍBRIDOS:
Um estudo de caso sobre o rMiífase

SÃO LUÍS
2017

STEFANO WALKER PEREIRA PONTES

DESENVOLVIMENTO DE APLICATIVOS MÓVEIS HÍBRIDOS:

Um estudo de caso sobre o rMiíase

Trabalho de Conclusão de Curso apresentado à Universidade Federal do Maranhão, como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Tiago Bonini Borchartt

SÃO LUÍS
2017

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo autor.
Núcleo Integrado de Bibliotecas/UFMA

Pontes, Stefano Walker Pereira.

Desenvolvimento de Aplicativos Móveis Híbridos: Um estudo de caso sobre o rMiíase / Stefano Walker Pereira Pontes. - 2017.

83 f.

Orientador(a): Tiago Bonini Borchartt.

Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2016.

1. Aplicativos Móveis. 2. Desenvolvimento Híbrido. 3. Desenvolvimento Web. 4. Miíase. I. Borchartt, Tiago Bonini. II. Título.

STEFANO WALKER PEREIRA PONTES

DESENVOLVIMENTO DE APLICATIVOS MÓVEIS HÍBRIDOS:

Um estudo de caso sobre o rMiíase

Trabalho de Conclusão de Curso apresentado à Universidade Federal do Maranhão, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 20 de janeiro de 2017

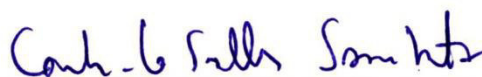
BANCA EXAMINADORA



Prof. Dr. Tiago Bonini Borchardt (Orientador)
Universidade Federal do Maranhão



Prof. Dr. Livio Martins Costa Junior
Universidade Federal do Maranhão



Prof. Dr. Carlos de Salles Soares Neto
Universidade Federal do Maranhão

Dedico este trabalho à minha mãe, meu fundamento. À minha avó materna, por ter seu primeiro descendente a se graduar. E à minha avó paterna que, com amor, me ensinou a ler e escrever.

AGRADECIMENTOS

Agradeço a meu Deus pela bênção de ter chegado até aqui com saúde e vitória. À minha querida mãe, Clemilda, e meus irmãos, Jossana e Dassaiev, pelo carinho e por sempre acreditarem em mim.

À minha amada Fran, por estar ao meu lado, me incentivando a alcançar os nossos sonhos. Aos meus amigos de academia pelo companheirismo e parceria, o que tornou a experiência e aprendizado mais agradável.

Agradeço, aos meus professores do curso, pelos ensinamentos e pelas lições de vida e, especialmente, aos professores Tiago Bonini e Livio Martins pela assistência no desenvolvimento desse trabalho e pelas oportunidades que a mim confiaram, permitindo meu crescimento dentro da universidade.

Aos colegas de trabalho do Departamento de Direito da UFMA representados pela professora Valéria Montenegro, que me apoiaram nos meus estudos. Agradeço de todo coração a todos estes e desejo que Deus continue nos guiando.

"Nada temos a temer quanto ao futuro, a menos que nos esqueçamos como Deus tem nos conduzido no passado."

Ellen G. White

RESUMO

Este trabalho apresenta um estudo acerca das diversas abordagens e tecnologias utilizadas no desenvolvimento de aplicativos móveis, dando ênfase ao tipo de desenvolvimento que apresentou uma grande evolução e que mais vem se popularizando nos últimos anos. Desenvolvimento Híbrido é um termo utilizado para se referir às diversas modalidades para desenvolvimento de aplicativos móveis multiplataforma, que tem possibilitado aos desenvolvedores a criação de aplicativos de forma ágil. Adicionando uma fundamentação teórica acerca da computação móvel, fez-se um estudo detalhado apresentando características, comparativos e análises sobre esse novo tipo de desenvolvimento. Como estudo de caso, é apresentado o projeto e análise do rMiíase, que é um aplicativo para o registro de caso de miíase de forma colaborativa. Utilizando o desenvolvimento híbrido, obteve-se um aplicativo multiplataforma com bons resultados de usabilidade e confiabilidade.

Palavras-chave: Aplicativos Móveis. Desenvolvimento Web. Desenvolvimento Híbrido. Miíase.

ABSTRACT

This paper presents a study about the different approaches and technologies used in the development of mobile applications, emphasizing the type of development that presented great evolution and that has been popularizing in recent years. Hybrid Development is a term used to refer to the various modalities for cross-platform mobile application development that have enabled developers to build applications agile. Adding a theoretical basis about mobile computing, a detailed study was presented with characteristics, comparisons and analyzes about this new type of development. As a case study, is presented the project and analysis of the rMíase, which is an application for collaborative registration of myiasis' cases. Using the hybrid development, a cross-platform application was obtained with good usability and reliability results.

Key words: Mobile Applications. Web Development. Hybrid Development. Myiasis.

LISTA DE FIGURAS

Figura 1 - Aplicativo Fruit Maps	19
Figura 2 - Aplicativo Dengue SC	20
Figura 3 - Modelo de Comunicação Genérico SBL	25
Figura 4 - Arquitetura do Sistema Operacional Android	32
Figura 5 - Ciclo de vida de uma Activity no Android.	34
Figura 6 - Arquitetura iOS.....	35
Figura 7 - Ciclo de vida de aplicação iOS.....	36
Figura 8 - Tipos de SOs móveis e suas linguagens de programação	38
Figura 9 - Aplicação web iOS de <i>The Financial Time</i> , sem interface do navegador (a); <i>WebView</i> de uma aplicação (b).	40
Figura 10 - Arquitetura genérica de componentes de aplicação híbrida.....	43
Figura 11 - Compatibilidade de <i>plug-ins</i> Cordova com diversos dispositivos	45
Figura 12 - Diagrama de aplicação Cordova	46
Figura 13 - Tela de emulação do Intel XDK.....	48
Figura 14 - Estrutura de interação do Intel XDK como seu sistema de compilação.	48
Figura 15 - Trecho de código em JavaScript e o resultado de sua interpretação no console.....	52
Figura 16 - Diagrama de classe relacionado ao banco de dados do aplicativo	57
Figura 17 - Diagrama de Sequência descrevendo a função Visualização de Casos.	58
Figura 18 - Diagrama de Sequência descrevendo a função Cadastro de Casos.	59
Figura 19 - Diagrama de Atividades com esquema de sincronismo de dados dos casos.....	60
Figura 20 - Arquitetura do sistema da aplicação.	61
Figura 21 - Arquivos do projeto do <i>app</i> rMífase gerado pelo Intel XDK.....	62
Figura 22 - Trecho de código-fonte HTML referente à tela de Registro de Usuário	62
Figura 23 - Tela de Registro de Usuário.....	63
Figura 24 - Painel do Google Play Developer Console	65
Figura 25 - Aplicativo rMífase: <i>Menu</i> (a); Tela inicial mostrando o mapa e os marcadores de casos cadastrados (b); Tela de Detalhes de um caso de outro usuário(c); Tela de Detalhes, após o <i>download</i> da imagem (d)..	66

Figura 26 - Aplicativo rMífase: Tela inicial com a adição do marcador de Cadastro de Caso (a). Formulário de Cadastro de caso (b). Captura de imagem da câmera ou galeria (c).	67
Figura 27 - Aplicativo rMífase: Lista dos casos cadastrados pelo usuário (a). Tela de detalhes de um caso cadastrado pelo usuário (b). Formulário de registro do usuário (c). Seção Sobre a Mífase (d).....	67
Figura 28 - <i>Homepage</i> do rMífase	69
Figura 29 - Funcionalidade de Edição de Caso no sistema de gerenciamento	70

LISTA DE TABELAS

Tabela 1 - Tecnologias para desenvolvimento híbrido de aplicativos	41
Tabela 2 - Comparativo entre aplicativos nativos e híbridos em relação às características da aplicação.	54
Tabela 3 - Tipos de requisições feitas pela aplicação ao servidor	63

LISTA DE SIGLAS E ABREVIATURAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
APK	package file format used by the Android operating system
ARM	Advanced RISC Machine
CPU	Central Processing Unit
CSS	Cascading Style Sheets
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HTML	HyperText Markup Language
HTML5	HyperText Markup Language - versão 5
HTTP	HyperText Transfer Protocol
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	Integrated Development Environment
iOS	iPhone Operational System
IPA	package file format used by the iOS operating system
JPG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
MVC	Model-View-Controller
ORM	Object Relational Mapping
PHP	Hypertext Preprocessor
PNG	Portable Network Graphics
POST	método de requisição fornecido pelo protocolo HTTP
RAM	Random Access Memory
SBL	Serviço Baseado em Localização
SDK	Software Development Kit
SO	Sistema Operacional
UI	User Interface
USB	Universal Serial Bus
WebGL	Web Graphics Library
XAP	package file format used by the Windows Phone operating system
XML	eXtensible Markup Language

SUMÁRIO

1	INTRODUÇÃO	16
1.1	Motivação	17
1.2	Objetivos	18
1.2.1	Objetivo Geral	18
1.2.2	Objetivos Específicos	18
1.3	Projetos Semelhantes	18
1.4	Organização do trabalho	20
2	FUNDAMENTAÇÃO TEÓRICA	21
2.1	Computação Móvel	21
2.1.1	Serviço Baseado em Localização	22
2.1.2	Técnicas Básicas de Localização	23
2.1.3	Modelo de Comunicação	24
2.2	Sistema de Posicionamento Global	26
2.2.1	Arquitetura	26
2.2.2	Funcionamento	28
2.3	Sistema de Posicionamento Baseado em Rede de Telefonia Móvel	28
2.4	Google API	29
2.4.1	Google Maps	30
2.4.2	Google Geolocation	30
2.5	Sistemas Operacionais Móveis	31
2.5.1	Android	31
2.5.2	Apple iOS	34
3	DESENVOLVIMENTO MÓVEL NATIVO E WEB	38
3.1	Aplicações Móveis Nativas	38
3.2	Aplicações Móveis Web	38
4	DESENVOLVIMENTO HÍBRIDO	41
4.1	Arquitetura	42
4.2	Frameworks e IDEs	43

4.2.1	Apache Cordova	44
4.2.2	Intel XDK.....	47
4.2.3	Frameworks UI	49
4.3	Discussões sobre desempenho	50
4.3.1	Compilação e Interpretação.....	50
4.3.2	Interface e Experiência de Usuário	52
5	ESTUDO DE CASO.....	55
5.1	Modelagem	55
5.1.1	Requisitos	55
5.1.2	Diagramas	56
5.2	Desenvolvimento	60
5.2.1	Arquitetura da aplicação	60
5.2.2	Ferramentas e tecnologias utilizadas	61
5.2.3	Servidor da Aplicação.....	63
6	RESULTADOS	65
6.1	Aplicativo Móvel	65
6.2	Website e Sistema de Gerenciamento	68
6.3	Avaliação dos resultados.....	70
7	CONCLUSÃO.....	72
	REFERÊNCIAS BIBLIOGRÁFICAS	74
	APÊNDICE A	77
	APÊNDICE B	79
	APÊNDICE C	81
	APÊNDICE D	82

1 INTRODUÇÃO

A evolução da tecnologia dos *smartphones* fez com que ligações e mensagens fossem apenas uma das diversas funcionalidades desses aparelhos. Com o advento de *hardwares* mais robustos, os *smartphones* conquistaram sistemas operacionais mais avançados, o que permitiu a execução de aplicativos mais complexos, com cada vez mais recursos e serviços ao usuário. Com essa enorme evolução, um *smartphone* se transformou em uma oportunidade de entretenimento, acesso a informação e solução de problemas, integrando-se assim ao cotidiano das pessoas e facilitando diversas tarefas do dia-a-dia. Pode-se observar que a característica da conexão contínua dos usuários em rede fez com que fosse permitido ao usuário fazer coisas que jamais fariam antes, pois agora é possível ter um minicomputador no bolso.

Toda essa evolução do *hardware* e a rápida conexão de dados fornecida pelos dispositivos móveis criou uma explosão de aplicativos móveis. Anteriormente, acessando a *web* através de aparelhos móveis com limitações no tamanho de tela e capacidade de processamento, a experiência era claramente frustrante. Depois, com a introdução do design *web* responsivo, o *website* passou a ser flexível para se apresentar em diversas resoluções e também ser controlado através das telas dos dispositivos móveis.

Esses aparelhos, por serem móveis, possuem sensores para diversas funções como orientação, localização, transmissão de dados e captura de imagens e vídeos. Todas elas tornam os aparelhos perfeitos instrumentos para redes sociais, pagamentos eletrônicos, compartilhamento de arquivos, aprendizado à distância, etc.

Contudo, para um aparelho de determinada plataforma, os sensores só poderiam ser acessados com a aplicação que foi desenvolvida para aquela plataforma específica. Uma aplicação de determinada plataforma deveria ser desenvolvida novamente para se adaptar a outra plataforma. Para solucionar este problema, surgiu o desenvolvimento de aplicações híbridas.

Segundo Charland e Leroux (2011), um aplicativo híbrido é parcialmente nativo, porque acessa as funcionalidades do *hardware*, e parcialmente *web*, porque utiliza HTML, CSS e JavaScript como tecnologias *web*. No desenvolvimento híbrido,

o produto final é um aplicativo com tecnologia *web* envolvida por um contêiner nativo que provê acesso a funcionalidades da plataforma nativa. Dessa forma, esse tipo de desenvolvimento oferece a grande vantagem de que o mesmo aplicativo funciona em várias plataformas e que, na maioria das vezes, não é necessário escrever um código diferente para cada uma delas. Utilizando-se dessas tecnologias, é possível o desenvolvimento menos oneroso de aplicativos.

Como estudo de caso desse trabalho, será abordado o desenvolvimento do aplicativo rMiíase, um aplicativo desenvolvido para o registro colaborativo dos casos de miíase. Esta doença é causada pelos estádios larvais de moscas que se alimentam de tecido vivo de animais de sangue quente provocando graves lesões dos tecidos e até a morte. Dentre as principais espécies de moscas que podem ser agentes etiológicos de miíase nas Américas, a principal é a *Cochliomyia hominivorax* (HALL; WALL 1995).

O objetivo do rMiíase (*report* miíase) é que seja um aplicativo acessível para qualquer pessoa registrar os casos de miíase, a popular “bicheira”. O usuário poderá cadastrar um caso de miíase preenchendo alguns dados básicos, como espécie do animal, endereço e parte de corpo lesionado. Depois, pode marcar um local no mapa, onde o caso está sendo registrado. Logo após, tirar uma foto do local lesionado pela doença. No final, há opção de sinalizar se o usuário quer colaborar coletando e enviando a amostra para um dos laboratórios credenciados, que aparecerão no mapa também.

O usuário pode ver todos os casos que foram cadastrados por outros usuários, bem como a foto que foi registrada por eles. Através do aplicativo, espera-se que as pessoas enviem as amostras para o laboratório credenciado mais próximo de sua localidade, fazendo com que os casos sejam homologados, obtendo-se informações da espécie de mosca que está causando a patologia. Uma das finalidades do aplicativo é que sirva como base estatística para estudo da possibilidade de implantação de um programa para controle da miíase.

1.1 Motivação

A temática abordada gira em torno da descrição resumida do desenvolvimento nativo e *web* de aplicativos móveis, apresentação detalhada do desenvolvimento de aplicativos híbridos, apresentação das tecnologias utilizadas em

diferentes tipos de aplicações, linguagens de programação, ambiente de desenvolvimento, plataforma de desenvolvimento e a compatibilidade com diferentes *hardwares* e sistemas operacionais.

O desenvolvimento de aplicativos, claramente, é um dos mercados que mais cresce no setor de TI (IBGE 2014). Desenvolvedores já observaram a imensa importância do desenvolvimento de aplicações híbridas, pois uma mesma implementação pode ser executada em várias plataformas, diminuindo assim, o tempo, custo, e esforço para disponibilizar um mesmo aplicativo para várias plataformas.

1.2 Objetivos

1.2.1 Objetivo Geral

Apresentar o desenvolvimento de aplicações móveis híbridas e suas tecnologias utilizadas, como linguagens de programação, processos de concepção, ferramentas, APIs e *frameworks*.

1.2.2 Objetivos Específicos

- a) Demonstrar as características dos aplicativos com desenvolvimento nativo e *web*.
- b) Apresentar as modalidades de desenvolvimento de aplicações híbridas, bem como as tecnologias e ferramentas utilizadas.
- a) Apresentar as tecnologias de recurso de mapas e GPS e como são utilizadas nas aplicações de dispositivos móveis.
- b) Apresentar o desenvolvimento do sistema de aplicação do rMífase e analisar os resultados obtidos.

1.3 Projetos Semelhantes

Existem algumas soluções já consolidadas no mercado, sobre aplicativos que têm o objetivo de funcionar como plataformas de registro, utilizando mapas e fazendo com que o usuário possa adicionar dados à base do aplicativo, criando uma

base de dados colaborativa. Foram escolhidos, com exemplo, o Fruit Map e o Dengue SC.

Fruit Map

O Fruit Map, apresentado na Figura 1 - Aplicativo Fruit Maps permite que os usuários marquem e encontrem árvores com frutos de forma colaborativa. O usuário indica no mapa a localização das árvores, selecionando o tipo e o grau de disponibilidade da fruta. Essas informações ficam disponíveis para todos visualizarem pelo dispositivo de forma que outros usuários podem saber quais as frutas estão mais próximas e onde estão (FRUITMAP, 2016).

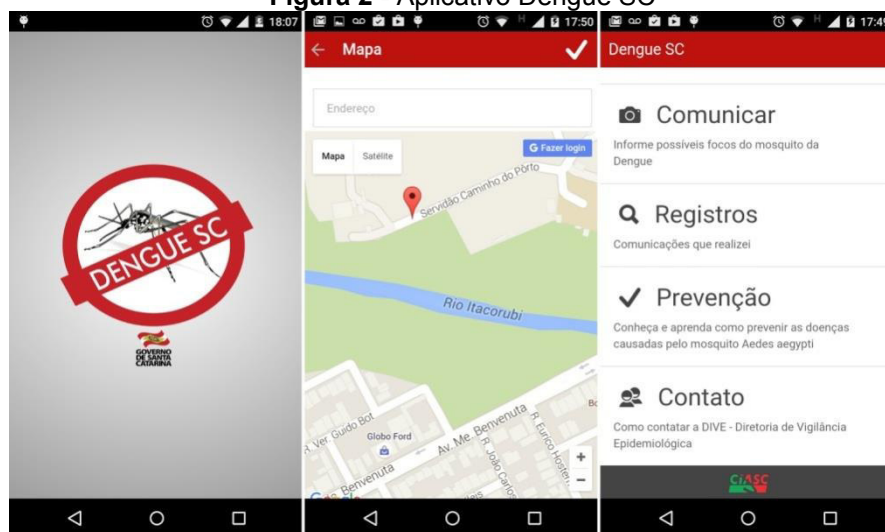


Fonte: Google Play. Adaptado pelo autor.

Dengue SC

Tendo características semelhantes ao anterior, o Dengue SC, apresentado na Figura 2, é um *app* interativo destinado a reforçar o combate ao mosquito *Aedes Aegypti*. Através deste canal, a população pode denunciar os focos e casos suspeitos de dengue. Assim que a denúncia for registrada, a prefeitura da cidade terá acesso à ocorrência por meio de uma ferramenta *web* que permitirá acompanhar as solicitações recebidas e dar andamento à verificação (CIASC, 2016).

Figura 2 - Aplicativo Dengue SC



Fonte: Google Play. Adaptado pelo autor.

1.4 Organização do trabalho

O Capítulo 2 apresenta a fundamentação teórica necessária para a compreensão do presente trabalho de forma completa. São descritos conceitos relacionados à Computação Móvel, Serviços Baseados em Localização e APIs Google. O Capítulo 3 detalha a estrutura dos sistemas operacionais móveis. É feito um comparativo entre o desenvolvimento nativo e o desenvolvimento *web* para aplicativos móveis. O Capítulo 4 apresenta o Desenvolvimento Híbrido de aplicações móveis, destacando a sua arquitetura, as ferramentas e o funcionamento da aplicação. Além disso, são apresentadas as vantagens e desvantagens desse tipo de desenvolvimento. O Capítulo 5 demonstra o processo de desenvolvimento do rMífase, a arquitetura do sistema da aplicação e as tecnologias de desenvolvimento. No Capítulo 6 apresentam-se os resultados obtidos, encerrando com o Capítulo 7, onde são feitas as considerações finais acerca de todo o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados alguns dados sobre a computação móvel, descrevendo as suas características. Logo após, serão apresentados os Serviços Baseados em Localização, as técnicas de localização e os modelos de comunicação. Depois será apresentado o funcionamento do sistema de posicionamento global, a sua arquitetura, e os possíveis problemas que afetam esse sistema. Encerrando o capítulo, serão apresentadas as APIs da Google utilizadas para a concepção do aplicativo desenvolvido como caso de estudo neste trabalho. O detalhamento dessas tecnologias se faz necessário, pois foi utilizado no estudo de caso deste trabalho.

2.1 Computação Móvel

A Computação móvel pode ser representada como um novo paradigma computacional, que permite que usuários desse ambiente tenham acesso a serviços independentemente de sua localização, mesmo estando em movimento. De forma mais técnica, é um conceito que envolve processamento, mobilidade e comunicação sem fio. A ideia é ter acesso à informação em qualquer lugar e a qualquer momento (FIGUEIREDO, 2003).

A evolução da tecnologia dos aparelhos celulares oferece ao usuário recursos que vão muito além da realização de uma chamada ou do envio de uma mensagem. As melhorias de *hardware* dos aparelhos celulares permitiram o desenvolvimento de sistemas operacionais mais avançados. Com este avanço foi possível desenvolver aplicativos melhores. Outro destaque é a comunicação sem fio que, em suas diversas formas, elimina a necessidade de dispositivos móveis estarem conectados a uma rede física. A comunicação sem fio é a mais importante tecnologia da computação móvel.

Um dispositivo é considerado móvel quando é portátil, pessoal, permanece junto ao usuário, fácil e rápido de usar e possui algum tipo de conexão de rede.

Os aparelhos móveis, segundo (FIRTMAN, 2010) podem ser categorizados em: Telefone Móvel (Celular), *Feature Phone*, Telefones Sociais, *Smartphones*, *Tablets*, *Netbooks* e Dispositivos móveis sem telefone.

Os ambientes de computação ubíqua deverão expandir e motivar a computação móvel. Segundo pesquisa divulgada pelo IBGE, os *smartphones* ultrapassaram os computadores e se tornaram os aparelhos preferidos do brasileiro para se conectar a internet em 2014 (IBGE, 2014). No mundo, a expectativa é que, até 2016, 3.2 bilhões de pessoas ao redor do mundo tenham acesso à internet, sendo que 2 bilhões acessarão pelo telefone celular (IDC, 2016).

As principais características da computação móvel são a conectividade, a descentralização, a diversidade, e a simplicidade. A conectividade é a propriedade que se preocupa em garantir a troca de dados entre os dispositivos conectados a uma rede sem fio. As informações devem ser facilmente trocadas, independentemente do tipo do dado ou do tipo da aplicação. A característica de descentralização permite que dispositivos em lugares distintos possam cooperar entre si, de acordo com sua necessidade. A terceira característica diz que variados dispositivos possam ter comunicação compatível, compreendendo os variados protocolos de comunicação. A característica de simplicidade visa oferecer aos usuários sistemas usuais, ou seja, sistemas fáceis de serem utilizados.

2.1.1 Serviço Baseado em Localização

Os SBLs podem ser definidos como serviços que integram a localização de um dispositivo móvel com outras informações de modo a proporcionar valor agregado ao usuário (SCHILLER; VOISARD, 2004). Estes serviços coletam informações do usuário, como por exemplo, dados de seu perfil e a localização geográfica de lugares que frequenta, que aplicadas em um contexto, oferecem serviços que atendem às suas necessidades.

Esse tipo de serviço surgiu quando o Departamento de Defesa dos Estados Unidos implantou o Sistema de Posicionamento Global (GPS) em 1970. Inicialmente, o GPS estava restrito para usos militares, mas na década de 1980 começou a ser disponível de forma restrita para o uso civil e industrial, a fim de estimular a inovação em torno da tecnologia por satélite. Depois disso, empresas de

todo o mundo têm agregado este tipo de serviço aos seus produtos. Mas somente no fim da década de 1990, o GPS começou a ter seu uso intensificado.

Um das importantes classificações SBL é quanto à orientação. O *SBL orientado a pessoa* é quando o foco da aplicação é usar a posição de uma pessoa para melhorar um serviço. Normalmente, a pessoa localizada pode controlar o serviço. No *SBL orientado a dispositivo*, em vez de apenas uma pessoa, um objeto ou um grupo de pessoas poderia também ser localizado. Em aplicativos orientados a dispositivos, a pessoa ou objeto localizado normalmente não controla o serviço como, por exemplo, rastreamento de carro roubado.

Além desta primeira classificação, os serviços podem ser definidos como *push* e *pull*. Os serviços *push* implicam que o usuário recebe informações como resultado de seu paradeiro sem ter que solicitar ativamente. As informações podem ser enviadas ao usuário com ou sem consentimento prévio. Geralmente as aplicações possuem opções de configuração, onde o usuário autoriza se deseja receber notificações *push* e em quais condições de conectividade.

Serviços *pull* ocorrem quando o usuário usa ativamente um aplicativo e, neste contexto, "puxa" informações da rede. Essas informações podem ser baseadas na localização atual como, por exemplo, encontrar o cinema mais próximo do usuário.

Um fator que aumenta o custo dos serviços *push* é o gerenciamento do perfil do usuário. Além do consentimento e das regras para entrar em contato com os usuários, os aplicativos *push* precisam armazenar o histórico de mensagens enviadas como, cupons de publicidade enviados, para evitar duplicação. Outra preocupação dos serviços *push* é em relação à privacidade, pois o caráter inerente da rede móvel atualizando frequentemente a posição da pessoa na rede já levanta a noção de rastreamento em tempo real.

2.1.2 Técnicas Básicas de Localização

Os sistemas que determinam a localização de um usuário móvel podem ser divididos em duas categorias: rastreamento e posicionamento (SCHILLER; VOISARD, 2004).

Acontece rastreamento, se uma rede de sensores determina a localização do usuário. Nesse caso, o usuário tem que usar uma etiqueta ou distintivo específico

que permita que a rede de sensores rastreie sua posição. As informações de localização do usuário ficam disponíveis na rede de sensores. Se o usuário móvel precisar de seus dados de localização, a rede de sensores tem que transferir essas informações para o usuário por comunicação sem fio.

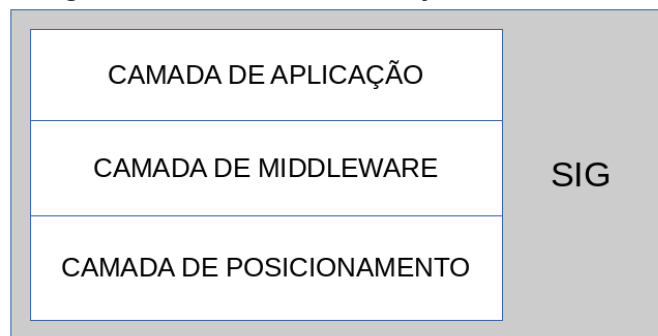
Se o sistema móvel determina sua própria localização, usa-se o termo posicionamento. Um sistema de transmissores ou balizas envia sinais de rádio, infravermelho ou ultrassom. As informações de localização estão diretamente disponíveis no sistema móvel e não precisam ser transferidas sem fio.

Os sistemas que utilizam o rastreamento e o posicionamento baseiam-se em algumas das seguintes técnicas básicas, frequentemente utilizadas em combinação:

- a) *Cell of Origin (COO)*: Técnica utilizada quando o sistema de posicionamento tiver uma estrutura celular. A célula identifica quando determinado dispositivo está presente em sua área de cobertura.
- b) *Time of Arrival (TOA)*, *Time Difference of Arrival (TDOA)*: Os sinais eletromagnéticos se movem com a velocidade da luz. Como esta velocidade é muito alta, os tempos de execução correspondentes são muito curtos. Assumindo a velocidade da luz como constante, pode-se usar a diferença de tempo entre enviar e receber um sinal, para calcular a distância espacial do transmissor e do receptor. Um princípio semelhante pode ser usado com ultrassom. Os sinais levam um tempo mais longo, assim a medida é mais simples, mas o ultrassom só pode alcançar distâncias baixas. Se medimos a diferença de tempo entre dois sinais, usamos o termo TDOA.
- c) *Angle of Arrival (AOA)*: Se antenas com características de direção são utilizadas, pode-se descobrir de que direção chega um certo sinal. Dadas duas ou mais direções de posições fixas para o mesmo objeto, pode-se calcular a localização do objeto.

2.1.3 Modelo de Comunicação

A realização de um serviço baseado em localização pode ser descrito como um modelo de três camadas. A camada de posicionamento, a camada de *middleware* e a camada de aplicação são apresentadas na Figura 3.

Figura 3 - Modelo de Comunicação Genérico SBL

Fonte: Adaptado de Schiller e Voisard (2004).

A camada de Posicionamento é responsável pelo cálculo da posição do dispositivo móvel ou usuário. Isto pode ser feito com ajuda de um PDE (*Position Determination Equipment*) ou dados geoespaciais armazenados em um Sistema de Informação Geográfico (SIG).

O PDE tem a função de calcular onde um dispositivo está, em termos de rede, para que o SIG traduza essa informação para dados geoespaciais (latitude e longitude). O resultado final deste cálculo é repassado ou para uma plataforma de *middleware* ou diretamente para um aplicativo. Apesar de existir a possibilidade de enviar dados diretamente para a camada de aplicação, é aconselhável manter uma camada de *middleware* para fazer a conexão entre as duas extremidades.

O objetivo do *middleware* é fornecer um modelo mais produtivo para os programadores de aplicativo SBL, reduzindo significativamente a complexidade de integração de serviços, uma vez que, por meio deste recurso, a camada de posicionamento vê a camada de aplicação como uma única entidade, sendo necessário apenas uma integração, independentemente da quantidade de aplicações SBL reais existentes nela. Em relação à camada de aplicação, as aplicações só veem um único serviço de posicionamento, e através de uma única API, podem localizar dispositivos, obter imagens de mapas, enviar e receber mensagens, e obter diversas outras informações geoespaciais (SCHILLER; VOISARD, 2004).

A terceira e mais alta camada, a de aplicação, (a qual a indústria de SBL tem frequentemente e confusamente tem se referido como “cliente”) é responsável por entregar os dados de localização aos serviços que os solicitaram.

Na sessão seguinte, será apresentado o Sistema de Posicionamento Global, que é o mais clássico exemplo de aplicação SBL existente.

2.2 Sistema de Posicionamento Global

O sistema de posicionamento global, GPS (*Global Positioning System*) é um sistema de posicionamento por satélite que fornece a um aparelho receptor móvel a sua posição, assim como informação horária, sob quaisquer condições atmosféricas, a qualquer momento e em qualquer lugar na Terra, desde que o receptor se encontre no campo de visão de, no mínimo, três satélites GPS (VAZ, 2013).

Inicialmente, o GPS foi denominado NAVSTAR (Navigation System with Timing and Ranging), desenvolvido em 1973 pelo Departamento de Defesa do Governo dos Estados Unidos e inicialmente idealizado para uso militar e aberto gratuitamente para uso civil a partir da década de 80 (GPS, 2016).

O Sistema de Posicionamento Global é composto de três segmentos: o segmento espacial, o segmento de controle e o segmento do usuário. Possui as vantagens de que o posicionamento pode ser obtido em qualquer lugar da Terra, as condições ambientais pouco afetam o processo, e possui alta precisão. As desvantagens são os custos e supervisão dos satélites, o posicionamento só é obtido se for possível receber sinal de uma certa quantidade de satélites e obtenção da localização dentro de prédios e locais fechados não é possível.

2.2.1 Arquitetura

Segmento Espacial

O segmento espacial GPS: Consiste numa constelação de satélites que transmitem sinais de rádio aos utilizadores. Os Estados Unidos mantêm em órbita terrestre 31 satélites de GPS, para garantir que haja no mínimo 24 satélites funcionando simultaneamente. Os satélites GPS voam em órbita terrestre média a uma altitude de aproximadamente 20.200 km. Cada satélite circunda a Terra duas vezes por dia (GPS, 2016).

Os satélites na constelação GPS estão dispostos em seis planos orbitais igualmente espaçados que cercam a Terra. Cada plano contém quatro "slots"

ocupados por satélites de linha de base. Este arranjo de 24 *slots* garante que os usuários podem ver pelo menos quatro satélites de praticamente qualquer ponto do planeta.

Em junho de 2011, a Força Aérea concluiu com êxito uma expansão de constelação GPS conhecida como configuração "Expandable 24". Três dos vinte e quatro *slots* foram expandidos e seis satélites foram reposicionados, de modo que três dos satélites extras se tornaram parte da linha de base da constelação. Como resultado, o GPS agora funciona efetivamente como uma constelação de 27 *slots* com cobertura melhorada na maioria das partes do mundo.

Segmento de Controle Terrestre

Consiste em uma rede global de instalações terrestres (uma estação de controle mestre, uma estação de controle mestre alternativo, quinze locais de monitoramento, e onze antenas de comando e controle) que rastreiam os satélites GPS, monitoram suas transmissões, executam análises e enviam comandos e dados para a constelação.

A finalidade de todo esse segmento é monitorar continuamente todo o sistema de forma que, sempre que surja uma anomalia, o Departamento de Defesa dos Estados Unidos possa atualizar a posição dos satélites, sincronizar o relógio atômico presente nos satélites e nas estações terrestres (GPS, 2016).

Segmento de Usuários

Refere-se à parte do sistema que interage diretamente com os usuários, tanto civil como militar. Esse segmento inclui receptores, programas de processamento, métodos e técnicas de levantamentos.

Se refere à parte do sistema que é disponível aos usuários. Diversos dispositivos, desde um carro até um relógio, podem possuir um sensor de geolocalização, um receptor GPS, com o objetivo de determinar a localização atual do dispositivo na Terra. Receptores GPS fornecem ao usuário informações de latitude e longitude, sendo que alguns conseguem calcular também a altitude.

O GPS é um elemento essencial da infra-estrutura de informação global. A natureza livre, aberta e confiável do GPS levou ao desenvolvimento de centenas de aplicações que afetam todos os aspectos da vida moderna. As principais redes

de comunicações, sistemas bancários, mercados financeiros e redes elétricas dependem fortemente do GPS para uma sincronização precisa do tempo.

2.2.2 Funcionamento

Os satélites, assim como os receptores GPS, possuem um relógio interno que marca as horas com uma precisão incrivelmente grande em nanosegundos. Satélites possuem órbita fixa, o que permite calcular a exata posição dos mesmos. Um almanaque contém a lista dos satélites em operação e a sua órbita. Ele é frequentemente carregado no dispositivo móvel. Quando o satélite emite o sinal para o receptor, o horário em que ele saiu do satélite também é enviado. Este sinal enviado para o receptor é um sinal de rádio, que viaja na velocidade da luz.

Resta ao receptor calcular quantos nanosegundos este sinal demorou chegar até ele. Assim ele consegue estimar em que raio de distância o receptor está do satélite. A medição do tempo é crítica: um erro de 1 μ s ocasiona uma variação de 300 metros. Por isso, cada satélite é equipado com um relógio atômico. O tempo exato de todo o sistema é chamado tempo do sistema.

O receptor usa triangulação para determinar sua localização aproximada. Basicamente, ele define uma esfera em torno de cada um dos três satélites que ele pode localizar. A interseção entre essas esferas com referência mais próxima à superfície terrestre é o ponto de localização. Para ter mais precisão ou para conseguir saber a altura do receptor em relação ao nível do mar, é necessário um quarto satélite.

2.3 Sistema de Posicionamento Baseado em Rede de Telefonia Móvel

Sistemas celulares são particularmente interessantes como ferramenta de localização, já que a identificação da célula pode ser utilizada para este objetivo. Mecanismos adicionais como TOA e AOA, mencionados acima, podem melhorar a precisão do sistema.

Em sistemas GSM, localização de célula pode ser obtida sem a necessidade de instalações adicionais. A Ericsson desenvolveu o MPS (*Mobile Positioning System*) que aumenta a precisão de localização em redes GSM sem necessidade de modificar os terminais móveis.

As seguintes aplicações são possíveis: o usuário pode consultar por dados dependentes da localização; usuários podem supervisionar a localização de outros usuários; aplicações de navegação podem calcular rotas ótimas.

Segundo Schiller e Voisard (2004), através da rede de telefonia móvel, é possível a localização por meio das seguintes técnicas:

- a) CGI (*Cell of Global Identity*): método baseado na localização da célula, utilizado somente se outros mecanismos mais precisos não estiverem disponíveis. Possui precisão de alguns quilômetros.
- b) *Segment Antenna*: estações base normalmente possuem várias antenas que dividem os 360 graus da célula em segmentos de 180, 120 ou 90 graus.
- c) TA (*Timing Advance*): medem o tempo que leva um sinal para viajar entre um telefone celular e um transmissor, para calcular a distância aproximada. Precisão é de cerca de 1 km.
- d) UL-TOA (*Uplink Time of Arrival*): uma precisão maior ainda pode ser obtida se um dispositivo móvel puder ser alcançado por pelo menos 4 estações base; Medindo-se o tempo de propagação do sinal do dispositivo móvel para as estações base a posição pode ser determinada com uma precisão de 50 a 150 m.

2.4 Google API

Google API é um conjunto de interfaces de programação de aplicações (APIs), desenvolvido pela Google que permite a comunicação com serviços da Google e sua integração a outros serviços. As APIs fornecem funcionalidades como análise, aprendizado de máquina como um serviço ou acesso aos dados do usuário (quando é dada permissão para ler os dados). Exemplos destes serviços incluem Search, Gmail, Translate ou Google Maps. Aplicativos de terceiros podem usar essas APIs para aproveitar ou estender a funcionalidade dos serviços existentes de modo a agregar valor a uma aplicação, porém, em outros casos, são inteiramente essenciais para as mesmas, de modo que sem estas APIs as aplicações não existiriam (LECHETA 2010).

As APIs utilizadas no desenvolvimento da ferramenta desenvolvida foram Google Maps e Google Geolocation.

2.4.1 Google Maps

A API Google Maps permite que os usuários utilizem mapas e informações de mapeamento personalizado em suas aplicações, possibilitando aos mesmos explorarem o mundo através dos mapas fornecidos pela Google.

O usuário pode escolher o tipo de vista do mapa (satélite, terreno e mapas híbridos), personalizá-lo adicionando marcadores sobre o mapa para indicar pontos especiais de interesse e efetuar efeitos de câmera, como por exemplo, rotação, zoom e inclinação que dependendo da região, podem apresentar mapas em três dimensões. O Google Maps é gratuito e para ser utilizado em aplicações de qualquer plataforma é necessário obter uma chave de autenticação para utilizar o serviço (MAPS, 2016).

As Google Maps APIs são categorizadas por plataforma: *Web*, Android e iOS. Essas APIs de plataformas nativas são complementadas pelo conjunto de serviços *web* HTTP da Google (MAPS, 2016). Como o desenvolvimento da aplicação do estudo de caso deste trabalho é baseado em tecnologia de plataforma *web*, então utilizou-se a Google Maps JavaScript API, que pertence à categoria de APIs *web*. Nesta vertente de API, assim como nas outras, todo o mapa é personalizável. Os estilos de personalização do mapa são definidos usando o formato JSON¹, sendo assim, a definição de um mesmo mapa é suportada em toda a *web*, Android e iOS. E utilizando o formato JSON, pode-se armazená-lo de qualquer forma, podendo aplicar o estilo dinamicamente na aplicação em tempo de execução. A Google disponibiliza, além de sua documentação, o Assistente Styling que é uma plataforma visual para geração da definição JSON dos componentes visuais do mapa (rótulos, cores, tipos de vias e suas densidades, etc).

2.4.2 Google Geolocation

O Google Geolocation é o serviço que fornece ao usuário informações necessárias para obter a localização de seu aparelho. Usando esta API, aplicativos podem solicitar a localização conhecida do dispositivo. Especificamente, é utilizado

¹ JSON (com a pronúncia J-son em inglês), um acrônimo para "*JavaScript Object Notation*", é um formato leve para intercâmbio de dados computacionais. JSON é um subconjunto da notação de objeto de JavaScript, mas seu uso não requer JavaScript exclusivamente (FLANAGAN, 2006).

um provedor de localização mesclado para recuperar a localização conhecida do dispositivo. O provedor de localização mesclado é uma das APIs de localização da Google. Ele provê uma lista dos provedores de localização disponíveis e registra como receber essas informações sobre a localização, definindo o local através da latitude e da longitude. O provedor de localização também é responsável em otimizar o uso de energia da bateria do dispositivo.

A Google Geolocation API retorna uma localização e um raio de precisão com base em informações de torres de celular e pontos de acesso Wi-Fi que o cliente móvel pode detectar (GEOLOCATION, 2016).

A comunicação é realizada por HTTPS usando POST. Tanto a solicitação quanto a resposta têm formatação JSON e o tipo de conteúdo de ambas é *application/json*.

2.5 Sistemas Operacionais Móveis

Com o surgimento de novas tecnologias foi necessário a criação e aperfeiçoamento de novos dispositivos móveis como *smartphones*, *tablets* e *netbooks* e conseqüentemente a necessidade de ter sistemas operacionais em dispositivos móveis para gerir os recursos dos dispositivos. Grandes empresas de tecnologia investiram em sistemas operacionais próprios ajudando evoluir a área, visto a necessidade de uma reformulação na arquitetura de *hardware* e *software*.

Os mais conhecidos sistemas operacionais móveis são: Android, iOS, Windows Phone, Symbian, WebOS, BlackBerry, etc. Cada SO tem suas particularidades, mas todos possuem algumas funcionalidades em comum como: uso de recursos, apoio a multitarefas, execução de aplicativos, conectividade e interface limpa. O Android da Google e iOS da Apple são os SOs mais utilizados atualmente em *smartphones* (LECHETA, 2010).

2.5.1 Android

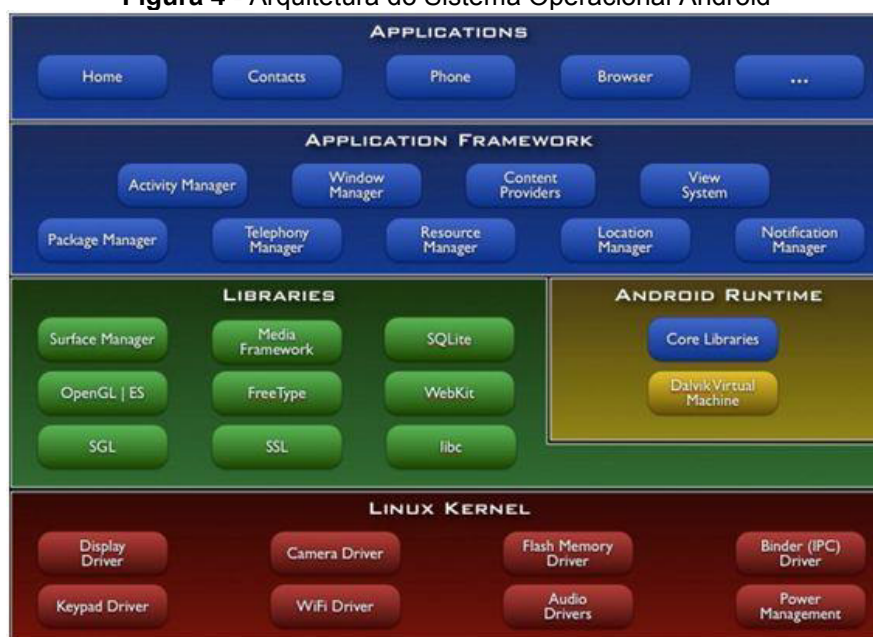
O Android é uma plataforma para *smartphones*, baseada no sistema operacional Linux, que possui diversos componentes, com uma variada disponibilidade de bibliotecas e interface gráfica, além de disponibilizar ferramentas para a criação de aplicativos. O código do sistema operacional é disponibilizado pela

Google sob licença de código aberto. A Google lidera a Open Handset Alliance, que é o consórcio entre empresas de *hardware*, *software* e telecomunicações com o intuito de desenvolver a indústria de dispositivos móveis (LECHETA, 2010).

2.5.1.1 Arquitetura

O Android é definido em uma arquitetura de 4 camadas, cada uma com sua função bem definida, e todas elas fundamentais para um bom funcionamento de todo o sistema. Roda sobre o *kernel* do Linux, dando suporte para o desenvolvimento de aplicações através de um conjunto de bibliotecas e serviços, apresentado na Figura 4.

Figura 4 - Arquitetura do Sistema Operacional Android



Fonte: (DEVELOPER, 2016a).

A base da pilha é o *Linux Kernel* que inclui os programas de gerenciamento de memória, as configurações de segurança, o *software* de gerenciamento de energia e vários *drivers* de *hardware*. O *Linux Kernel* se baseia no Linux 2.6, isso garante que o Android possua características similares a qualquer outro sistema operacional comum que roda em um computador pessoal.

A segunda camada é dividida em duas partes: na primeira encontram-se as bibliotecas, um conjunto de instruções que dizem ao dispositivo como lidar com diferentes tipos de dados. Em sua maior parte, são desenvolvidas nas linguagens C

e C++, usadas por diversos componentes do sistema e são expostas a desenvolvedores através da estrutura de aplicativo Android; a segunda é a *Android Runtime* que inclui um conjunto de bibliotecas do núcleo Java (*Core Libraries*) e a *Dalvik Virtual Machine* (DVM). O desenvolvimento de aplicativos deve ser em linguagem de programação Java, porém, suas aplicações não serão executadas em uma máquina virtual Java tradicional, e sim na DVM. Esta permite que cada *thread* rode sua própria instância da máquina virtual, isto garante que nenhuma aplicação seja dependente de outra, e se uma aplicação parar, ela não afetará quaisquer outras aplicações que estão rodando no dispositivo.

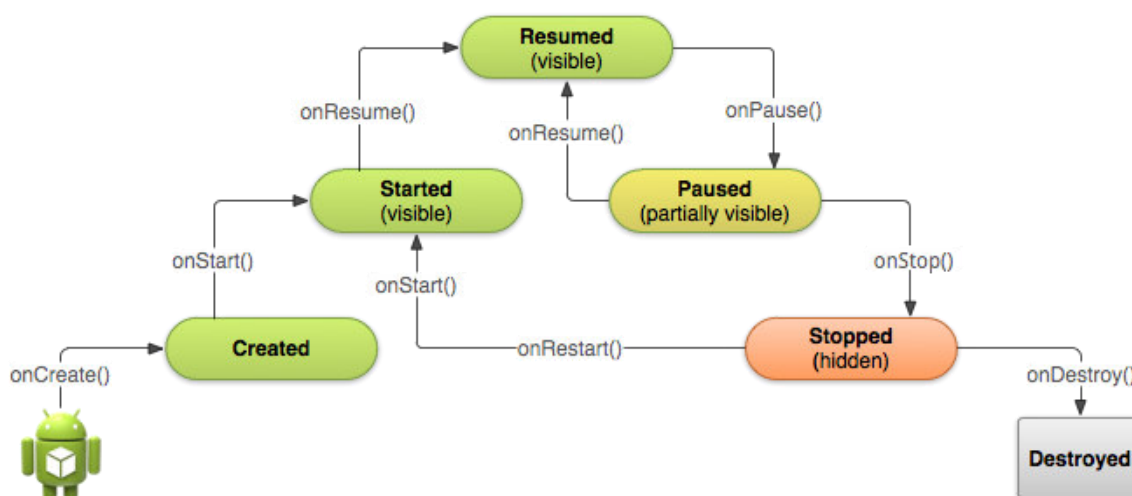
A camada *Application Framework* funciona como uma API que é usada abertamente no desenvolvimento de aplicações nativas. A camada *Applications* se encontra no topo das camadas e contém todo o conjunto de aplicações nativas ou desenvolvidas por terceiros. Esta camada promove a interação entre o dispositivo móvel e o usuário.

2.5.1.2 Ciclo de Vida da Activity

Uma Activity, na maioria dos casos, representa uma tela da aplicação que serve como interface para os usuários, onde eles podem interagir com a aplicação realizando algo ou visualizando alguma informação. Pode-se também dizer que uma Activity representa uma atividade, ação ou funcionalidade que o usuário pode realizar dentro da aplicação (LECHETA, 2010).

Apesar das atividades serem geralmente representadas para os usuários como janelas em telas cheias, elas também podem ser usadas de outras maneiras como janelas flutuantes ou incorporadas dentro de outras atividades (DEVELOPER, 2015b). Na Figura 5 é apresentado o ciclo de vida de uma Activity.

Figura 5 - Ciclo de vida de uma Activity no Android.



Fonte: (DEVELOPER, 2016b).

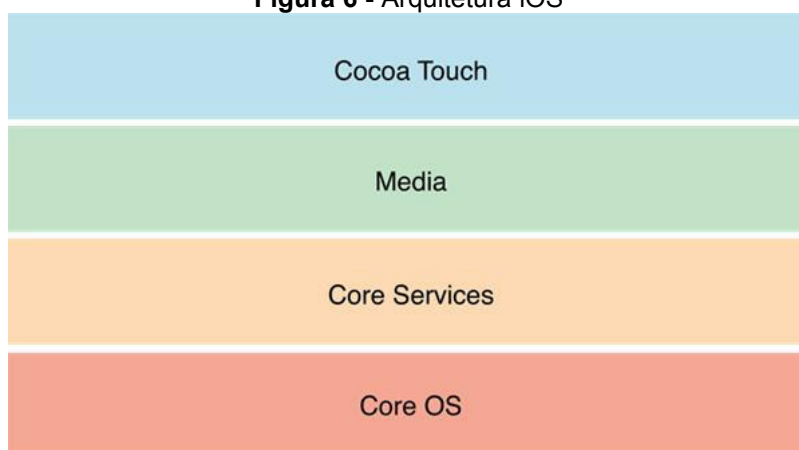
2.5.2 Apple iOS

É o sistema operacional móvel dos *smartphones* da Apple Inc. Anteriormente chamando de iPhone OS, o iOS foi lançado em 2008, se apresentando como um sistema com bom desempenho, que mantém alta estabilidade e segurança para dispositivos móveis. Com base em Unix, o sistema é de código fechado, o que impossibilita a instalação do mesmo em equipamentos que não pertençam à empresa desenvolvedora (APPLE, 2016a).

2.5.2.1 Arquitetura

A arquitetura do iOS é formada por quatro camadas, sendo que cada uma delas oferece um conjunto de *frameworks* que podem ser utilizados durante o desenvolvimento de aplicativos para os dispositivos móveis da Apple Inc. As camadas são: Core OS, Core Services, Media e Cocoa Touch, como mostrado na Figura 6.

No nível mais alto, a arquitetura atua como intermediário entre o *hardware* subjacente e os aplicativos que aparecem na tela. Os aplicativos desenvolvidos para o iOS raramente se comunicam diretamente com o *hardware* do dispositivo, ao invés disso, os aplicativos se comunicam com o *hardware* através de um conjunto de interfaces de sistema bem definidas que protegem o aplicativo de alteração de *hardware* (APPLE, 2016a).

Figura 6 - Arquitetura iOS

Fonte: (APPLE, 2016b).

Nas camadas superiores estão as tecnologias e serviços mais sofisticados. O desenvolvedor deve olhar primeiro, sempre que possível, os serviços das camadas superiores, pois nestas camadas estão os *frameworks* que fornecem abstração orientada a objetos das camadas de níveis inferiores. Estas abstrações geralmente facilitam o processo de escrita de código, pois reduzem a quantidade de código que o desenvolvedor tem que escrever e encapsula características complexas, tais como *threads*.

Na camada Cocoa Touch são definidos os principais *frameworks* para a construção de aplicações e a infraestrutura para as tecnologias fundamentais, tais como multitarefa, serviço de notificação Apple *push* e diversos serviços de alto nível do sistema.

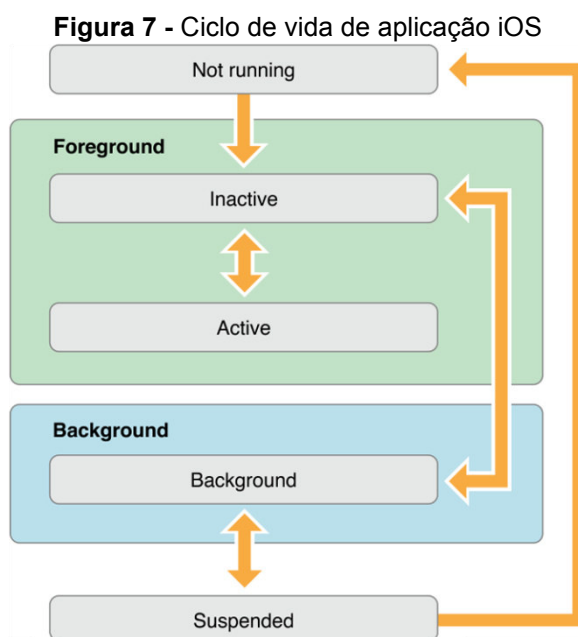
A camada Media contém as tecnologias de gráfico, áudio e vídeo. As tecnologias nessa camada foram projetadas para tornar mais fácil a implementação de aplicativos multimídia. Os *frameworks* de nível superior oferecem tecnologias que tornam mais fácil a criação de gráficos e animações, enquanto os *frameworks* de nível inferior permitem o acesso a ferramentas fundamentais que o desenvolvedor pode utilizar para criar aplicativos mais robustos e complexos (APPLE, 2016b).

A camada Core Services contém os serviços fundamentais do sistema que todos os aplicativos utilizam. Mesmo se o desenvolvedor não usar esses serviços diretamente, muitas partes do sistema são construídas em cima deles. As principais tecnologias disponíveis na camada Core Services são: *grand central dispatch*, *in-app purchase*, *SQLite* e *XML support*.

A camada Core OS contém características de baixo nível que foram utilizadas na implementação de outras tecnologias. Em situações onde o desenvolvedor precisa lidar explicitamente com segurança ou comunicação com acessório de *hardware* externo, ele pode fazer isso utilizando os *frameworks* nessa camada (APPLE, 2016b).

2.5.2.2 Ciclo de Vida

O iOS muda os estados de uma aplicação em resposta a ações que acontecem em todo o sistema. A Figura 7 mostra as transições de estado de uma aplicação.



Fonte: (APPLE, 2016c).

Cada estado pode ser definido como:

- Not running*: o aplicativo não foi iniciado ou estava em execução, mas foi encerrado pelo sistema.
- Inactive*: o aplicativo está sendo executado em primeiro plano, mas atualmente não está recebendo eventos, embora possa estar executando outro código.
- Active*: o aplicativo está sendo executado em primeiro plano e está recebendo eventos. Este é o modo normal para aplicativos de primeiro plano.

- d) *Background*: o aplicativo está em segundo plano e em execução de código. A maioria dos aplicativos entra neste estado brevemente antes de ser suspenso. No entanto, um aplicativo que solicita tempo de execução extra pode permanecer neste estado por um período de tempo. Além disso, um aplicativo sendo lançado diretamente no *background* entra nesse estado em vez do estado *Inactive*.
- e) *Suspended*: o aplicativo está no segundo plano, mas não está executando o código. Enquanto suspenso, um aplicativo permanece na memória, mas não executa nenhum código. Quando ocorre uma condição de pouca memória, o sistema pode limpar os aplicativos suspensos sem aviso prévio para criar mais espaço para o aplicativo em primeiro plano.

3 DESENVOLVIMENTO MÓVEL NATIVO E WEB

3.1 Aplicações Móveis Nativas

São denominadas aplicações móveis nativas aquelas que são desenvolvidas para determinado sistema operacional. Esse tipo de desenvolvimento tem como característica principal o fato de que é desenvolvido em linguagem de programação específica para cada SO. A Figura 8 relaciona as plataformas com as linguagens utilizadas.

Figura 8 - Tipos de SOs móveis e suas linguagens de programação

Mobile OS Type	Skill Set Required
Apple iOS	C, Objective C
Google Android	Java (Harmony flavored, Dalvik VM)
RIM BlackBerry	Java (J2ME flavored)
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Window 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

Fonte: (FIRTMAN, 2010).

Depois de desenvolvida a aplicação, é necessário ter uma conta associada a cada uma das plataformas para que o desenvolvedor possa disponibilizar o aplicativo nas lojas oficiais. Cada plataforma exige um conjunto de procedimentos de assinatura, certificação e revisão de modo a garantir que haja aplicativos seguros nas lojas.

3.2 Aplicações Móveis Web

A *web* móvel é basicamente a mesma *web desktop*, pois usa a mesma arquitetura básica e as mesmas tecnologias. Mas diferem em alguns pontos como: redes mais lentas com maior latência, *hardware* mais lento e menos memória disponível, experiência de navegação diferente, diferentes contextos de usuário,

comportamento diferente do navegador (normalmente apenas a guia atual em um navegador para dispositivos móveis está em execução ativa), muitos navegadores *web* móveis, com versões diferentes no mercado ao mesmo tempo.

Uma aplicação móvel *web* pode ser executada em um navegador, ou também dentro de algum outro aplicativo nativo ou plataforma que pode levar o desenvolvimento da *web* para um nível diferente.

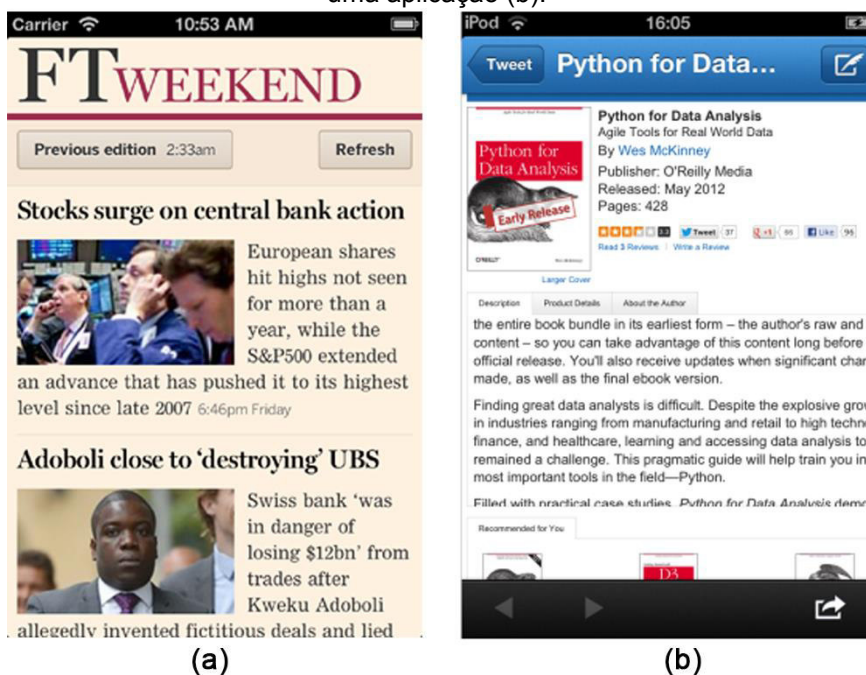
HTML5 Web Apps

Algumas plataformas nos permitem criar experiências semelhantes a aplicativos, usando o navegador como plataforma de instalação, fornecendo uma experiência de tela cheia baseada no navegador. Enquanto o nome "aplicativo HTML5" também pode ser usado para todos os aplicativos da *web* que podem ser executados em um navegador. Normalmente, esta categoria envolve a instalação através do próprio navegador ou de uma loja acessada através do navegador. Os aplicativos *Web* HTML5 são hospedados em um servidor *web* e distribuídos através do navegador.

iOS Web App

Em dispositivos iOS, pode-se atualizar um *site* responsivo para um aplicativo HTML5 em tela cheia. Com esta técnica, o usuário pode instalar um ícone na tela inicial, compartilhando o espaço com os aplicativos nativos. Quando aberto, em vez de ser apenas um atalho para um *website*, o usuário terá uma experiência em tela cheia e nenhuma interface do navegador será fornecida. Mas alguns comportamentos são diferentes do navegador e para um bom ajuste, são necessários muitos testes. A Figura 9 (a) mostra um exemplo de uma aplicação *web* iOS.

Figura 9 - Aplicação web iOS de *The Financial Times*, sem interface do navegador (a); *WebView* de uma aplicação (b).



Fonte: (FIRTMAN, 2010).

WebViews

As plataformas nativas incluem um controle ou componente que permite que o conteúdo da *web* seja incorporado dentro de uma aplicação nativa, que é genericamente conhecido como uma *WebView* (visualização da *web*), como pode ser vista na Figura 9 (b).

Pseudo Navegador

Um pseudo-navegador é um aplicativo nativo comercializado como um navegador *web* que, em vez de fornecer seus próprios mecanismos de renderização e execução, usa a visualização *web* nativa. Do ponto de vista do usuário, é um navegador, mas do ponto de vista de um desenvolvedor, é apenas a visualização da *web* com uma interface de usuário específica.

4 DESENVOLVIMENTO HÍBRIDO

Hoje, no cenário de desenvolvimento de aplicativos móveis híbridos, basicamente existem três tipos de desenvolvimento, dois tipos de produto final, e diversas ferramentas de desenvolvimento, conforme esquematizados na Tabela 1.

Tabela 1 - Tecnologias para desenvolvimento híbrido de aplicativos

Tipo de Desenvolvimento Híbrido	Tipo de Aplicativo Produzido	Frameworks/IDEs
Web-Based	Híbrido	Apache Cordova e afins
Cross-Compiled	Nativo	Xamarin
JavaScript RunTime	Nativo	Appcelerator Titanium, React Native, NativeScript, etc.

Fonte: Compilado de NATIVESCRIPT (2016), CORDOVA (2016), XAMARIN (2016) e FIRTMAN (2010).

Segundo Firtman (2010), um dos tipos de aplicativo híbrido é um aplicativo desenvolvido usando tecnologias *web* (HTML, CSS, JavaScript), mas é empacotado e compilado como um aplicativo nativo para que possa ser distribuído em lojas de aplicativos nativos. Normalmente usa uma visualização *web* (WebView) em tela inteira como o contêiner de todo o aplicativo.

Uma vertente a esse método é o desenvolvimento de um aplicativo nativo, usando algumas camadas nativas e algumas camadas *web* ao mesmo tempo. Utiliza-se como componentes nativos da interface do usuário com apenas algumas partes em uma visualização *web*, com controle JavaScript, como foi o caso do aplicativo do Facebook em uma de suas primeiras versões (FIRTMAN, 2010).

Um segundo tipo de desenvolvimento híbrido é aquele que possui desenvolvimento multiplataforma, mas produz aplicativos realmente nativos, sem uso de WebViews. Durante a compilação ocorre uma interpretação da linguagem desenvolvida para a linguagem nativa, gerando o aplicativo inteiramente em código nativo (NATIVESCRIPT 2016) (XAMARIN, 2016).

Durante algum tempo, os aplicativos nativos do Facebook para iOS e Android tiveram alguns dos seus componentes implementados com HTML5 e utilizando a WebView nativa para a experiência de navegação principal. Depois de um tempo, encontrando alguns problemas de desempenho, o Facebook decidiu migrar algumas telas baseadas em WebView (como a linha de tempo principal) para

o código nativo completo, melhorando duas vezes o desempenho. Para provar que um aplicativo HTML5 era bom o suficiente, tanto quanto o nativo, alguns meses depois, os desenvolvedores do Sencha Touch², de forma autônoma, desenvolveu o aplicativo *web* Fastbook baseado em HTML5, usando técnicas de otimização, afim de comparar os desempenho e viram que esse último atingiu os mesmos níveis de desempenho e capacidade de resposta em relação ao aplicativo nativo (FIRTMAN, 2010).

Firtman (2010) afirma que problemas como o do aplicativo do Facebook acontecem quando se toma uma abordagem de desenvolvimento de *site* para construir um aplicativo e muitas vezes não usam as ferramentas e arquiteturas certas para o desenvolvimento.

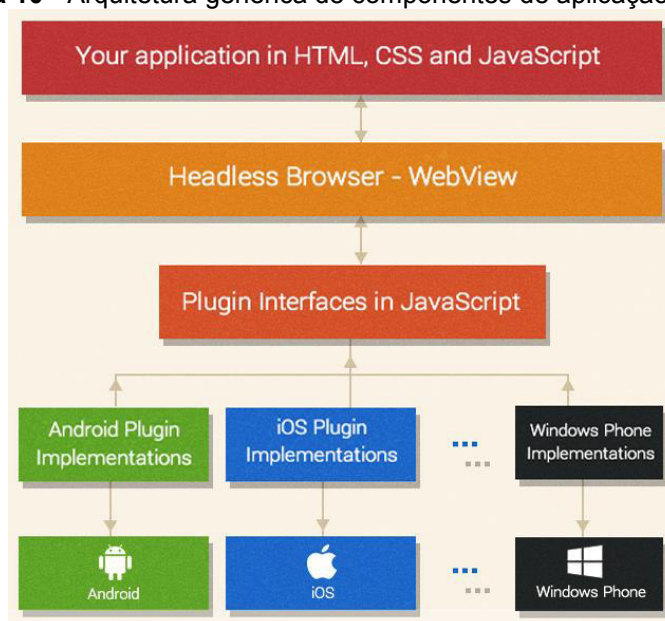
Para a maioria dos aplicativos, é necessário acessar as funções de *hardware* e isso não é possível através de tecnologias *web*. Essas estruturas multiplataforma se baseiam nos componentes do WebView de cada sistema. No iOS, este componente é representado pela classe *Objective-C UIWebView*. No Android, é chamado de *android.webkit.WebView*.

4.1 Arquitetura

De forma geral, os *frameworks* para desenvolvimento híbrido apresentam a arquitetura demonstrada na Figura 10. As aplicações sempre são instaladas como aplicações nativas. Isso significa que a construção de seu código produzirá um arquivo com extensão IPA para iOS, um arquivo APK para o Android, um arquivo XAP para Windows Phone, etc. Se a interface for bem trabalhada, os usuários podem nem sequer perceber que eles não estão usando um aplicativo nativo.

² Sencha Touch é um *framework* UI para aplicativos. É baseado em HTML5, CSS3 e JavaScript. Fornece aparência nativa aos aplicativos de diversas plataformas (SENCHA, 2016).

Figura 10 - Arquitetura genérica de componentes de aplicação híbrida



Fonte: (FIRTMAN, 2010)

4.2 Frameworks e IDEs

Os *frameworks* (arcabouços) são estruturas definidas e especializadas em uma funcionalidade ou finalidade de uso. Eles não devem ser utilizados como ferramentas utilitárias, mas como base para o desenvolvimento de aplicações.

Entre os diversos *frameworks* para construção de aplicativos multiplataforma baseados na *web*, destaca-se o Apache Cordova. Esse pode ser considerado como *framework* pai, pois ele é associado a diversos outros *frameworks* e ferramentas como: PhoneGap, Sencha Touch, Kendo UI, Intel XDK, IONIC, entre outros.

Como mencionado anteriormente, existem *frameworks* que permitem o desenvolvimento híbrido, mas produzem aplicativos nativos. Estes não fazem compilação cruzada, ou seja, a linguagem de programação será convertida para código nativo quando o aplicativo for compilado, para que o aplicativo possa ter melhor desempenho do que se estivesse usando WebView diretamente.

Dessa categoria, as mais conhecidas ferramentas são: NativeScript, Appcelerator Titanium, React Native e Xamarin. No caso do NativeScript, permite o desenvolvimento nas linguagens Angular.JS, TypeScript ou JavaScript (NATIVESCRIPT, 2016). O Titanium e o React Native utilizam JavaScript.

O Xamarin possui um diferencial, pois para cada tipo de plataforma, permite desenvolver a interface com código nativo através dos SDKs. Os controladores de eventos, são implementados em linguagem C#. Ainda no Xamarin, outra forma de criar os *layouts* é utilizando o Xamarin.Forms, que é um conjunto de ferramentas multiplataforma para interfaces, que permite aos desenvolvedores criar *layouts* de interface de usuário nativo que podem ser compartilhados entre Android, iOS e Windows Phone.

Com o Xamarin, o código C# para iOS é compilado para um binário ARM nativo. A compilação para Android produz um código de linguagem interpretada (IL). O código IL é executado diretamente no topo do *kernel* do Linux, e não através da máquina virtual Dalvik. Dessa forma, obtém um desempenho aproximado ao de um nativo puro (XAMARIN, 2016).

A seguir serão mencionadas algumas das ferramentas e tecnologias mais utilizadas no desenvolvimento híbrido e que também foram utilizadas no estudo de caso deste trabalho.

4.2.1 Apache Cordova

O Apache Cordova é um *framework* de aplicativo *web* nativo de código aberto multiplataforma, que é conhecido no mercado por causa de uma de suas implementações: o PhoneGap da Adobe.

Com ele é possível desenvolvimento de aplicativos utilizando as tecnologias de desenvolvimento *web* HTML, CSS e Javascript. Ele fornece um conjunto de APIs para acesso à funções nativas do Sistema Operacional e do *hardware* do dispositivo, utilizando Javascript.

Esse conjunto de APIs foram criadas para acessar as funcionalidades que são comuns em diferentes *smartphones* como eventos de ciclo de vida do aplicativo, armazenamento (armazenamento local do HTML5 e bancos de dados), contatos, câmera, geolocalização, bússola, acelerômetro, etc.

O Cordova oferece vários eventos para ser usados pela aplicação. Entre eles, podemos destacar:

- a) *OnStart()* - também conhecido como *deviceready*, é o evento essencial para qualquer aplicação. Ele sinaliza que as APIs de dispositivo do Cordova foram carregadas e estão prontas para acessar.

- b) *onResume()* - evento acionado quando a plataforma nativa retorna o aplicativo que estava em segundo plano.
- c) *onOnline()* - acionado quando um aplicativo é conectado à internet.
- d) *onBackKeyDown()* - acionado quando o usuário pressiona o botão Voltar.

Aplicativos que usam o Apache Cordova se tornam aplicativos compatível com a plataforma e SDKs desejadas, e podem ser disponibilizados para instalação na loja de cada plataforma. Vários *frameworks* possuem um compilador Cordova baseado em nuvem, portanto, o desenvolvedor não precisa lidar com o SDK nativo no próprio computador.

Ao usar as APIs do PhoneGap, um aplicativo pode ser construído sem qualquer código nativo da plataforma (Java, Objective-C, C#, entre outros). Em vez disso, tecnologias *web* são utilizadas, e eles são hospedados no próprio dispositivo localmente, o código é compilado junto ao código nativo, dispensando a necessidade de um servidor HTTP remoto para hospedar o código *web*.

As APIs JavaScript são consistentes em várias plataformas e construídas sobre padrões *web*, com isso o aplicativo pode ser portátil para outras plataformas de dispositivos com o mínimo de alterações. O *framework* está disponível para as seguintes plataformas: iOS, Android, Blackberry, Windows Phone, Palm WebOS, Bada e Symbian. A Figura 11 mostra quais APIs estão disponíveis para cada dispositivo.

Figura 11 - Compatibilidade de *plug-ins* Cordova com diversos dispositivos

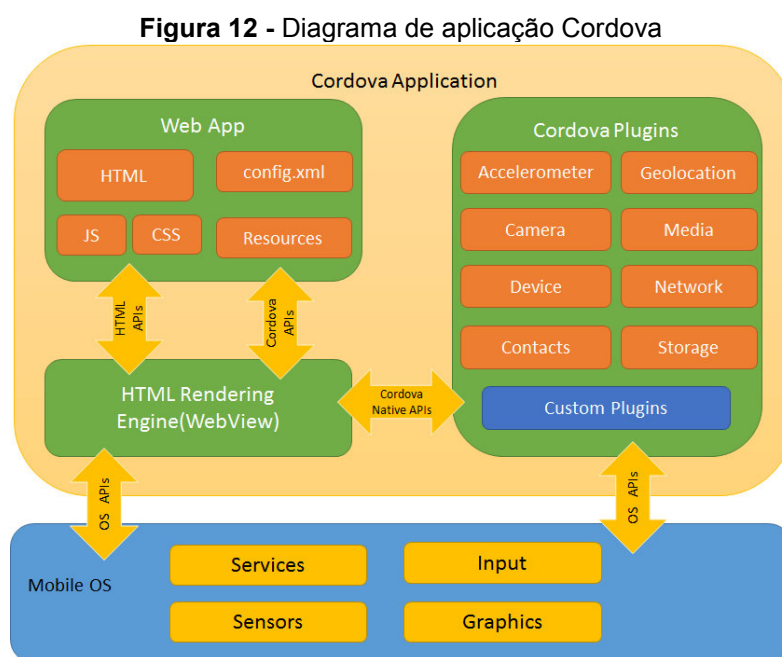
	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 5.x	Blackberry OS 6.0+	WebOS	Windows Phone 7 + 8	Symbian	Bada
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compass		✓	✓			✓	✓		✓
Contacts	✓	✓	✓	✓	✓		✓	✓	✓
File	✓	✓	✓	✓	✓		✓		
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓				✓		
Network	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓		

Fonte: (CORDOVA, 2016).

Com o Apache Cordova, é possível as seguintes estratégias de desenvolvimento:

- a) Estender um aplicativo em mais de uma plataforma, sem precisar reimplementar a linguagem e utilizar o conjunto de ferramentas de cada plataforma.
- b) Empacotar um aplicativo *web* para ser distribuído como aplicativo nativo nas lojas de aplicativos.
- c) Misturar componentes de aplicativos nativos com um WebView, sem perder o acesso às funcionalidades de *hardware*.
- d) Desenvolver uma interface de *plug-in* entre componentes nativos e WebView.

Existem vários componentes para uma aplicação Cordova. O diagrama presente na Figura 12 mostra uma visão de alto nível da arquitetura de um aplicativo Cordova.



Fonte: (CORDOVA, 2016).

Com o Cordova, o aplicativo é implementado como uma página da *web*. Por padrão a interface é definida por um arquivo local chamado `index.html`, que faz referência a código CSS, código JavaScript, imagens, arquivos de mídia ou outros recursos que são necessários para que ele seja executado. O aplicativo é executado em uma WebView dentro de um contêiner de aplicativo nativo, que pode ser

distribuído nas lojas de aplicativos. Toda a programação dos eventos do aplicativo é feita em JavaScript.

Os *Plug-ins* são parte integrante do ecossistema Cordova. Eles fornecem uma interface para os componentes nativos se comunicarem uns com os outros e servem para fazer ligações às APIs padrões do dispositivo. Isso permite que seja possível invocar código nativo do JavaScript. Existe, também, um conjunto de *plug-ins* chamado Core Plug-ins. Esses *plug-ins* permitem que o aplicativo acesse recursos do dispositivo, como bateria, câmera, contatos, etc.

4.2.2 Intel XDK

Para o desenvolvimento do estudo de caso deste trabalho, foi utilizado o Intel XDK, que é um ambiente que permite o desenvolvimento de aplicativos multiplataforma com HTML5 e Apache Cordova. O Intel XDK é baseado em tecnologias web (HTML, CSS, JavaScript e *back-end* Node-Webkit) e é compatível com Windows, OS X e Linux Ubuntu. É possível desenvolver aplicativos HTML5, aplicativos HTML5 + Cordova e aplicações para Internet das Coisas.

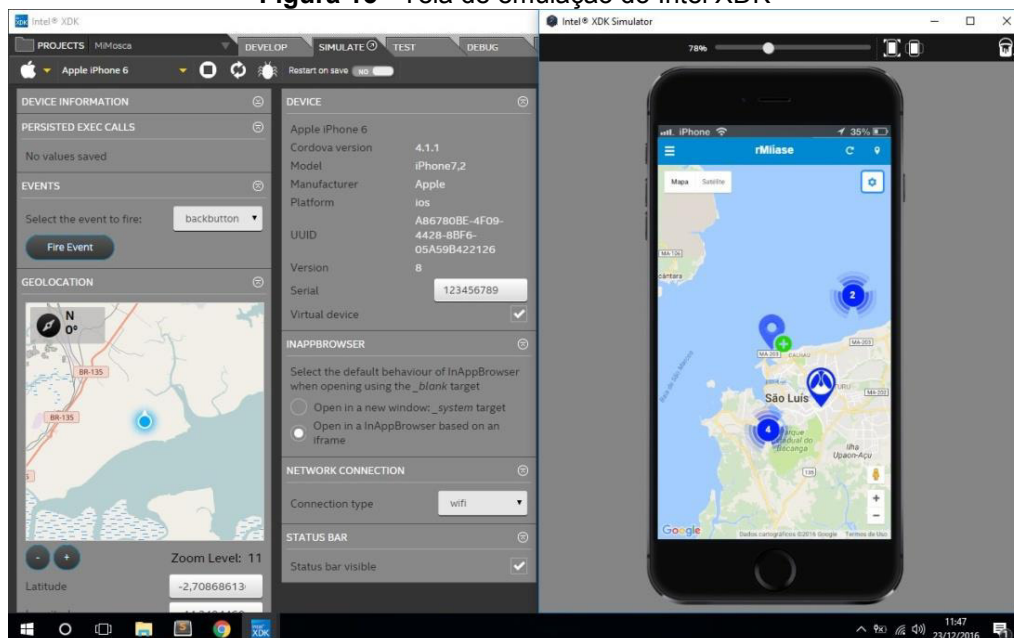
É possível adicionar *web services* e serviços de monetização do Google pela própria ferramenta. No Intel XDK existem vários *frameworks* de interface de usuário, como AppFramework, JQuery Mobile, Twitter Bootstrap, Ionic, Ratchet, Framework7, Topcoat, etc.

Para fins de aceleração do processo, no ambiente existe o App Designer, que é um construtor de interface intuitivo. Com ele é possível utilizar a ferramenta *drag-and-drop* de *layout* de interface com o usuário. A edição de elementos de interface é integrada ao JavaScript, ou seja, permite a edição personalizada do código JavaScript dos elementos da interface com o usuário, provendo os eventos desses elementos. O Intel XDK, apesar de já possuir um editor de texto, permite usar qualquer outro para edição dos arquivos de projeto. Possui suporte ampliado para API de dispositivo, através do Apache Cordova e suporte de *plug-ins* Cordova básicos ou de terceiros.

A Figura 13 apresenta o emulador do Intel XDK, por meio do qual é possível testar os aplicativos em desenvolvimento. O emulador se adapta, em tempo real, às alterações feitas no código do projeto e simula vários eventos de *hardware*. A IDE também permite testar o aplicativo em dispositivos por meio do Intel App

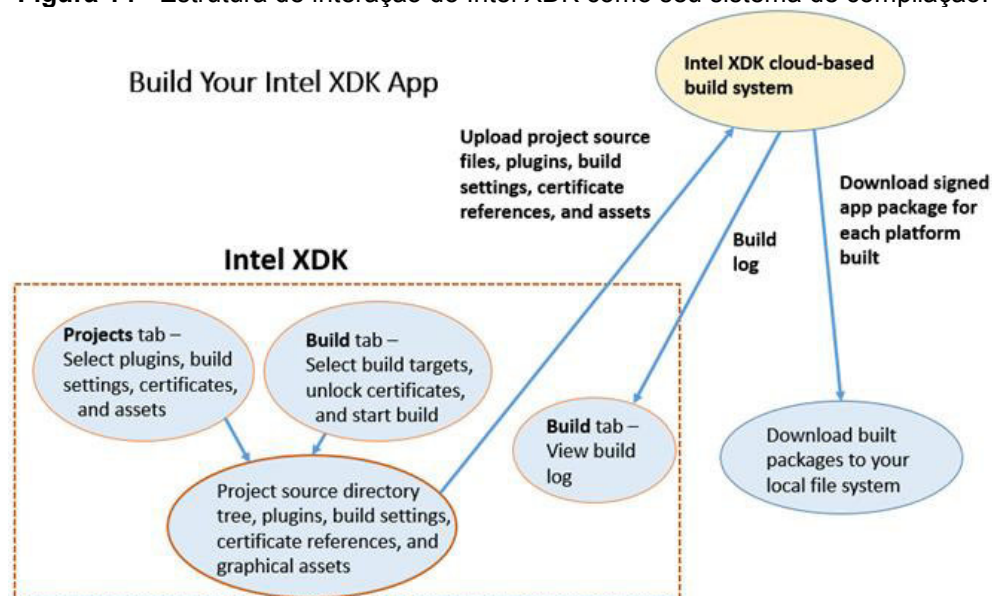
Preview, via Wi-Fi ou USB. Para gerar os arquivos binários do aplicativo para os diversos sistemas operacionais, os arquivos do projeto são enviados para um sistema de compilação baseado em nuvem, conforme é descrito na Figura 14.

Figura 13 - Tela de emulação do Intel XDK



Fonte: Produzido pelo autor.

Figura 14 - Estrutura de interação do Intel XDK como seu sistema de compilação.



Fonte: (XDK, 2016).

Depois de desenvolvido, o aplicativo, híbrido ou *web*, pode ser distribuído nas seguintes lojas de aplicativos através da própria IDE: Apple App Store, Google

Play, Amazon Store e Windows Store e Chrome Web Store. O aplicativo também pode ser publicado em outras lojas, utilizando apenas o arquivo binário gerado. O desenvolvedor pode escolher também a versão de API mínima aceitável de acordo com as versões de plataforma que deseja aceitar.

4.2.3 Frameworks UI

A seguir, os frameworks que foram utilizados para construção de interfaces de usuário no estudo de caso.

4.2.3.1 Bootstrap

O Twitter Bootstrap é um *framework* que fornece um conjunto de classes CSS e funções JavaScript para o processo de desenvolvimento *front-end*. As suas características de design responsivas permitem o suporte para *displays* móveis e *desktop*. Os *sites* desenvolvidos são compatíveis com vários navegadores e dispositivos, funcionando bem em monitores *desktop* e móveis. O desenvolvedor não precisa trabalhar com o CSS para tornar o *site* atraente ou apoiar princípios de projeto responsivos, a menos que seja necessário (BOOTSTRAP, 2016).

4.2.3.2 AppFramework

AppFramework é uma biblioteca JavaScript para desenvolvimento de aplicativos HTML5 para dispositivos móveis. Ela permite desenvolver interfaces para Android e iOS. Esse *framework* é composto por três elementos: uma biblioteca de consultas e gerenciamento de eventos, uma biblioteca de interface gráfica e um *plug-in* para a biblioteca WebKit.

Mesmo inspirado no jQuery, a biblioteca é funcionalmente mais rica do que o jQuery Mobile. Outro ponto forte do AppFramework é que ele requer apenas 3kB de memória contra 35kB para o jQuery. Os scripts são três vezes mais rápidos do que o jQuery no Android e 2.2 vezes mais rápido no iOS.

Existem vários *frameworks* concorrentes para AppFramework que possuem quase as mesmas funcionalidades. Entre os mais conhecidos estão: jQuery Mobile, Sencha Touch, jQTouch e Framework7. O peso e a velocidade de

execução do AppFramework continuam ser a maior vantagem em relação a esses outros (APPFRAMEWORK, 2016).

4.3 Discussões sobre desempenho

4.3.1 Compilação e Interpretação

Antes de adentrar a questões de desempenho propriamente, é preciso falar um pouco sobre linguagens de programação. As linguagens utilizadas são projetadas para adotar uma sintaxe de alto nível, que pode ser mais facilmente interpretadas por humanos. As linguagens de alto nível são formas para que programadores possam escrever programas mais organizados e com maior rapidez. Elas também tornam os programas menos dependentes de ambientes computacionais específicos, isto acontece porque, programas escritos em linguagens de programação são traduzidos para o código de máquina do computador no qual será executado. Uma linguagem de programação pode ser convertida em código de máquina por compilação ou interpretação.

Se o método utilizado traduz todo o código fonte do programa, para só depois executar o programa, então diz-se que o programa foi *compilado* e que o mecanismo utilizado para a tradução é um compilador. A versão compilada do programa é armazenada, de forma que o programa pode ser executado um número indefinido de vezes sem precisar nova compilação, o que compensa o tempo gasto na compilação. Isso acontece com linguagens como Pascal, C, etc.

Se o código do programa é traduzido à medida que vai sendo executado, como em Javascript, Python ou Perl, num processo de tradução de trechos seguidos de sua execução imediata, então diz-se que o programa foi *interpretado* e que o mecanismo utilizado para a tradução é um interpretador. Programas interpretados são geralmente mais flexíveis, já que podem interagir com o ambiente mais facilmente. Mas são geralmente mais lentos do que os compilados. Ainda há linguagens compiladas para um código de uma máquina virtual como o Java e o Parrot.

Uma técnica de interpretação é o Just in Time, que é quando o código-fonte têm módulos compilados para a memória de acordo com as necessidades, ao

invés de ser interpretado linha a linha. Dessa forma, aumenta a performance dos programas quando os mesmos módulos são chamados várias vezes.

Chegando à abordagem sobre desempenho, Charland e Leroux (2011), em seu trabalho apresenta os aplicativos híbridos como um desenvolvimento mais barato do que os nativos. Embora a experiência do usuário nos aplicativos nativos seja melhor, um aplicativo híbrido pode apresentar desempenho satisfatório, fazendo com que o desenvolvimento híbrido compense ao final. Além de que, com o avanço do poder de processamento e a evolução do HTML5 e JavaScript, a diferença entre nativo e híbrido está se tornando algo imperceptível.

É preciso considerar que a tecnologia de máquina virtual JavaScript é a nova linha de frente para as guerras de navegadores. Microsoft, Google, Apple, Opera e Mozilla estão trabalhando muito para superar as implementações concorrentes, fazendo com que o JavaScript seja executado com maior rapidez possível.

Trabalhar com linguagens nativas traz mais custo de implementação e manutenção. O que no JavaScript pode ser executado em vários dispositivos, talvez com apenas alguns ajustes por plataforma.

Um dos novos interpretadores de JavaScript é o Node.js que funciona do lado do servidor. Seu objetivo é ajudar programadores na criação de aplicações de alta escalabilidade. O Node.js é baseado no interpretador V8 JavaScript Engine, construído para obter melhor desempenho com um menor custo de CPU e, portanto, maior duração da bateria (TEIXEIRA, 2012).

Todos os recursos presentes no Node.js, e também a maioria das bibliotecas feitas para ele, adotaram um padrão não obstrutivo de escrever código. A Figura 15 demonstra um exemplo simples de como criar um código não obstrutivo. Trata-se de um trecho de um programa em JavaScript que escreve duas frases no terminal. Nesse exemplo foi criada uma função chamada *carregaFrase()*, cujo objetivo é ler uma determinada frase de uma fonte de dados, e uma outra função chamada *imprimeFrase()*, que imprime o valor de uma determinada variável no console. Como depende-se da leitura da frase na fonte de dados para imprimir o valor, passa-se a função que imprime como parâmetro para a função de leitura para que execute-se essa função quando a leitura for concluída. Esse tipo de função que é passada como parâmetro dessa maneira é chamada de *callback*.


Ao executar este exemplo com Node.js, ou qualquer mecanismo JavaScript, percebe-se que a segunda frase será impressa antes da primeira, isso se deve ao fato de que a execução da segunda frase não depende de nada. Já a execução da primeira frase depende de uma operação que leva 3 segundos.

Figura 15 - Trecho de código em JavaScript e o resultado de sua interpretação no console.

```

5   carregaFrase = function (callback) {
6     setTimeout(function() {
7       //Simula leitura da frase no banco de dados.
8       frase = "Minha frase obstrutiva";
9       callback();
10    }, 3000)
11  }
12
13  imprimeFrase = function () {
14    console.log(frase);
15  }
16
17  carregaFrase(imprimeFrase);
18
19  console.log("Olá");

```



Output	Location
Olá	ex1Nodejs.html:19
Minha frase obstrutiva	ex1Nodejs.html:14

Fonte: produzido pelo autor.

4.3.2 Interface e Experiência de Usuário

Charland e Leroux (2011) afirma que, de um modo geral, uma experiência de usuário é influenciada por dois aspectos principais: o **contexto** e a **implementação**.

No **contexto**, há elementos que devem ser entendidos, mas não podem ser alterados ou controlados. Estes incluem *affordances*³ de *hardware*, capacidades de plataforma e *hardware*, convenções de interface do usuário e o ambiente em que seu aplicativo é usado. Este contexto de execução pode ser radicalmente diferente de um usuário para outro, mesmo em uma única plataforma.

³ Interpretado como "reconhecimento". É a qualidade de um objeto que permite ao indivíduo identificar sua funcionalidade sem a necessidade de prévia explicação, o que ocorre intuitivamente, por exemplo, uma maçaneta (BARBOSA 2010).

Na **implementação**, os elementos podem ser controlados em uma aplicação como a abordagem de desenvolvimento escolhida, as boas práticas de programação, o design de interface e a integração com recursos da plataforma, como dados de sensores ou notificações.

A performance das interfaces nos aplicativos nativos são, sem dúvida, superior. As três mais populares plataformas possuem paradigmas diferentes para implementação de interface de usuário e possuem também APIs diferentes para instanciar e acessar a interface. Já a plataforma *web* é consistente, e mesmo tendo o número de controles internos no SDK limitado, *frameworks* como o Apache Cordova, suprem essa falta.

Outra variável importante é a latência. Seja um aplicativo nativo ou *web*, há um custo de desempenho para consumir dados. Nessa transferência de dados, quanto menor a carga útil, mais rápido o aplicativo. Então usar dados de formatação de JSON tende a resultar em uma carga de dados menor em comparação com uma carga de XML equivalente, na maioria dos casos. Por outro lado, os dados XML podem ajudar na performance retornando fragmentos HTML, que devem ser inseridos em uma página da *web* ou em um WebView, em vez de retornar dados formatados em JSON que, embora sejam pequenos, precisam ser convertidos em um fragmento HTML usando JavaScript.

Para aplicativos móveis, a tecnologia *web* não atingiu o nível de desempenho que pode-se alcançar com o código nativo, mas está ficando próximo. Há um clima de confiança de que as tecnologias *web* se tornarão indistinguíveis das experiências nativas, um exemplo é o WebGL. Onde é possível rodar um jogo 3D no navegador (CHARLAND; LEROUX, 2011).

De acordo com White (2013), alguns tipos de aplicativos tendem a se dar melhor com o desenvolvimento nativo ou com híbrido. Baseado nisso, existem algumas considerações que podem ajudar na hora da escolha do tipo de desenvolvimento. Na Tabela 2, é demonstrada, de forma simplificada, a relação entre a característica dos aplicativos e a melhor indicação do tipo de desenvolvimento.

Tabela 2 - Comparativo entre aplicativos nativos e híbridos em relação às características da aplicação.

	Nativo(Android/iOS)	Híbrido	Melhor
Gráfico	APIs Nativas	HTML, Canvas, SVG	Nativo
Performance	Rápida	Média\Rápida	Nativo
Aparência nativa	Aparência Real	Emulado	Nativo
Recurso de hardware	Total	Muito	Nativo
Publicação nas lojas	Normal	Quase normal	Nativo
Reutilização de código	Baixa	Total	Híbrido
Custo de desenvolvimento	Alto	Médio	Híbrido
Tempo de desenvolvimento	Alto	Baixo	Híbrido
Facilidade de atualização	Médio	Fácil	Híbrido
Curva de aprendizado	Lenta	Média	Híbrido
Conhecimentos requeridos	Java, ObjectiveC e Swift	HTML5, CSS e Javascript	---

Fonte: Sintetizado a partir de White (2013).

5 ESTUDO DE CASO

Como estudo de caso de um aplicativo móvel híbrido, foi desenvolvido o rMífase, conforme já mencionado na introdução. O rMífase é um aplicativo híbrido do tipo Web-Based desenvolvido através da IDE Intel XDK, que utiliza o Apache Cordova. A interface de usuário foi desenvolvida à base do AppFramework.

5.1 Modelagem

Para iniciar o desenvolvimento, foi necessário definir os requisitos e elaboração dos diagramas que compõe a modelagem do sistema da aplicação.

5.1.1 Requisitos

O sistema da aplicação, como um todo, tem os seguintes requisitos:

- a) Oferecer uma interface pelo qual usuários possam ter acesso aos dados de registro de mífase. Esses casos devem ter informações básicas como o tipo de animal, endereço, parte do corpo lesionada, local de atendimento, localização geográfica e imagem do local afetado pela doença.
- b) Possuir um meio de registro do usuário de forma simples.
- c) Mostrar ao usuário através de um mapa os casos de mífase já cadastrados por todos os usuários.
- d) Permitir ao usuário o cadastro de casos de mífase estando *on-line* ou *off-line*.
- e) Os usuários só poderão editar os seus próprios casos cadastrados.
- f) Os usuários não poderão apagar casos cadastrados que já estão homologados.
- g) Só é permitido ao usuário ter acesso às funções de registro e edição de casos se ele estiver cadastrado.
- h) O processo de cadastro de usuário deve ser simples. Exige apenas nome, e-mail e telefone. Cada cadastro de usuário fica vinculado a um dispositivo móvel.

- i) Deverá haver um *website* de acesso restrito por usuário e senha para que os administradores do aplicativo possam homologar (confirmar se cada caso é mífase ou não) os casos registrados que tiverem a sua amostra analisada por laboratório.
- j) Os administradores poderão excluir casos e bloquear usuários.
- k) A previsão de número de usuários, em seis meses, não deve ultrapassar 150.
- l) A previsão de número de casos, em seis meses, não deve ultrapassar 600.

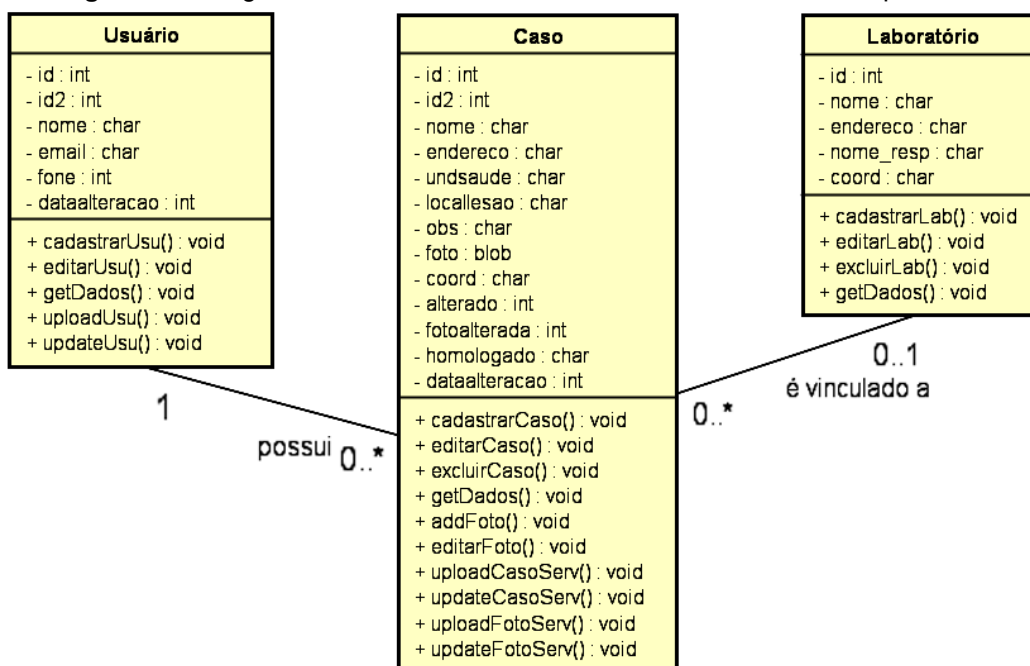
5.1.2 Diagramas

A seguir, serão apresentados os diagramas de classe, sequência e atividades para a melhor compreensão da estrutura do sistema.

5.1.2.1 Diagrama de Classe

A Figura 16 apresenta o diagrama de classe que representa as ligações, relacionamentos e a estrutura das classes que deram origem aos objetos utilizados no diagrama de sequência. Esse diagrama serve para prever os objetos que serão necessários no projeto, para a modelagem do sistema e para demonstrar com detalhes o conteúdo das classes, seus métodos e atributos. Além de servir de base para a construção do banco de dados, onde, basicamente, cada classe do diagrama representa uma tabela que será criada e cada atributo será um campo desta tabela.

Figura 16 - Diagrama de Classe relacionado ao banco de dados do aplicativo



Fonte: Produzido pelo autor.

Em todo o sistema da aplicação, existem dois bancos de dados: O banco do dispositivo móvel e o banco do servidor. Para fins de simplificação de desenvolvimento, a estrutura dos dois bancos é a basicamente a mesma.

O banco interno do dispositivo, cuja definição em JavaScript é apresentada no APÊNDICE A, possui uma tabela *Usuário*, que irá armazenar somente uma *tupla*, que é preenchida no cadastro do usuário no aplicativo. O cadastro do usuário é feito na primeira vez que o usuário quiser cadastrar um caso, ou na primeira vez que acessar o *menu* Perfil.

O banco de dados do dispositivo e o banco do servidor possuem basicamente a mesma estrutura. As tabelas *Caso*, *Usuário* e *Laboratório* possuem os atributos *id*, o *id2*. Que servem para tratar o sincronismo de dados. O esquema de sincronismo será detalhado na Seção 5.1.2.3.

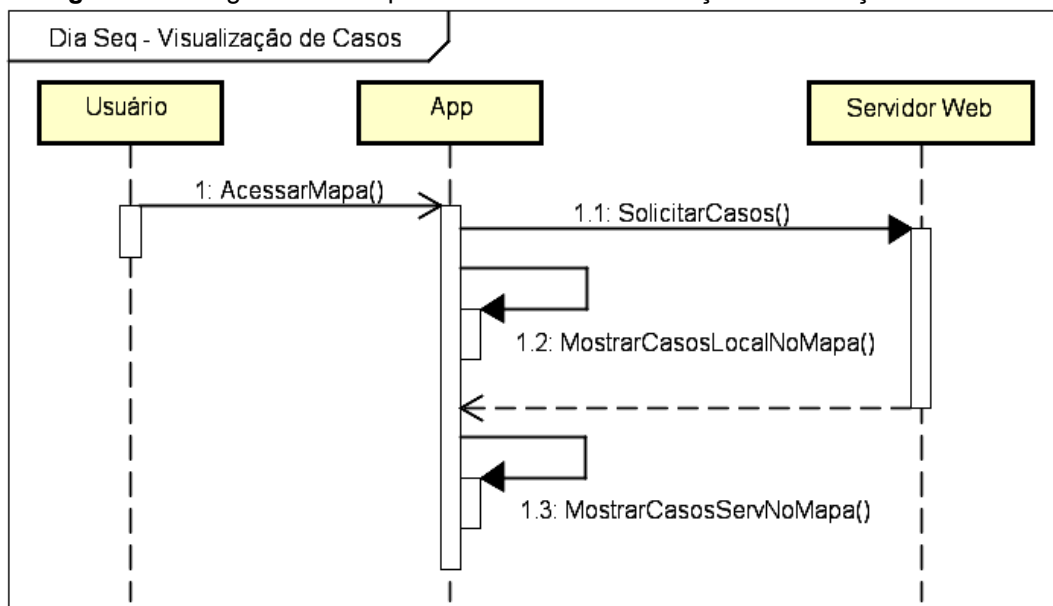
Caso haja a necessidade de se reestruturar o banco interno, no futuro será preciso realizar *scripts de transição de banco*⁴ de forma a garantir que os dados da versão antiga sejam transportados e reajustados para a nova versão, sem prejuízo ao usuário depois da atualização do aplicativo, que é feita através das lojas oficiais de aplicativos.

⁴ Uma estratégia para implementar transição de banco é a verificar a versão do banco, salvar todos os dados na memória RAM do aparelho, destruir o banco e depois criar um novo banco como a nova estrutura e inserir os dados antigos, já de forma reestruturada (sugestão do autor).

5.1.2.2 Diagrama de Sequência

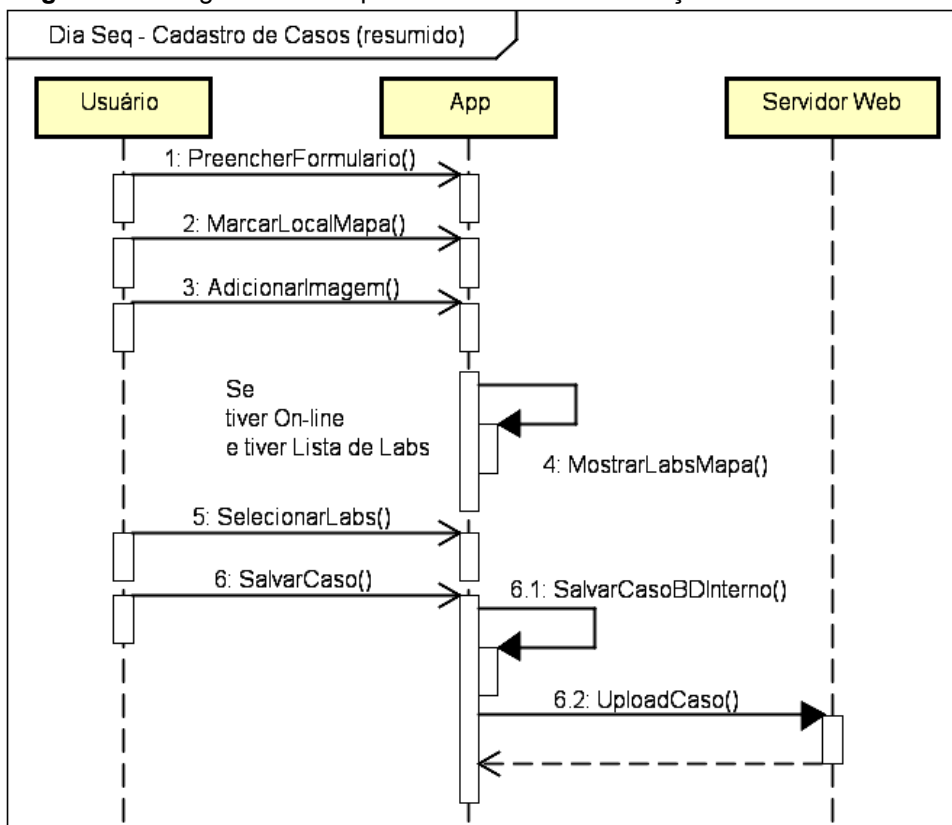
No diagrama de sequência é demonstrado como ocorre a sequência de interação entre os objetos da aplicação, as ações do usuário e as interações próprias do sistema, determinando de forma precisa a sequência de ações e eventos para a realização dos casos de uso. Os diagramas das Figuras 17 e 18 mostram as funções *Visualização de Casos* e *Cadastro de Casos*, respectivamente.

Figura 17 - Diagrama de Sequência descrevendo a função Visualização de Casos.



Fonte: Produzido pelo autor.

Figura 18 - Diagrama de Sequência descrevendo a função Cadastro de Casos.



Fonte: Produzido pelo autor.

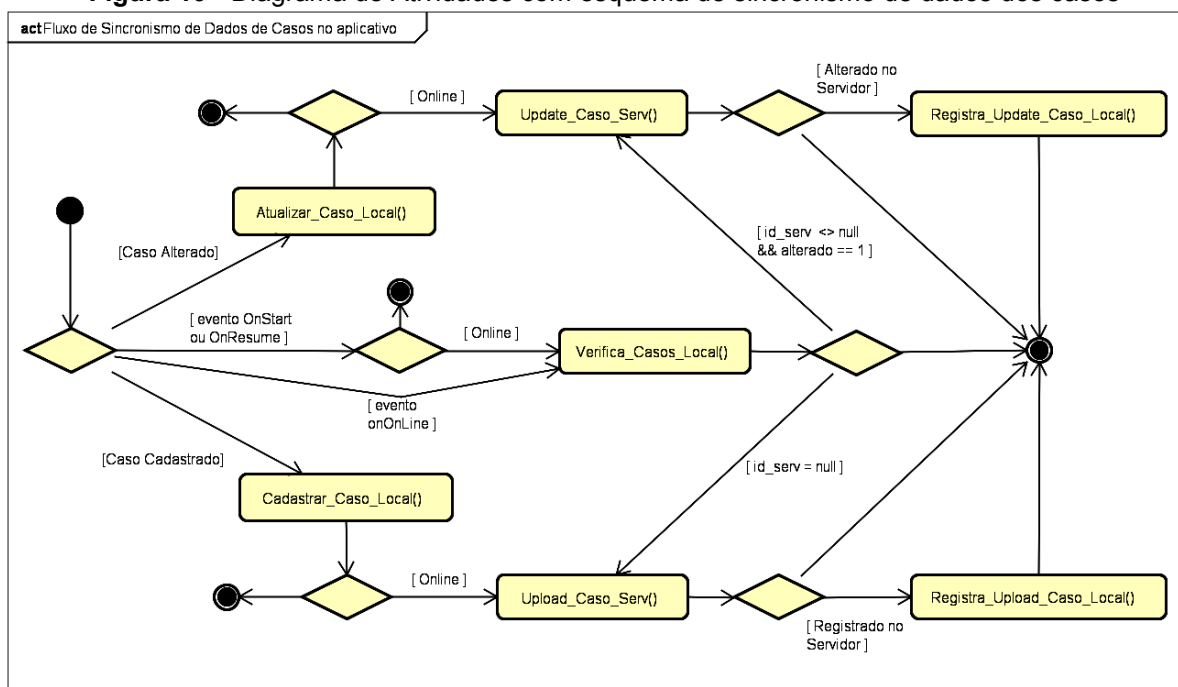
5.1.2.3 Diagrama de Atividades

Uma das questões mais difíceis para implementação é o sincronismo de dados, pois o aplicativo exige que todos os casos e o perfil de usuário possam ser cadastrados ou editados de forma *off-line* para que, assim que tiver conexão, os dados possam ser registrados ou atualizados no servidor.

De forma semelhante, a função de exclusão de casos. Assim que o usuário indicar que deseja excluir um caso, o aplicativo só poderá excluir o registro do aparelho e do servidor, após verificar no servidor se o caso não está homologado, ou não está em fase de homologação.

A Figura 19 apresenta o esquema de sincronismo de dados dos casos, a partir das funções de cadastro de casos, alteração de caso e da inicialização do aplicativo.

Figura 19 - Diagrama de Atividades com esquema de sincronismo de dados dos casos



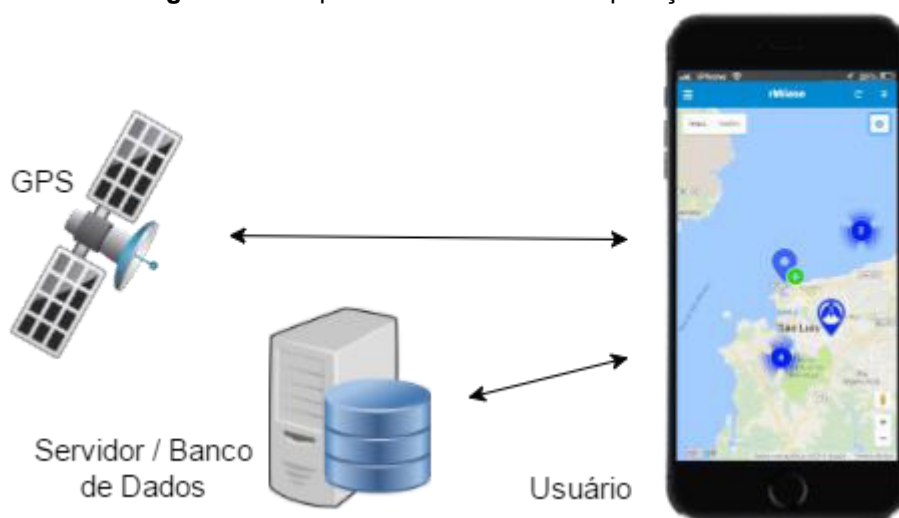
Fonte: Produzido pelo autor.

5.2 Desenvolvimento

5.2.1 Arquitetura da aplicação

Para atender os requisitos, levando em conta a complexidade do aplicativo e o fluxo de dados previsto, foi necessário implementar a arquitetura cliente-servidor. O lado cliente, representado pelo aplicativo móvel, solicita dados e serviços ao servidor *web*, que é responsável por executar as requisições de consulta, inserção, atualização e exclusão no banco de dados do servidor. O aplicativo conta com o sistema GPS para obter a localização geográfica do dispositivo. A Figura 20 apresenta o diagrama da arquitetura do sistema da aplicação.

Figura 20 - Arquitetura do sistema da aplicação.



Fonte: Produzido pelo autor.

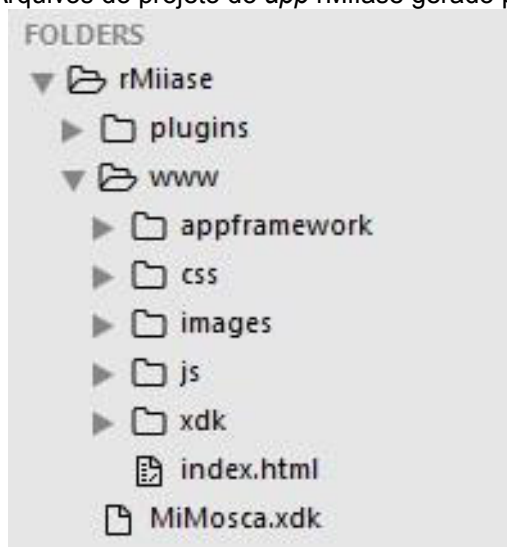
5.2.2 Ferramentas e tecnologias utilizadas

Foi utilizada a IDE Intel XDK que foi detalhada na Seção 4.2.2. Por meio desta ferramenta foi possível criar o projeto de aplicativo híbrido, com todos os *frameworks* de interface e *hardware* integrados. O modo App Designer, que utiliza a função *drag-and-drop* para construção de interfaces, não foi utilizado, pois algumas telas do aplicativo exigiam *layouts* de interface que não poderiam ser construídos pelos componentes de interface fornecidos pela ferramenta, como um botão em cima do mapa, por exemplo. Outra vantagem de se implementar a interface por meio de programação direta, é a maior liberdade de modificação, o que não é possível quando se usa o App Designer.

Os *plug-ins* Apache Cordova utilizados foram: Camera, Device, Dialogs, Geolocation, Inappbrowser, Network-information e Splashscreen. As APIs utilizadas foram Google Maps JavaScript API, Google Maps Geolocation API apresentadas na Seção 2.4 e um *web service* próprio detalhado na Seção 5.2.3.

O Intel XDK gera um projeto que é organizado da maneira como é apresentado na Figura 21. O padrão de organização do projeto sugere uma divisão dos arquivos de acordo com tipos específicos.

Figura 21 - Arquivos do projeto do *app rMiiase* gerado pelo Intel XDK.



Fonte: Produzido pelo autor.

Toda a estrutura da interface gráfica é construída por meio do arquivo `index.html` e estilizada por meio de CSS. Cada tela do aplicativo é definida pelo nome de classe `upage` do atributo `class` do elemento de objeto `div` do HTML. A Figura 22 mostra a definição, em HTML, da tela de Registro de Usuário, cujo resultado é a interface apresentada na Figura 23.

Figura 22 - Trecho de código-fonte HTML referente à tela de Registro de Usuário

```

<!-- Tela de Registro de Usuário -->

<div class="upage vertical-col left panel" title="Registro" id="pgRegister" data-footer="none">
  <div style="text-align:center">
    <br>
    <h2>Registre-se. É bem simples.</h2>
    </br>
  </div>

  <h2>Nome</h2><input id="nome" name="nome" type="text" placeholder="Nome Completo" />
  <h2>E-mail</h2><input id="email" name="email" type="text" placeholder="email@email.com" />
  <h2>Fone</h2><input id="phone" name="phone" type="number" placeholder="fone" />

  <a class="button block green" href="#" id="btRegister">Registrar</a>
</div>

```

Fonte: Produzido pelo autor.

Figura 23 - Tela de Registro de Usuário

Fonte: Produzido pelo autor.

5.2.3 Servidor da Aplicação

O servidor da aplicação, foi implementado em PHP com um banco de dados MySQL e, basicamente, possui a mesma estrutura de modelagem do banco interno do aplicativo móvel. O servidor *web* pode ser acessado por requisições via POST, que devolvem dados ou parâmetros de erro. Os tipos possíveis de requisições são detalhados na Tabela 3. Para fins de exemplificação, o APÊNDICE C demonstra a função responsável por realizar o *upload* do caso cadastrado a partir do dispositivo para o servidor *web*. E o APÊNDICE D apresenta o código PHP responsável por receber e responder as diversas requisições dos dispositivos.

Tabela 3 - Tipos de requisições feitas pela aplicação ao servidor

Parâmetro de requisição feita pelo aplicativo ao servidor	Execução feita pelo servidor
<i>uploadUsuario</i>	Recebe os dados de usuários de cadastra no Banco do Servidor
<i>uploadCaso</i>	Recebe os dados de casos de cadastra no Banco do Servidor
<i>apagarCaso</i>	Apaga caso do servidor
<i>verCasoHomologado</i>	Verifica se o caso já foi homologado (chamado antes de <i>apagarCaso</i>)
<i>updateUsuario</i>	Recebe os dados de usuários e atualiza no servidor
<i>updateCaso</i>	Recebe os dados de casos e atualiza no servidor
<i>downloadFoto</i>	Recebe o id do caso e devolve a imagem correspondente

Fonte: Produzido pelo autor.

Todos os casos cadastrados no servidor aparecem no mapa do aplicativo sob forma de marcadores. Os marcadores azuis são os casos cadastrados por outros usuários/aparelhos e os roxos são os "meus casos", marcados pelo próprio usuário.

Para que o aplicativo carregue todos os marcadores, ele acessa um *web service* próprio que fornece uma *string* no formato JSON com todos os dados dos casos. Da mesma forma para acessar a lista de laboratórios.

Foi desenvolvida uma API em PHP, que é responsável pela integração de dados, disponibilizando as informações do banco de dados para que tanto a plataforma do aplicativo móvel quanto o sistema de gerenciamento possam consumir via requisição GET. Esse serviço poderá fornecer dois tipos de informação: os dados (com exceção da imagem) dos casos cadastrados e os dados dos laboratórios cadastrados. Quando a API é acessada, consulta os dados no banco e monta um pacote JSON em um endereço *web*, permitindo que o aplicativo móvel capture esse pacote para listar os dados em um determinado momento. A implementação desse *web service* é apresentada no APÊNDICE B.

A requisição para acessar os dados da API é feita por meio de requisição Ajax⁵ tirando proveito de sua característica assíncrona e o carregamento de componentes isolados.

As imagens dos casos cadastradas por outros usuários não são carregadas junto com os marcadores do mapa. Se fosse assim, demandaria muito fluxo de dados do servidor, além de que o *smartphone* não suportaria uma quantidade grande de dados carregados na sua memória RAM. Diante disso, as imagens dos casos só serão visualizadas se o usuário acessar os detalhes do caso e clicar no botão específico para a visualização da imagem. O *download* e *upload* das imagens também são feitos através de requisições Ajax.

⁵ Asynchronous JavaScript and XML. Basicamente utiliza JavaScript, XML e HTML de forma dinâmica. Permite, de forma imediata, a validação de formulários, sugestão de preenchimento automático e demais troca de dados, sem precisar recarregar a página web inteira.

6 RESULTADOS

6.1 Aplicativo Móvel

Como produto do desenvolvimento, obteve-se um aplicativo nas versões para Android e iOS, disponíveis nas suas lojas oficiais, utilizando-se do mesmo design de interface gráfica para ambos. O aplicativo está em contínua melhoria, mas em pleno funcionamento. Está integrado com o *web service* e sendo monitorado por administradores através do sistema de gerenciamento.

Ter o aplicativo com suas funcionalidades básicas finalizadas e já disponível nas lojas oficiais tem suas vantagens pelo fato de oferecer ao desenvolvedor dados estatísticos acerca do uso do aplicativo, como o número de usuários ativos, os países dos usuários, as versões de SO utilizadas, o modelo dos *smartphones*, qual o tipo de conexão mais utilizada, os possíveis erros que ocorrem, etc, como apresentado na Figura 24. Além de ser possível disponibilizar versões Beta somente para um grupo de usuários específicos.



Fonte: Produzido pelo autor.

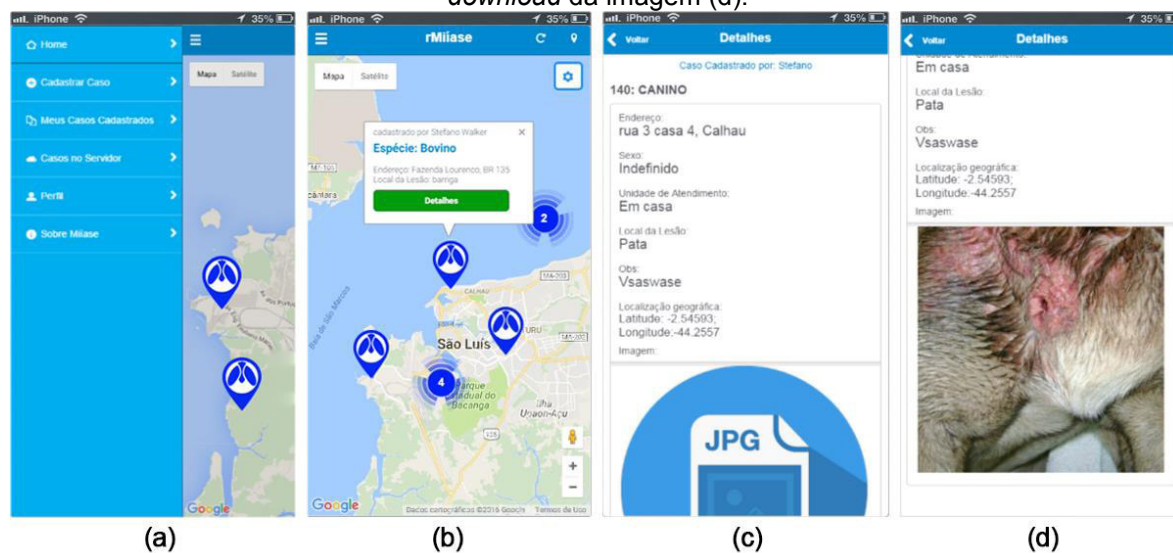
As imagens dos casos podem ser captadas através da galeria do dispositivo ou pela câmera. Como padrão, foi definido que a dimensão das imagens seria quadrada. O *plug-in Camera* do Apache Cordova possibilita que o aplicativo

utilize de seu dimensionador de fotos nativo para permitir ao usuário recortar a imagem logo depois da captura, seja da Galeria ou da Câmera. Outra configuração para as imagens é a qualidade de resolução, definida como 50% da resolução do aparelho. Esse parâmetro foi escolhido, após vários testes, para garantir que a imagem em aparelhos de diferentes configurações de *hardware* tenha qualidade razoável e pouco tamanho de armazenamento.

As imagens, no formato JPG, são armazenadas na galeria, no diretório padrão de armazenamento e, também, são armazenadas no banco de dados interno da aplicação no formato Base64⁶. A *string* em Base64 não é enviada ao banco de dados *web* junto com os dados do caso, é enviada de forma separada, pois possui um tamanho de armazenamento maior. O custo computacional dedicado para conversão de formato de arquivo compensou pelo fato de que, em Base64, é produzido um arquivo binário com compactação de aproximadamente 75%.

As Figuras 25, 26 e 27 apresentam telas do aplicativo desenvolvido.

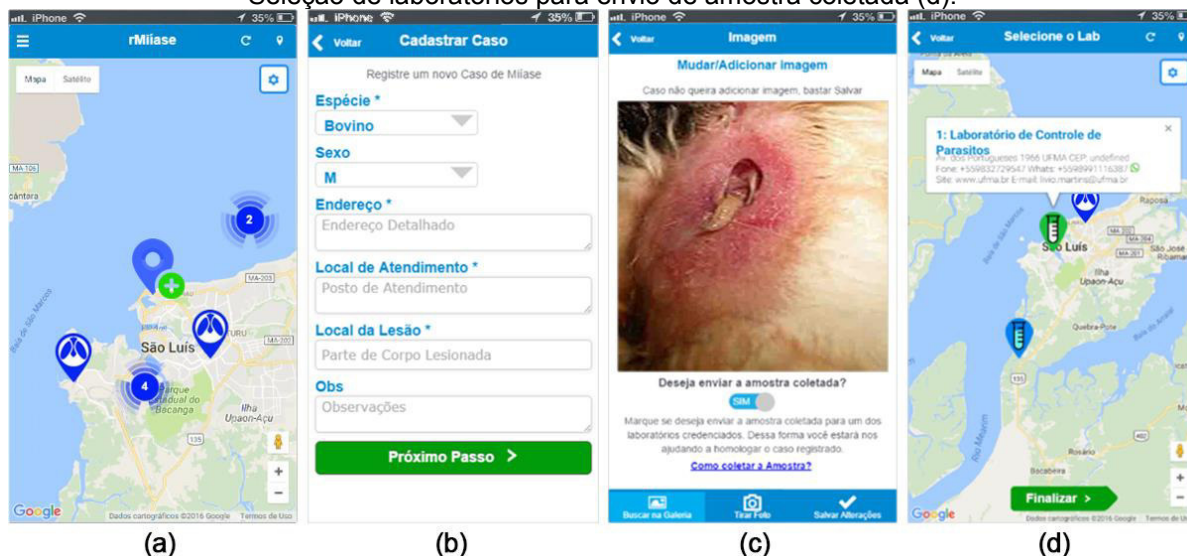
Figura 25 - Aplicativo rMifase: *Menu* (a); Tela inicial mostrando o mapa e os marcadores de casos cadastrados (b); Tela de Detalhes de um caso de outro usuário(c); Tela de Detalhes, após o *download* da imagem (d).



Fonte: Produzido pelo autor.

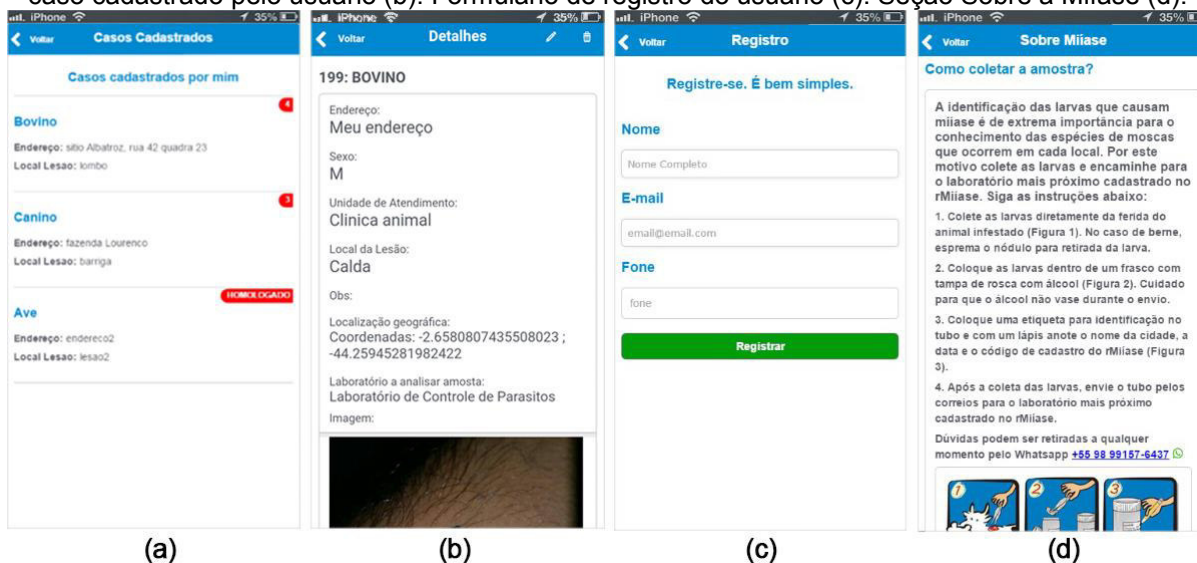
⁶ É um método para codificação de dados para transferência na Internet (codificação MIME para transferência de conteúdo). É utilizado frequentemente para transmitir dados binários por meios de transmissão que lidam apenas com texto.

Figura 26 - Aplicativo rMiase: Tela inicial com a adição do marcador de Cadastro de Caso (a). Formulário de Cadastro de caso (b). Captura de imagem da câmera ou galeria (c). Seleção de laboratórios para envio de amostra coletada (d).



Fonte: Produzido pelo autor.

Figura 27 - Aplicativo rMiase: Lista dos casos cadastrados pelo usuário (a). Tela de detalhes de um caso cadastrado pelo usuário (b). Formulário de registro do usuário (c). Seção Sobre a Miase (d).



Fonte: Produzido pelo autor.

Para o aplicativo móvel, foram desenvolvidas as seguintes funcionalidades básicas:

- a) Cadastro de Casos – Permite ao usuário registrar os casos de miíase preenchendo alguns dados e adicionar imagem. O usuário pode iniciar o cadastro de casos através do *menu*, ou através do mapa, adicionando um marcador de adição de caso.

- b) Edição de Casos – Edição de casos que o usuário cadastrou. Essa função é acessível pela lista de “meus casos” acessível pelo *menu*, ou pelo mapa, na tela principal.
- c) Exclusão de Casos – Permite ao usuário excluir os casos que cadastrou. Função acessível pela lista de “meus casos” ou pelo mapa.
- d) Visualização de Casos no Servidor – Permite ao usuário visualizar todos os casos cadastrados por todos os usuários do aplicativo, ver todas as informações, inclusive a imagem. Função acessível pelo mapa, na tela principal.
- e) Registro de Usuário – O usuário se registra fornecendo um nome, *e-mail* e número de telefone. Esses dados não são validados, servem apenas como um apelido para cada dispositivo.
- f) Informações e Contato – Área meramente informativa, acessada através do *menu* “Sobre”, onde o usuário encontra informações sobre a doença miíase, o objetivo do aplicativo, instrução sobre a coleta de amostras e acesso a página *web*.

6.2 Website e Sistema de Gerenciamento

A *homepage*⁷ (Figura 28), hospedada no servidor da UFMA, foi desenvolvida utilizando o *framework* para *front-end* Bootstrap. Essa página contém apenas informações acerca do projeto, links para *download* do aplicativo e informações de contato.

Juntamente foi desenvolvido o **sistema de gerenciamento**, que possui acesso restrito por *login* e senha. Os administradores desse sistema *web* podem exercer as seguintes funções: editar ou excluir os casos cadastrados por qualquer usuário do aplicativo; editar, excluir ou bloquear qualquer usuário; e gerenciar a lista de laboratórios credenciados. Ainda sobre os casos, o administrador poderá homologar os casos cadastrados, assim que a amostra enviada pelo usuário do *app* for analisada por um dos laboratórios.

O sistema de gerenciamento, também hospedado no servidor da UFMA, foi desenvolvido através de Phreeze, que é um *framework* para desenvolvimento de

⁷ Acessível através de <<http://www.rmiiase.ufma.br>>.

CRUDs⁸ em PHP. Por meio do Phreeze é possível gerar um sistema genérico completo no padrão MVC (*Model-View-Controller*) e ORM (*Object Relational Mapping*). Mas antes já é preciso definir o banco de dados da aplicação. O código da aplicação gerado pelo Phreeze é apenas um ponto de partida e pode ser alterado e customizado completamente (PHREZEE, 2016).

No sistema de gerenciamento *web*, o administrador poderá adicionar, excluir e editar a lista de Usuários, Casos e Laboratórios. A Figura 29 apresenta a tela da funcionalidade de edição de caso.

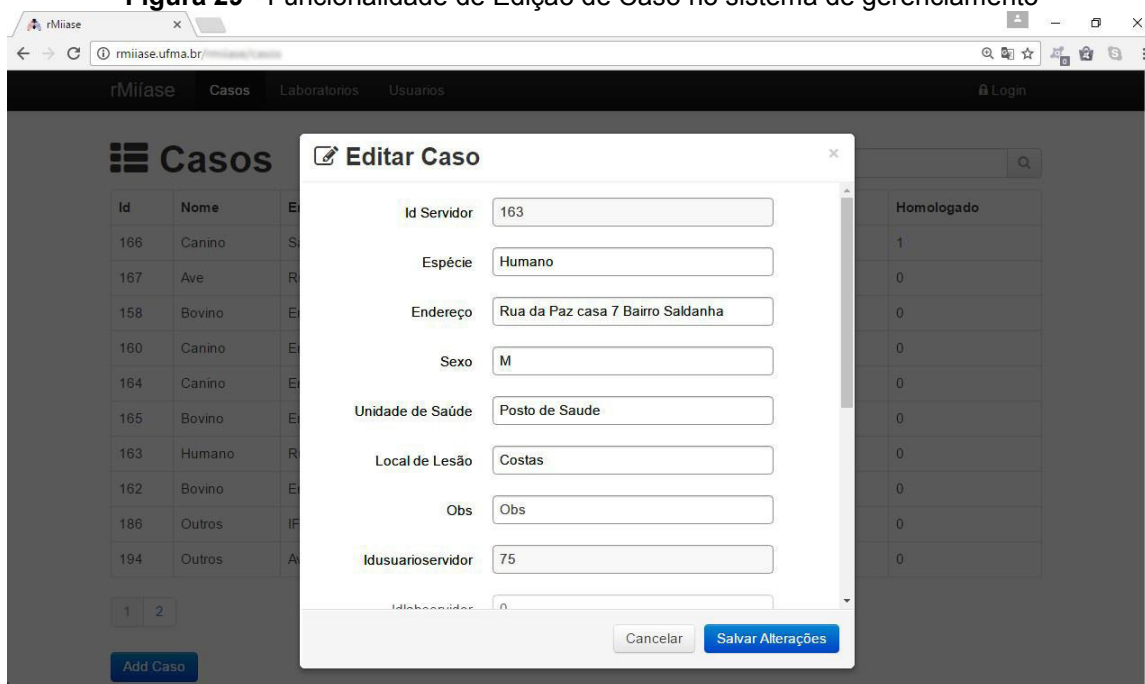
Figura 28 - Homepage do rMiíase



Fonte: Produzido pelo autor.

⁸ CRUD, na língua Inglesa, é acrônimo de Create, Read, Update e Delete, para as quatro operações básicas utilizadas em bases de dados relacionais.

Figura 29 - Funcionalidade de Edição de Caso no sistema de gerenciamento



Fonte: Produzido pelo autor.

6.3 Avaliação dos resultados

Os principais testes do aplicativo foram feitos na plataforma Android, no *smartphone* Samsung Galaxy A5 - 2015, que possui processador Quad Core de 1.2GHz, 2GB de RAM, tela de 5 polegadas, câmera traseira com 13Mpx capaz de capturar imagens com resolução de até 4128 x 3096 pixels.

Para a plataforma Android, o *app* foi produzido com retrocompatibilidade até a versão 4.0 (API 14), ou seja, no presente ano, o alcance é de aproximadamente 94% dos dispositivos com Android. Para iOS, alcança até versão 8.0, possuindo compatibilidade com 96,5%⁹ dos aparelhos iOS.

O aplicativo, nas versões para Android e iOS, se apresentou com um bom *layout* de interface gráfica com AppFramework, apesar de não possuir semelhanças com as interfaces nativas de ambas as plataformas. Ressalta-se que é possível, através de *frameworks* como o Framework7, fazer com que o um aplicativo híbrido tenha os componentes de interface semelhantes ao de um nativo.

É notável que a interface do rMiase não tem a rapidez e fluidez tanto quanto a das plataformas nativas. Uma característica que explica isso é o fato de que todas as telas da aplicação são carregadas na memória de uma só vez, embora

⁹ Dados fornecidos pela Play Store e App Store, nos portais dos desenvolvedores.

só apareça uma tela por vez. Já no Android, cada Activity pode ser administrada separadamente, podendo ser carregada na memória ou destruída. Para otimizar o desempenho da interface, pode-se, no futuro, utilizar outros *frameworks* UI e realizar testes de comparação de usabilidade.

Embora os arquivos HTML e CSS, que compõem a interface, não tenham todos juntos mais do que 100kB, outros componentes como o Google Maps e imagens carregadas todas juntas na memória RAM poderão causar sobrecarga de processamento, proporcionando uma má experiência ao usuário.

Outro fator que deixa os aplicativos híbridos levemente mais lentos são os *plug-ins* de controle de *hardware*, que não são melhores do que os acessos de forma nativa.

Ainda sobre a interface, em alguns dispositivos com configurações semelhantes ao Galaxy Pocket da Samsung, apresentaram alguns problemas nos componentes da interface, apresentando, às vezes, uma lenta resposta em alguns botões e *checkboxs*.

A captura de imagem através da câmera forneceu fotos com qualidade suficiente para a proposta e tamanho de armazenamento na faixa de 30 a 90 kB, o que é muito bom para facilitar a transferência de dados. Caso o usuário não queira ter sua foto compactada, a captura de imagem através da galeria não altera a qualidade da imagem.

Para acessar a página *web*, não é necessário sair do aplicativo. Por meio do *plug-in* Inappbrowser do Apache Cordova, o usuário pode acessar o *site* através de um navegador da própria aplicação.

O esquema de sincronismo de dados foi implementado conforme o diagrama de atividades da Figura 19 e consegue tratar a situação de ausência de conexão à internet. As rotinas de sincronismo são acionadas em todas as funções de cadastro e edição e, também, nos eventos *OnStart()* (inicialização), *onResume()* (retorno do segundo plano), ou *onOnline()* (conexão à internet).

De forma geral, o aplicativo apresentou um bom desempenho e usabilidade através de fluxos de atividade bem definidos, interface agradável e rapidez no processamento de dados, fazendo dele uma ferramenta confiável.

7 CONCLUSÃO

Com base no que foi apresentado, pode-se concluir que as tecnologias para o desenvolvimento de aplicações híbridas apresentaram uma evolução muito grande desde que começaram a serem utilizadas no desenvolvimento de aplicativos móveis. Grandes empresas de tecnologia continuam apostando no HTML5 e suas tecnologias afins como indispensáveis e poderosas em relação ao custo/benefício para desenvolvimento.

Foi também, objetivo desse trabalho, demonstrar que compensa investir no desenvolvimento híbrido *web-based*, pois o custo de produção manutenção e evolução da aplicação é muito menor. Aplicação híbrida apresenta performance levemente menor, mas que, em certos casos, é até desprezível como foi comprovado no estudo de caso. E, levando em consideração que o poder de processamento dos *smartphones* tende a crescer e as tecnologias *web* tendem a produzir softwares mais otimizados, pode-se concluir que o desenvolvimento híbrido ainda irá evoluir bastante.

Fica como sugestão de trabalhos futuros, o estudo sobre o desenvolvimento híbrido com ênfase em compilação nativa (utilizado por Xamarin, NativeScript, React Native, etc), pois se, de fato, os aplicativos nativos gerados pelo desenvolvimento híbrido alcançarem desempenhos equivalentes aos nativos puros, haverá uma verdadeira revolução para esse paradigma de desenvolvimento.

Outros melhoramentos fundamentais para o sistema da aplicação são os seguintes:

- Melhoramento do carregamento de casos cadastrados no servidor: Fazer com que não sejam carregados todos os casos do servidor de uma vez só, pois na maioria das vezes só interessará os casos cadastrados na região próxima ao usuário. Nesse caso, o *web service* deve ser modificado. Esse requisito se tornará obrigatório quando o número de casos aumentar.
- Visualização e edição da imagem do caso através do sistema de gerenciamento.

- No sistema de gerenciamento, visualização de um mapa com os casos, na mesma forma que aparecem no *app*, de forma a facilitar na localização e edição do caso.
- Experimentação de outros *frameworks* de interface gráfica, para melhoria de performance.
- Otimização do código, para contornar a característica assíncrona do JavaScript.
- Otimização do código da interface, visando o melhor uso da memória de execução e melhor modulação das telas da aplicação.
- Fornecimento de dados estatísticos da doença em forma de gráficos, consultas com filtro personalizado e exportação de relatórios para os administradores do sistema *web*.
- Links dentro do *app* para permitir o contato pelas redes sociais.
- Implantação de suporte a múltiplos idiomas.

Foi apresentado o projeto e implementação do sistema da aplicação que correspondeu bem às diretrizes de interface humano-computador. Com uma estrutura de implementação completa, o aplicativo rMífase, juntamente com seu sistema de gerenciamento *web*, poderá ser uma ferramenta útil para a coleta de dados seguros sobre a mífase, fornecendo dados estatísticos para o estudo da viabilidade de implantação de um projeto de controle da doença.

REFERÊNCIAS BIBLIOGRÁFICAS

APPLE, I. **iOS Develop**. [S.l.], 2016. Disponível em: <<https://developer.apple.com/develop/>>. Acesso em: 30 nov. 2016.

_____. **iOS Technology Overview**. [S.l.], 2016. Disponível em: <<https://developer.apple.com/library/content/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>>. Acesso em: 30 nov. 2016.

_____. **iPhone OS Programming Guide**. [S.l.], 2016. Disponível em: <<https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/>>. Acesso em: 30 nov. 2016. Ciclo de Vida

BOOTSTRAP. **Bootstrap Framework**. [S.l.], 2016. Disponível em: <<https://getbootstrap.com/>>. Acesso em: 25 dez. 2016.

CHARLAND, Andre; LEROUX, Brian. **Mobile application development: web vs. native**. Communications of the ACM, v. 54, n. 5, p. 49-53, 2011.

CIASC. **Dengue SC**. [S.l.], 2016. Disponível em: <<http://www.ciasc.sc.gov.br/302-aplicativo-dengue-sc-reforca-o-combate-ao-mosquito-aedes-aegypti-no-estado>>. Acesso em: 8 nov. 2016.

CORDOVA, Apache. **The Apache Software Foundation**. [S.l.], 2016. Disponível em: <<https://cordova.apache.org>>. Acesso em: 5 dez. 2016.

DEVELOPER. **The Google Developer**. [S.l.], 2016. Disponível em: <<https://console.developers.google.com/>>. Acesso em: 25 nov. 2016.

DEVELOPER. **Activity**. [S.l.], 2016. Disponível em: <<https://developer.android.com/reference/android/app/Activity.html>>. Acesso em: 28 nov. 2016.

DIRECTION. **The Google Directions API**. [S.l.], 2016. Disponível em: <<https://developers.google.com/maps/documentation/directions/>>. Acesso em: 22 nov. 2016.

FIGUEIREDO, Carlos MS; NAKAMURA, Eduardo. **Computação móvel: Novas oportunidades e novos desafios**. T&C Amazônia, v. 1, n. 2, 2003.

FIRTMAN, Maximiliano. **Programming the mobile web**. " O'Reilly Media, Inc.", 2010.

FRUITMAP. **Fruit Map**. [S.l.], 2016. Disponível em: <<https://play.google.com/store/apps/details?id=com.adarley1.fruitmap>>. Acesso em: 8 nov. 2016.

GEOLOCATION. **The Google Maps Geolocation API**. [S.l.], 2016. Disponível em: <<https://developers.google.com/maps/documentation/geolocation/>>. Acesso em: 22 nov. 2016.

GPS. **The Global Positioning System**. [S.l.], 2015. Disponível em: <<http://www.gps.gov/systems/gps/>>. Acesso em: 13 mai. 2015.

HALL, Martin; WALL, Richard. **Myiasis of humans and domestic animals**. Advances in parasitology, v. 35, p. 257-334, Elsevier, 1995.

IBGE, PNAD; IBGE. **Pesquisa nacional por amostra de domicílios**. 2014.

IDC, **Research. Mobile Internet Users to Top 2 Billion Worldwide in 2016**. [S.l.], 2016. Disponível em: <<https://www.idc.com/getdoc.jsp?containerId=prUS40855515>>. Acesso em: 8 nov 2016.

INTEL; Intel XDK: **Uma única solução para aplicativos de Internet das coisas e desenvolvimento de aplicativos móveis**. [S.l.], 2016. Disponível em: <<https://software.intel.com/intel-xdk/>>. Acesso em: 10 dez. 2016.

LECHETA, R. R. **Google Android: aprenda a criar aplicações para dispositivos móveis com Android SDK**. São Paulo: Novatec, 2010.

MAPS. **The Google Maps API**. [S.l.], 2016. Disponível em: <<https://developers.google.com/maps/>>. Acesso em: 22 nov. 2016.

NATIVESCRIPT, [S.l.], 2016. **Build amazing iOS and Android apps with technology you already know**. Disponível em: <<https://www.nativescript.org/>>. Acesso em: 10 dez. 2016.

PHREEZE. **Phreeze Framework**. [S.l.], 2016. Disponível em: <<http://www.phreeze.com/>>. Acesso em: 27 dez. 2016.

SCHILLER, Jochen; VOISARD, Agnès (Ed.). **Location-based services**. Elsevier, 2004.

SENCHA, I. **Sencha Touch**. [S.l.], 2016. Disponível em: <<https://www.sencha.com/>>. Acesso em: 04 nov. 2016.

TEIXEIRA, Pedro. Professional Node.js: **Building Javascript based scalable software**. John Wiley & Sons, 2012.

VAZ, Jhannes Alberto. **Comparação da cobertura e acurácia entre os sistemas GLONASS e GPS obtidas dos dados de observação de uma estação da rede brasileira de monitoramento contínuo**. Revista Brasileira de Cartografia, n. 65/3, 2013.

WHITE, James. **Going native (or not): Five questions to ask mobile application developers**. Australasian Medical Journal [AMJ 2013, 6, 1, 7-14]. Disponível em: <<http://dx.doi.org/10.4066/AMJ.2013.1576>>. Acesso em: 30 nov. 2016.

XAMARIN; [S.I.], 2016. Disponível em: <<https://www.xamarin.com/platform>>. Acesso em: 11 dez. 2016.

APÊNDICE A

Código JavaScript da definição do banco de dados interno do aplicativo móvel.

```

window.dati = {
  /* Variáveis do BD do sistema */
  DB_NAME: "rmiiase_bd",
  DB_VERSION: "1.0",
  DB_SIZE: 1*1024*1024,
  DB_TABLES: [
    {
      "table": "usuario",
      "fields": [
        {"name": "ID", "type": "INTEGER", "size": 11, "default": null, "key": true},
        {"name": "ID2", "type": "INTEGER", "size": 11, "default": null, "key": false}, // idservidor
        {"name": "NOME", "type": "VARCHAR", "size": 60, "default": "", "key": false},
        {"name": "EMAIL", "type": "VARCHAR", "size": 120, "default": "", "key": false},
        {"name": "FONE", "type": "VARCHAR", "size": 20, "default": "", "key": false},
        {"name": "ALTERADO", "type": "VARCHAR", "size": 2, "default": "", "key": false},
        {"name": "PAIS", "type": "VARCHAR", "size": 50, "default": "", "key": false},
        {"name": "ESTADO", "type": "VARCHAR", "size": 50, "default": "", "key": false},
        {"name": "CIDADE", "type": "VARCHAR", "size": 50, "default": "", "key": false},
        {"name": "LATITUDE", "type": "VARCHAR", "size": 20, "default": "", "key": false},
        {"name": "LONGITUDE", "type": "VARCHAR", "size": 20, "default": "", "key": false},
        {"name": "IDIOMA", "type": "VARCHAR", "size": 2, "default": "", "key": false},
      ]
    },
    {
      "table": "caso",
      "fields": [
        {"name": "ID", "type": "INTEGER", "size": 11, "default": null, "key": true},
        {"name": "ID2", "type": "INTEGER", "size": 11, "default": null, "key": false}, // idservidor
        {"name": "NOME", "type": "VARCHAR", "size": 60, "default": "", "key": false},
        {"name": "ENDERECO", "type": "VARCHAR", "size": 120, "default": "", "key": false},
        {"name": "SEXO", "type": "VARCHAR", "size": 20, "default": "", "key": false},
        {"name": "UNDSAUDE", "type": "VARCHAR", "size": 120, "default": "", "key": false},
        {"name": "LOCALLESAO", "type": "VARCHAR", "size": 120, "default": "", "key": false},
        {"name": "OBS", "type": "VARCHAR", "size": 200, "default": "", "key": false},
        {"name": "IDUSUARIOSEVIDOR", "type": "INTEGER", "size": 11, "default": null, "key": false},
        {"name": "FOTO", "type": "LONGBLOB", "default": null, "key": false},
        {"name": "LATITUDE", "type": "VARCHAR", "size": 20, "default": "", "key": false},
        {"name": "LONGITUDE", "type": "VARCHAR", "size": 20, "default": "", "key": false},
        {"name": "ALTERADO", "type": "INTEGER", "size": 2, "default": null, "key": false},
        {"name": "FOTOALTERADA", "type": "INTEGER", "size": 2, "default": null, "key": false},
        {"name": "HOMOLOGADO", "type": "INTEGER", "size": 2, "default": null, "key": false},
        {"name": "EXCLUÍDO", "type": "INTEGER", "size": 2, "default": null, "key": false},
        {"name": "IDLABSEVIDOR", "type": "INTEGER", "size": 11, "default": null, "key": false},
      ]
    },
    {
      "table": "laboratorio",
      "fields": [
        {"name": "ID", "type": "INTEGER", "size": 11, "default": null, "key": true},
        {"name": "ID2", "type": "INTEGER", "size": 11, "default": null, "key": false}, // idservidor
        {"name": "NOMELAB", "type": "VARCHAR", "size": 60, "default": "", "key": false},
        {"name": "ENDLOGLAB", "type": "VARCHAR", "size": 60, "default": "", "key": false}, // Logradouro
        {"name": "ENDNUMLAB", "type": "VARCHAR", "size": 30, "default": "", "key": false}, // Número
        {"name": "ENDCOMPLAB", "type": "VARCHAR", "size": 30, "default": "", "key": false}, // Complemento
        {"name": "CEPLAB", "type": "VARCHAR", "size": 9, "default": "", "key": false},
      ]
    }
  ]
}

```

```

    {"name":"EMAILLAB", "type":"VARCHAR", "size":120, "default":"","key":false},
    {"name":"SITELAB", "type":"VARCHAR", "size":120, "default":"","key":false},
    {"name":"FONELAB", "type":"VARCHAR", "size":20, "default":"","key":false},
    {"name":"FONEWALAB", "type":"VARCHAR", "size":20, "default":"","key":false},
    {"name":"PAISLAB", "type":"VARCHAR", "size":50, "default":"","key":false},
    {"name":"ESTADOLAB", "type":"VARCHAR", "size":50, "default":"","key":false},
    {"name":"CIDADELAB", "type":"VARCHAR", "size":50, "default":"","key":false},
    {"name":"LATITUDELAB", "type":"VARCHAR", "size":20, "default":"","key":false},
    {"name":"LONGITUDELAB", "type":"VARCHAR", "size":20, "default":"","key":false},
    {"name":"DATAALTERLAB", "type":"VARCHAR", "size":20, "default":"","key":false},
  ]
}
],

/** Funções de conexão ao banco de dados local */
initialize: function(){...}
connect: function(callback) {...}

/** Funções de estrutura do banco de dados local */
loadSchema: function(callback) {...}
resetDatabase: function(callback) {...}
emptyTable: function(table,callback) {...}
dropTable: function(table,callback) {...}

/** Funções de manipulação do banco de dados local */
query: function(sql, callback) {...}
selectAll: function(table, callback) {...}
insert: function(table, jsonRegister, callback){...}
update: function(table, jsonFields, key, value, callback){...}
delete: function(table, key, value, callback){...}
function log(msg) {...}

```

APÊNDICE B

Código PHP da API responsável por fornecer ao aplicativo os dados dos Casos e dos Laboratórios, que estão armazenados no banco de dados *web*.

```
<?php
# Definindo pacotes de retorno em padrão JSON.
header('Content-Type: application/json;charset=utf-8');

# Carregando o framework Slim.
require 'Slim/Slim.php';
\Slim\Slim::registerAutoloader();

# Iniciando o objeto de manipulação da API SlimFramework
$app = new \Slim\Slim();
$app->response()->header('Content-Type', 'application/json;charset=utf-8');

# Função de teste de funcionamento da API. Quando acessarem "meu.site.com/api/"
$app->get('/', function () { echo "Bem-vindo a API do Sistema"; });

# Função para obter dados da tabela 'caso'...
$app->get('/casos',function()// Quando acessarem "meu.site.com/api/casos"

    # Variável que irá ser o retorno (pacote JSON)...
    $retorno = array();

    # Abrir conexão com banco de dados.
    $conexao = new MySQLi("meu_host","usuario_banco","senha","banco");

    # Validar se houve conexão.
    if(!$conexao){ echo "Não foi possível se conectar ao banco de dados"; exit;}

    # Selecionar todos os cadastros da tabela 'caso'.
    $registros = $conexao->query("select * from caso INNER JOIN usuario ON
caso.IDUSUARIOSEVIDOR=usuario.IDUSUARIOSEVIDOR order by caso.IDUSUARIOSEVIDOR, CODIGO");

    # Transformando resultset em array, caso ache registros.
    if($registros->num_rows>0){
        while($caso = $registros->fetch_array(MYSQL_BOTH)) {
            $registro = array(
                "CODIGO"           => $caso["CODIGO"],
                "NOME"             => $caso["NOME"],
                "NOMEUSUARIO"      => $caso["NOMEUSUARIO"],
                "ENDERECO"         => $caso["ENDERECO"],
                "SEXO"             => $caso["SEXO"],
                "UNDSAUDE"         => $caso["UNDSAUDE"],
                "LOCALLESAO"       => $caso["LOCALLESAO"],
                "OBS"              => $caso["OBS"],
                "IDSEVIDOR"        => $caso["IDSEVIDOR"],
                "IDUSUARIOSEVIDOR" => $caso["IDUSUARIOSEVIDOR"],
                "LATITUDE"         => $caso["LATITUDE"],
                "LONGITUDE"        => $caso["LONGITUDE"],
                "ALTERADO"         => $caso["ALTERADO"],
            );
            $retorno[] = $registro;
        }
    }
}
```

```

# Encerrar conexão.
$conexao->close();

# Retornando o pacote (JSON).
$retorno = json_encode($retorno);
echo $retorno;
});

# Função para obter dados da tabela 'laboratorios'.
$app->get('/laboratorios',function(){ // Quando acessarem "meu.site.com/api/laboratorios"

# Variável que irá ser o retorno (pacote JSON).
$retorno = array();

# Abrir conexão com banco de dados.
$conexao = new MySQLi("meu host","usuario_banco","senha","banco");

# Validar se houve conexão.
if(!$conexao){ echo "Não foi possível se conectar ao banco de dados"; exit;}

# Selecionar todos os cadastros da tabela 'laboratorio'.
$registros = $conexao->query("select * from laboratorio order by DATAALTERLAB");

# Transformando resultset em array, lab ache registros...
if($registros->num_rows>0){
    while($lab = $registros->fetch_array(MYSQL_BOTH)) {
        $registro = array(
            "IDLABSERVIDOR"      => $lab["IDLABSERVIDOR"],
            "IDLABLOCAL"        => $lab["IDLABLOCAL"],
            "NOMELAB"           => $lab["NOMELAB"],
            "ENDLOGLAB"         => $lab["ENDLOGLAB"],
            "ENDNUMLAB"         => $lab["ENDNUMLAB"],
            "ENDCOMPLAB"       => $lab["ENDCOMPLAB"],
            "CEPLAB"            => $lab["CEPLAB"],
            "EMAILLAB"          => $lab["EMAILLAB"],
            "SITELAB"           => $lab["SITELAB"],
            "FONELAB"           => $lab["FONELAB"],
            "FONEWALAB"         => $lab["FONEWALAB"],
            "PAISLAB"           => $lab["PAISLAB"],
            "ESTADOLAB"         => $lab["ESTADOLAB"],
            "CIDADELAB"         => $lab["CIDADELAB"],
            "LATITUDELAB"       => $lab["LATITUDELAB"],
            "LONGITUDELAB"      => $lab["LONGITUDELAB"],
            "DATAALTERLAB"     => $lab["DATAALTERLAB"],
        );
        $retorno[] = $registro;
    }
}

# Encerrar conexão.
$conexao->close();

# Retornando o pacote (JSON).
$retorno = json_encode($retorno);
echo $retorno;
});

# Executar a API e deixá-la acessível.
$app->run();
?>

```


APÊNDICE C

Código JavaScript da função de *upload* de casos para o servidor (lado cliente).

```
function uploadCasosServidor(){
  var iduserserver; // o id usuário registrado no servidor
  /* Verifica se há conexão */
  if(verMinhaConexao()==1){
    dati.selectAll("usuario", function (registros) { // verifica se é o primeiro uso do app
      $.each(registros, function (c, usuario) {
        iduserserver = usuario.IDUSUARIOSEVIDOR;

        /* Se o usuário ainda não tiver cadastrado no servidor */
        if(usuario.IDUSUARIOSEVIDOR==null || usuario.IDUSUARIOSEVIDOR==0){
          uploadUsuarioServidor(); // faz o registro do usuário no servidor (que é dono dos casos)
        }else{
          dati.selectAll("caso", function (registross) {
            $.each(registross, function (c, caso) {

              // Envia só os que não tenham sido feito upload
              if (caso.IDSERVIDOR==null || caso.IDSERVIDOR=="0") {
                var dadoCasoLocal = {
                  "parametro": "uploadcaso",
                  "codigo": caso.CODIGO,
                  "nome": caso.NOME,
                  "endereco": caso.ENDERECO,
                };
                $.post('servidor.php', dadoCasoLocal, function(res) {
                  if(res>0){ // retorna o id do caso no servidor / Se for zero, é erro.
                    dati.update("caso", {"IDSERVIDOR": res}, "CODIGO", caso.CODIGO, function(bool){
                      });
                    console.log("Caso registrado no servidor com sucesso! Id: "+res);
                  }else{
                    console.log("ERRO no upload do caso no server: "+res);
                  }
                });
              }
            });
          }
        });
      });
    });
  }
}
```

APÊNDICE D

Código PHP (lado servidor) que recebe as requisições do aplicativo móvel.

```
<?php
header("Content-Type: text/html; charset=ISO-8859-1",true);
$host = "my_host"; $username = "my_user"; $password = "pass"; $db = "my_db";
$parametro= $_POST['parametro'];

switch ($parametro) {
    case "uploadcaso":
        $codigo= $_POST['codigo'];
        $nome= $_POST['nome'];
        $endereco= $_POST['endereco'];
        $sexo= $_POST['sexo'];
        $undsauade= $_POST['undsauade'];
        $locallesao= $_POST['locallesao'];
        $obs= $_POST['obs'];
        $idservidor= $_POST['idservidor'];
        $idusuarioservidor= $_POST['idusuarioservidor'];
        $idlabservidor= $_POST['idlabservidor'];
        $latitude= $_POST['latitude'];
        $longitude= $_POST['longitude'];
        $homologado= $_POST['homologado'];

        try {
            $conn = new PDO("mysql:host=$host;dbname=$db",$username,$password);
            $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

            $sql = "INSERT INTO caso (CODIGO, NOME, ENDERECO, SEXO, UNDSAUDE,
            LOCALLESAO, OBS, IDSERVIDOR, IDUSUARIOSERVIDOR, DLABSERVIDOR,LATITUDE,
            LONGITUDE, ALTERADO)
            VALUES ('$codigo','$nome','$endereco','$sexo','$undsauade',
            '$locallesao','$obs','$idservidor','$idusuarioservidor',
            '$idlabservidor','$latitude','$longitude','$alterado')";

            // use exec() because no results are returned
            $conn->exec($sql);
            $last_id = $conn->lastInsertId();
            echo $last_id; // retorna o lastInsertId
        }
        catch(PDOException $e)      echo $sql . "<br>" . $e->getMessage();

        $conn = null;
        break;

    // Outras funções não detalhadas
    case "atualizarLabsLocal": (...)
    case "downloaddadoscaso": (...)
    case "apagarcaso": (...)
    case "vercasohomologado": (...)
    case "updatefoto": (...)
    case "downloadfoto": (...)
    case "updatecaso": (...)
    case "uploadusuario": (...)
    case "updateusuario": (...)
    default:
        break;
}
?>
```