FUNDAÇÃO UNIVERSIDADE DO TOCANTINS
SISTEMAS DE INFORMAÇÃO

ANDRÉIA GUALBERTO PEREIRA

DESENVOLVIMENTO DE APLICATIVO DE PROPAGAÇÃO DE MARKETING POR MEIO DE WEB SERVICE, DISPOSITIVOS MÓVEIS E SERVIÇO DE GEORREFERENCIAMENTO

PALMAS / TO
2014

FUNDAÇÃO UNIVERSIDADE DO TOCANTINS

SISTEMAS DE INFORMAÇÃO

ANDRÉIA GUALBERTO PEREIRA

DESENVOLVIMENTO DE APLICATIVO DE PROPAGAÇÃO DE MARKETING POR MEIO DE WEB SERVICE, DISPOSITIVOS MÓVEIS E SERVIÇO DE GEORREFERENCIAMENTO.

Trabalho de Conclusão de Curso de Sistemas de Informação da Fundação Universidade do Tocantins, apresentado como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Msc. Alex Coelho

PALMAS / TO

2014

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca da Universidade do Tocantins
Campus I

P436d Pereira, Andréia Gualberto.
Desenvolvimento de aplicativo de propagação de marketing por meio de web service, dispositivos móveis e serviço de georreferenciamento / Andréia Gualberto Pereira. – Palmas, 2014.
74f.

Monografia (TCC) – Universidade do Tocantins, Curso de Sistemas de Informação, 2014.
Orientador (a): Prof. Alex Coelho

1. Android. 2. Georreferenciamento. 3. Web Service. I. Título.

CDD 003.3

Bibliotecário: Paulo Roberto Moreira de Almeida
CRB-2 / 1118

DEDICATÓRIA

Dedico este trabalho a minha família,
amigos e todas as pessoas que acreditaram em
meus sonhos, e sempre deram força para que
pudesse realizá-los.

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida e pela força diante dos obstáculos.

Agradeço a minha família: aos meus pais Maria Anita Gualberto e Celeno Castro, que apesar de estarem longe sempre demonstraram muito carinho, amor, paciência, orações e atenção, e acima de tudo acreditaram e me encorajaram a ir em busca dos meus sonhos. E aos meus irmãos Celé, Laura, Jard, Selma e João Celino, pelo amor, amizade, e palavras de incentivo às quais tiveram muita importância para a conclusão desse curso.

Sou muito grata aos meus amigos, em especial aos colegas, Leydinaldo, João Paulo, e Bruno pelo apoio e por tudo que passamos juntos no decorrer do curso.

Agradeço também, ao meu orientador Alex Coelho pela ajuda e confiança. E aos demais professores que de alguma forma contribuíram para o meu desenvolvimento profissional.

E por fim, e não menos importante agradeço a Fundação Universidade do Tocantins, por dá a oportunidade de concluirmos uma graduação gratuita e proporcionar um ensino de qualidade.

“Não faz sentido olhar para trás e pensar: devia ter feito isso ou aquilo, devia ter estado lá. Isso não importa. Vamos inventar o amanhã, e parar de nos preocupar com o passado.”

(Steve Jobs)

RESUMO

Dentro do contexto moderno, a computação móvel se tornou algo natural nas mais diferentes esferas sociais e econômicas. Com o intuito de explorar tal nicho de mercado, inúmeras são as aplicações que trabalham tecnologias que propiciam uma maior propagação e alcance de técnicas de marketing. Assim, verifica-se que a tecnologia passou a ditar e fazer com que os meios para tal propagação se tornem também diferenciados. Assim, o presente trabalho tem por objetivo apresentar a criação de aplicativo que permite que sejam realizadas buscas de propagandas e promoções cadastradas por empresas em um raio que pode ser definido de 1 a 50 quilômetros. Essas propagandas e promoções podem ser visualizadas e compartilhadas. Para o desenvolvimento deste, foram utilizadas diversas tecnologias, sendo as principais a plataforma Android, *Web Service RESTful*, e ferramentas de georreferenciamento.

Palavras-chaves: Android; Georreferenciamento; *Web Service.*

ABSTRACT

Inside the modern context, mobile computing has become something natural in many different social and economic spheres. In order to use that many niche market, applications are working technologies that promote a greater spread and range of marketing techniques. Thus, it is apparent that the technology has to dictate and have means for such propagation also become different. That way, this paper aims to present the creation of an application that allows searches in advertisements and promotions by companies registered in a distance that can be set from 1 to 50 km. These advertisements and promotions can be viewed and shared. To develop this, various technologies were used the main Android platform, *RESTful Web Service*, and georeferencing tools.

Keywords: Android; Georeferencing; *Web Service.*

Lista de Abreviaturas

ADT – *Android Development Tools*

DAO – *Data Access Object*

GPS – *Global Positioning System*

HTTP – *HyperText Transfer Protocol*

IP – *Internet Protocol*

JAX-RS – *Java API for RESTful Web Services*

JCP – *Java Community Process*

JDBC – *Java DataBase Connectivity*

JDK – *Java Development Kit*

JSON – *JavaScript Object Notation*

JSR – *Java Specification Request*

REST – *Representation State Transfer*

SDK – *Software Development Kit*

SMS – *Short Message Service*

SO – *Sistema Operacional*

SOAP – *Simple Object Application Protocol*

URI – *Unified Resource Identification*

USB – *Universal Serial Bus*

Wi-fi – *Wireless Fidelity*

XML – *eXtensible Markup Language*

Lista de Quadros

Lista de Figuras

SUMÁRIO

1. INTRODUÇÃO

No atual contexto social, pode-se perceber que o uso dos dispositivos móveis tornou-se indispensável na vida das pessoas, por estarem facilitando a execução das atividades cotidianas. De acordo com a tecnologia do aparelho é permitido que sejam instalados vários atrativos nos dispositivos, denominados como aplicativos. Esses por sua vez, são softwares desenvolvidos em determinada linguagem, que permitem o acesso a diversos conteúdos e utilizam as funções do aparelho de forma eficiente.

Conforme afirma Lemos (2007) “os dispositivos móveis são as ferramentas mais importante de convergência midiática hoje”. Essa afirmação decorre devido às pessoas estarem grande parte do tempo conectadas a internet, realizando diversos tipos de atividades. Dentre as atividades que podem ser realizadas por meio da rede, está o desenvolvimento de trabalhos e estudos, interação em redes sociais, o acesso a diversos tipos de informações, e outras inúmeras possibilidades.

Segundo recente estudo realizado pela União Internacional de Telecomunicações, agência do Sistemas das Nações Unidas, o número de celulares chegará até o final deste ano ao montante de sete bilhões de aparelhos, o que significa que o número de celulares se igualará ao número de pessoas que existem na terra (ITU, 2014) .

Levando em consideração a disposição das pessoas em realizar suas atividades por meio dos dispositivos móveis, abriu-se então um grande nicho de mercado voltado para várias áreas, que é preenchido com o desenvolvimento de aplicativos. Porém, vale salientar que características como: agilidade, modernidade, flexibilidade e o fato da plataforma ser de código aberto e livre ou não, são fatores que contribuem para o uso de plataformas como o Android da Google ou iOS da Apple.

Assim, este trabalho tem por objetivo geral o desenvolvimento de aplicativo de colaboração de marketing capaz de consumir serviços disponíveis em *Web Services,* de acordo com a localização geográfica. Para o total funcionamento dessa aplicação, tem-se os seguintes objetivos específicos: desenvolvimento de aplicação web para cadastro e manipulação de propagandas e promoções de empresas; implementação do *Web Service* para

oferta de serviços; e o desenvolvimento de um aplicativo para dispositivos móveis, que de acordo com a localização geográfica, consuma propagandas e promoções disponibilizadas no *Web Service.*

Vale ressaltar que esse projeto visa facilitar a busca do consumidor por produtos que estejam em promoção, evitando assim o desperdício de tempo, e uma deslocação desnecessária que poderia acontecer dada a necessidade de verificar se determinado produto está em promoção. Além disso, o mesmo também contribui de forma significativa para os comerciantes, que por sua vez, terão mais um veículo de comunicação acessível e barato para que seus clientes fiquem sabendo de suas novidades por meio do aplicativo.

Este trabalho está organizado da seguinte forma. No capítulo 2, é apresentado o arcabouço teórico, que se resume em um estudo dos principais conceitos a serem trabalhados durante o desenvolvimento desse trabalho, dentre eles a plataforma Android, o Georreferenciamento, e o *Web Services*. Dentro desses conceitos existem diversos outros, que são desmembrados durante o desenvolvimento do trabalho. O capítulo 3 descreve a metodologia do trabalho. No capítulo 4 é apresentada uma análise dos resultados obtidos com o desenvolvimento, que traz como ainda as dificuldades encontradas, e como foram sanadas essas dificuldades. Por fim, no capítulo 5 estão as considerações finais sobre o estudo deste trabalho e trabalhos futuros.

2. ARCABOUÇO TEÓRICO

Este capítulo apresenta os conceitos usados durante o desenvolvimento deste trabalho. São apresentados conceitos sobre a plataforma Android, georreferenciamento, *Web Services*, além disso, algumas linguagens de programação e especificações utilizadas.

2.1 ANDROID

Para a criação de aplicação móvel, é preciso fazer uso de alguma plataforma de desenvolvimento. Diversos sistemas operacionais (SO's) tem dado essa possibilidade aos desenvolvedores, dentre eles pode-se citar as principais plataformas do contexto atual, sendo elas, o iOS, Windwos Phone, BlackBerry e o Android.

Outro fator importante para a criação de uma aplicação é a linguagem de programação que será utilizada. Vale ressaltar que a linguagem muda de acordo com a plataforma escolhida, e cabe ao desenvolvedor decidir qual sistema ou linguagem de programação melhor te satisfaz.

O Android consiste no principal player no mercado de SO's para dispositivos móveis. É uma plataforma de desenvolvimento baseada em conceitos do Sistema Operacional Linux, o que permite que suas aplicações sejam livres e de código aberto (*open source*). Esse fator é de grande importância e muito atrativo, pois permite a evolução rápida de seus componentes, devido ao fato de que programadores em qualquer lugar do mundo possam contribuir para melhorar a plataforma (LECHETA, 2013).

Diante de tais considerações, fato relevante a ser explorado consiste na arquitetura da plataforma Android, que será apresentada a seguir.

2.1.1 Arquitetura do Android

Android é uma pilha de software para dispositivos móveis que inclui sistema operacional, *middleware* e aplicações-chave. Esta pilha possui 4 níveis, conforme é apresentado na Figura 1.

Figura 1: Arquitetura da Plataforma Android

Fonte: ANDROID DEVELOPERS, 2014.

Na base da arquitetura Android, há o *Kernel (Linux Kernel),* formado por serviços essenciais dos sistemas, tais como *drives*, memórias, processos, configuração de redes, segurança e demais serviços que pode ser visualizado na Figura1.

Logo acima da camada de serviços, existe o *Runtime Android* e *Libraries*, no qual o primeiro é um conjunto de bibliotecas que fornece funcionalidades disponíveis na linguagem de programação Java, e a Máquina Virtual Dalvik, criada especialmente para a execução em dispositivos móveis. A segunda consiste em um conjunto de bibliotecas C/C++ utilizada por vários componentes da plataforma, dentre elas tem-se bibliotecas que permitem a manipulação de banco de dados (*SQLite*), a renderização de imagens e fontes (*FreeType*), a manipulação de objetos 2D e 3D (*Surface Manager*), dentre outras bibliotecas que também possuem funcionalidades importantíssimas (TOSIN, 2011).

Na próxima camada descrita na Figura 1, encontra-se o *framework* de aplicação (*Framework Application*), nível no qual estão localizados programas que gerenciam funções do próprio aparelho, e é o local que a aplicação irá executar. Dentre as funções pode-se citar as aplicações do telefone, mudanças

de processos, localização do aparelho por meio do GPS (*Global Positioning System* – Sistema de Posicionamento Global).

Por fim, no topo da arquitetura são encontradas as aplicações (*Applications*), local onde estão as funções básicas do telefone, também conhecida como interface de usuário comum, pois nesse local está tudo aquilo que esse pode manipular (TOSIN, 2011).

2.1.2 Componentes de uma aplicação Android

Existem diversos componentes de uma aplicação Android essenciais para que o sistema possa ser instanciado e executado sempre que necessário. Esses componentes por sua vez não trabalham de forma isolada, a Figura 2 apresenta os principais elementos de uma aplicação.

Figura 2: Componentes de uma aplicação Android.

FONTE: TOSIN, 2011

A Figura 2 demonstra que para o funcionamento de uma aplicação Android é utilizado diversos componentes, e esses formam o Android Core, que segundo Tosin (2011), nada mais é que a própria plataforma Android.

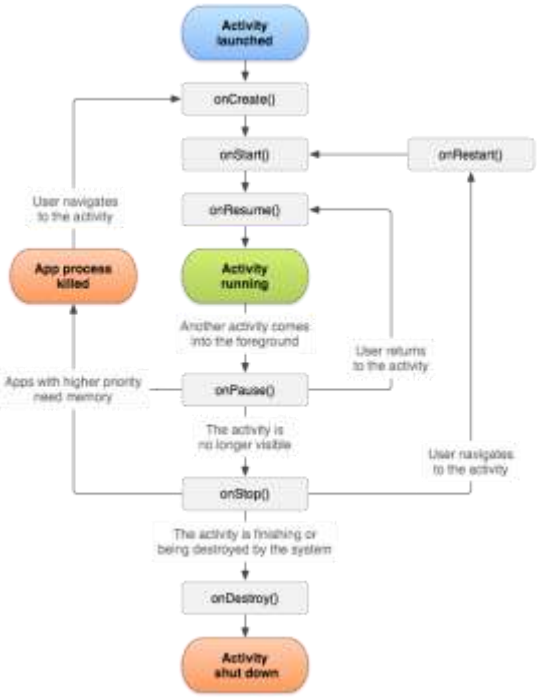
É necessário implementar esses componentes, pois as *Activities* são responsáveis pelo funcionamento das telas, os *Broadcast Receivers* "escutam" os eventos que ocorre durante a execução da aplicação e o *AndroidManifest* consiste no local no qual estão localizadas as configurações da aplicação. Dessa forma, estes elementos se tornam indispensáveis para o desenvolvimento de qualquer aplicação dentro desta plataforma.

A seguir serão apresentadas as principais características inerentes a esse processo e como cada uma funciona.

2.1.2.1 Activities

Estes importantes elementos da plataforma Android funcionam como mediadores que definem como as informações serão apresentadas ao usuário, além de controlar o fluxo da aplicação. Elas podem interagir com o usuário e trocar informações com outras *activities* ou *services* (MEIER, 2009 citado por GOMES, F.J.L. et al, 2011, p.103). Na Figura 3 pode-se visualizar o diagrama com os métodos mais importante de uma *activity*.

Figura 3: Ciclo de vida uma *activity.*



Fonte: ANDROID DEVELOPERS, 2014.

Os métodos do ciclo de vida de uma *activity*, apresentados na Figura 3 são descritos da seguinte forma:

a) onCreate (): é obrigatório e é chamado uma única vez. Nesse método deve-se criar uma *view* e chamar o método *setContentView(view)* para exibi-la na tela.
b) onStart(): é chamado quando a *activity* está ficando visível ao usuário e já tem uma *view*. Pode ser chamado depois dos métodos todos onCreate() ou onRestart(), dependendo do estado da aplicação.

c) onRestart(): é chamado quando uma *activity* foi parada temporariamente e está sendo iniciada outra vez. O método onRestart() chama o método onStart() de forma automática.
d) onResume(): é chamado quando a *activity* está no topo da pilha "*activity stack*", e dessa forma já está executando como a *activity* principal pronta para interagir com o usuário.
e) onPause() : se um evento ocorrer, como o celular entrar em modo espera para economizar energia, a *activity* do topo pilha que está executando pode ser temporariamente interrompida. Para isso o onPause() é chamado para salvar o estado da aplicação, para que posteriormente, quando a *activity* voltar a executar, tudo possa ser recuperado, se necessário, no método onResume().
f) método onStop() : é chamado quando a *activity* está sendo encerrada, e não está mais visível ao usuário, o que pode ocorrer quando outra *activity* é iniciada.
g) método onDestroy() : literalmente encerra a execução de uma *activity*. O método pode ser chamado automaticamente pelo sistema operacional para liberar recursos ou pode ser chamado pela aplicação pelo método finish() da classe *Activity*. Depois do método onDestroy() ter executado a *activity* é removida completamente da pilha e o seu processo no sistema operacional também é completamente encerrado. (LECHETA, 2013, p.119).

Para exemplificar, o Quadro 1 apresenta o ciclo de vida de uma *Activity.*

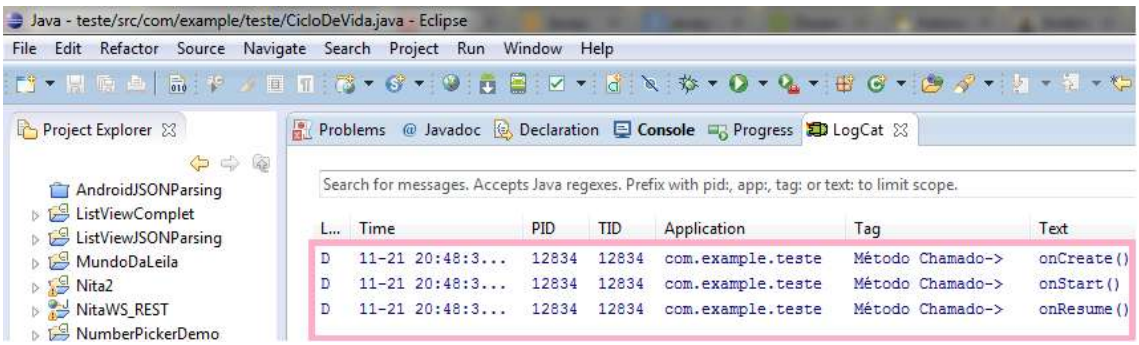
Quadro 1: Ciclo de Vida de uma Activity

```
public class CicloDeVida extends Activity {
	String metodo = "Método Chamado-> ";
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
		Log.d(metodo, "onCreate()");
	}
	@Override
	protected void onStart() {
		super.onStart();
		Log.d(metodo, "onStart()");
	}
	@Override
	protected void onResume() {
		super.onResume();
		Log.d(metodo, "onResume()");
	}
	@Override
	protected void onPause() {
		super.onPause();
		Log.d(metodo, "onPause()");
	}
	@Override
	protected void onStop() {
		super.onStop();
		Log.d(metodo, "onStop()");
	}
	@Override
	public void onDestroy() {
		super.onDestroy();
		Log.d(metodo, "onDestroy()");
	}
}
```

Fonte: Elaborado pela autora.

Ao executar o ciclo de vida exposto no Quadro 1, será apresentado na janela LogCat quais métodos que foram chamados, sendo eles o onCreate(), onStart() e onResume() como pode-se observar na Figura 4.

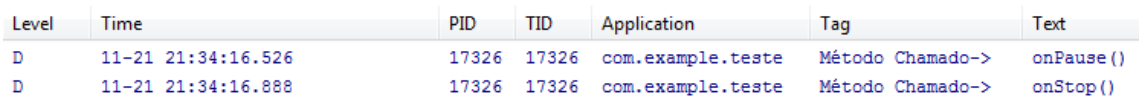
Figura 4: Resultado da execução no LogCat



Para um melhor entendimento do funcionamento dos métodos citados, uma situação clara do processo seria a seguinte. Se pressionado o ícone de

finalizar ligação do emulador do Android, os métodos onPause() e onStop() serão executados respectivamente, como apresentado na Figura 5.

Figura 5: Ícone finalizar ligação pressionado



Level	Time	PID	TID	Application	Tag	Text
D	11-21 21:34:16.526	17326	17326	com.example.teste	Método Chamado->	onPause()
D	11-21 21:34:16.888	17326	17326	com.example.teste	Método Chamado->	onStop()

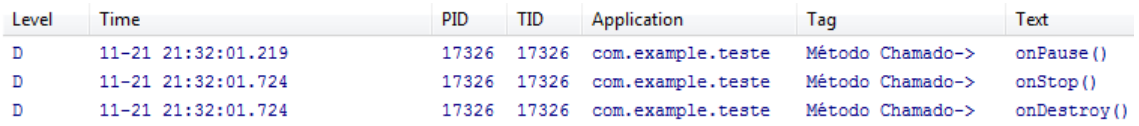
Se pressionado o ícone menu do emulador do Android os métodos onStart() e onResume() serão executados respectivamente, como pode-se observar na Figura 6.

Figura 6: Ícone menu pressionado

Level	Time	PID	TID	Application	Tag	Text
D	11-21 21:56:32.523	20501	20501	com.example.teste	Método Chamado->	onStart()
D	11-21 21:56:32.523	20501	20501	com.example.teste	Método Chamado->	onResume()

Se pressionado o ícone voltar do emulador do Android os métodos onPause(), onStop() e onDestroy() serão executados respectivamente, como apresentado na Figura 7.

Figura 7: Ícone voltar pressionado



Level	Time	PID	TID	Application	Tag	Text
D	11-21 21:32:01.219	17326	17326	com.example.teste	Método Chamado->	onPause()
D	11-21 21:32:01.724	17326	17326	com.example.teste	Método Chamado->	onStop()
D	11-21 21:32:01.724	17326	17326	com.example.teste	Método Chamado->	onDestroy()

A manipulação das *Activities* é um processo muito simples, e importante para o desenvolvimento de aplicações Android, pois o uso dos métodos definidos anteriormente permite a criação de uma aplicação mais robusta.

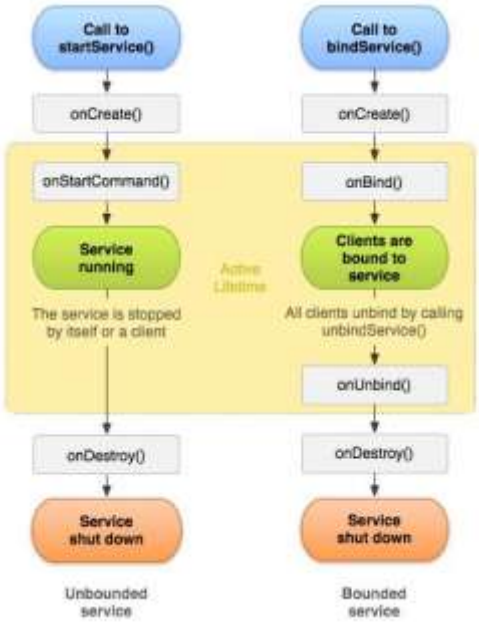
2.1.2.2 Services

São códigos executados em segundo plano e geralmente são utilizados para tarefas que requerem um grande tempo de execução. Estes códigos não possuem interfaces de usuário, rodam em background e não são interrompidos quando é feita a troca de atividade pelo usuário. Os *services* permanecem ativos até o momento que recebe uma nova ordem. Depois de estabelecido

conexão com o serviço, é possível efetuar comunicação através de uma interface apresentada pelo usuário (PEREIRA; SILVA, 2009).

Para melhor entendimento, a Figura 8 apresenta um ciclo de vida de um *service*.

Figura 8: Ciclo de vida de um *service*



Fonte: TUTORIALSPOINT, 2013

Segundo Lecheta (2013), existem duas maneiras de iniciar um serviço em segundo plano, por meio dos métodos startService(),e bindService(). O primeiro é o método utilizado para iniciar um serviço que fica executando por tempo indeterminado, até quando o método stopService() seja acionado ou até que o próprio serviço termine a sua própria execução chamando e método stopSelf(). Já o método bindService() pode iniciar um serviço se ele ainda não está executando ou simplesmente conectar-se a ele. Ao estabelecer a conexão é possível recuperar uma referência de uma classe ou interface que pode manipular o serviço. Para exemplificar, o Quadro 2 apresenta um exemplo de *service*.

Quadro 2: *Service* Android

```
public class MainActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
    public void startService(View view) {
        startService(new Intent(getBaseContext(), MyService.class));
    }
    public void stopService(View view) {
        stopService(new Intent(getBaseContext(), MyService.class));
    }
}
```

Fonte: TUTORIALSPOINT, 2013

A implementação de *services* não possui nada de complexo. Conforme demonstrado no Quadro 2, exemplificando os método para iniciar e finalizar um serviço, sendo eles startService() e stopService(). Para realizar esse evento o exemplo faz uso de um componente muito simples e importante do Android, o *intent*.

2.1.2.3 Intents e BroadcastReceivers

São componentes que ficam aguardando a ocorrência de um determinado evento, pode-se entender como evento a inicialização do sistema operacional, uma chamada de voz, a chegada de um SMS, um evento disparado por uma aplicação (MEIER, 2009 citado por GOMES, F.J.L. et al, 2011, p.104).

Intents são elementos chave no Android, porque facilitam a criação de novas aplicações a partir de aplicações já existentes. Assim, a utilização de *Intents* demonstra uma característica única e diferenciada a ser explorada em projetos como o proposto neste trabalho, ao fato que possibilita que exista uma interação com outras aplicações e serviços que podem proporcionar informações necessárias ao sucesso da aplicação. Para exemplificar, o Quadro 3 apresenta uma *intent*.

Quadro 3: Exemplo de *Intent*

```
Intent it = new Intent (this, Tela2.class);
startActivity(it);
```

Fonte: LECHETA, 2013, p.154

No Quadro 3 é apresentado um exemplo de *intent*. Se o código for executado será criada e instanciada a *intent* it, e posteriormente essa *intent* é

chamada por meio do método startActivity(), que irá iniciar a atividade definida na *intent* it, ou seja abrirá a classe Tela2.

Conforme descreve Lecheta (2013, p.313) a classe *BroadcastReceivers*, "é uma das classes mais importantes da arquitetura Android, utilizada para que aplicações possam reagir a determinados eventos gerados por uma *intent*". Diversos eventos podem ser tomados como exemplo, a chegada de uma mensagem de texto, uma chamada telefônica.

Existem duas maneiras de configurar um *BroadcastReceivers.* Pode-se registrar dinamicamente uma instância dessa classe com Context.registerReciever() ou estaticamente publicar uma aplicação através da *tag* <receptor> no *AndroidManifest.xml* (ANDROID DEVELOPER, 2014).

De acordo com Lecheta (2013, p.315), "independente de como o *BroadcastReceivers* for configurado, é necessário especificar um *IntentFilter* (filtro) para configurar a ação e categoria". Assim, no Quadro 4, é apresentado exemplo de como configurar um *BroadcastReceivers* de forma estática.

Quadro 4: Configuração Estática *BroadcastReceivers*

```
<receiver  android:name = "ExemploReceiver1" >
     <intent-filter>
        <action  android:name = "FAZER ALGO" />
     <category android:name="android.intent.category.DEFAULT"/>
     </intent-filter>
  </receiver>
```

Fonte: LECHETA, 2013, p.314

Logo na primeira linha de código do Quadro 4, está explicito qual classe a *tag* <receiver> está referenciando. Dessa forma, sempre que alguma *intent* chamar a ação "FAZER ALGO", a classe ExemploReceiver1, presente no Quadro 5, será executada.

Quadro 5: Exemplo de BrodcastReceiver

```
public class ExemploReceiver1 extends BroadcastReceiver{
       @Override
       public void on Receive(Context c, Intent intent){
              Toast.makeText(c,"ExemploReceiver1 1", Toast.LENGTH_SHORT.show();
       }
}
```

Fonte: LECHETA, 2013, p.314

Apesar de a classe *BroadcastReceivers* ser fundamental para o desenvolvimento de aplicativos móveis, vale ressaltar que a mesma não fica visível aos usuários, devido ao fato desta ser executada em segundo plano.

2.1.2.4 Content Providers

Consiste nos compartilhadores de conteúdo entre as aplicações, sendo que uma aplicação pode requisitar informações de outra. Por exemplo, uma aplicação pode receber dados da lista de contatos que é nativa do Android, e com base nesses dados, realizar algum processamento (LECHETA, 2010).

O Android tem uma série de provedores de conteúdo nativos que permitem consultar os contatos da agenda, visualizar os arquivos, imagens e vídeos (LECHETA, 2013, p.469). Para exemplificar, o Quadro 6 apresenta um exemplo de provedor de conteúdo.

Quadro 6: ContentProvider

```
public class ImprimeContatos extends Activity {
	private static final String CATEGORIA = "livro";
	@Override
	public void onCreate(Bundle icicle) {
		super.onCreate(icicle);

		TextView textView = new TextView(this);
		textView.setText("Verifique o LogCat");
		setContentView(textView);

		//	Recupera	a	Uri,	mesmo	que
"content://com.android.contacts/contacts".
		Uri contatos = ContactsContract.Contacts.CONTENT_URI;
		Log.i(CATEGORIA,"Uri: " + contatos);

		// Recupera o Cursor para percorrer a lista de contatos.
		Cursor cursor = getContentResolver().query(contatos, null, null,
null, null);
		imprimeContatos(cursor);
	}
	private void imprimeContatos(Cursor cursor) {
		Log.i(CATEGORIA,"Logando os nomes dos contatos cadastrados...");
		int count = cursor.getCount();
		Log.i(CATEGORIA,"Foram encontrados "+count+" contatos.");
		while(cursor.moveToNext()){
			String	nome	=
cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)
);
			Log.i(CATEGORIA,"Nome: " + nome);
		}
		// É importante fechar o cursor no final
		cursor.close();
	}
}
```

Fonte: LECHETA, 2013, p.472

No exemplo da classe ImprimeContatos, exposta no Quadro 6, a aplicação utiliza a URI *content://com.android.contacts/contacts/,* que retorna os contatos cadastrados na agenda do celular. Dessa forma, executando essa aplicação os nomes e telefones dos contatos são listados no LogCat.

Nas aplicações desenvolvidas na plataforma Android existe um arquivo de manifesto chamado "*AndroidManifest.xml*". Cada aplicação deve ter um arquivo deste em seu diretório raiz, visto que é obrigatório e único para cada aplicação. As configurações gerais da aplicação, assim como os componentes que fazem parte da mesma são feitas no manifesto Android (TOSIN, 2011).

Logo abaixo, no Quadro 7, é apresentado um exemplo de arquivo manifesto com algumas configurações.

Quadro 7: Exemplo *AndroidManifest*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.livro.android.cap4"
    android:versionCode="1"
    android:versionName="1.0.0" >

    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.INTERNET" />

    <uses-sdk
        android:minSdkVersion="5"
        android:targetSdkVersion="17" />
    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:theme="@style/AppTheme" >

        <activity
            android:name="Menu"
            android:label="Cap04-Activity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name=".activity.ExemploCicloVida" />
```

```
        <activity android:name=".activity.Tela2" />
    </application>
</manifest>
```

Fonte: Adaptado do LECHETA, 2013, p.138

No *AndroidManisfest.xml*, como pode se observar no Quadro 7 é apresentado de forma detalhada as configurações necessária para que o programa execute. Existem diversas informações contidas nele, tais como, qual pacote pertence, as permissões da aplicação, e a declaração das *activities* do projeto. No exemplo é apresentado, por exemplo, duas *activities*, sendo elas ExemploCicloVida e Tela2. Vale ressaltar que o número de *activities* não possui restrições, todas que forem utilizadas no projeto, devem ser declaradas.

Um componente importantíssimo apresentado no *AndroidManifest* são as permissões. Elas permitem que aplicação acesse outras funcionalidades do dispositivo móvel. No exemplo apresentado no Quadro 7, estão presentes as permissões *READ_CONTACTS, CALL_PHONE, INTERNET.* Essas representam respectivamente que o aplicativo terá permissão para ler os contatos da agenda telefônica do usuário, fazer ligação sem antes perguntar ao usuário se deseja realizar uma chamada telefônica, e permite que a aplicação tenha acesso a rede de internet.

Além dos componentes Android, existem também outras funcionalidades do Android que apesar de não serem obrigatórias para implementação de uma aplicação, também possuem importância no contexto móvel. As classes AsyncTask e LocationManager estão dentro desse grupo de funcionalidades que a plataforma Android permite utilizar, e foram indispensáveis para o desenvolvimento desse trabalho. Os próximos tópicos vêm apresentar esses dois conceitos.

2.1.2 AsyncTask

Existem situações em que há a necessidade da aplicação realizar uma requisição, e essa por sua vez, pode demandar algum tempo para responder devido a necessidade de criação da Classe AsyncTask. Segundo (LECHETA, 2013, p.408):

> Ela foi criada para auxiliar a criar uma *thread* e, ainda, sincronizar o acesso às *views* de um modo padronizado. Essa classe

> automaticamente vai executar o processo em uma *thread* separado e utilizar uma *handler* internamente, encapsulando todo o trabalho pesado.

Para se trabalhar com essa classe, é preciso criar uma classe abstrata que estenda da classe AsyncTask, e, além disso, é importante conhecer os seguintes métodos (LECHETA, 2013, p.409):

> a) onPreExecute() : Método executado antes de uma *thread* iniciar, sendo uma boa oportunidade para exibir uma janela de progresso ao usuário ou uma mensagem de “por favor, aguarde”.
> b) onInBackground(): Método executado sobre uma *thread* separada, devendo conter todo processamento pesado. Ele pode retornar um objeto qualquer, o qual será passado como parâmetro para o método onPostExecute(). É aqui que a thread executa, mas isso é feito automaticamente para você.
> c) onPostExecute(): Método executado na thread principal, a famosa thread de interface UI Thread, podendo atualizar os componentes da tela. Ele é chamado internamente, utilizando um Handler, encapsulando esse trabalho.

Os métodos citados anteriormente não precisam ser chamados na implementação do código, pois os mesmos já possuem uma sequência, e assim que é finalizado já se sabe qual o método que receberá o seu retorno para dá continuidade na execução do processo.

Resumindo, a AsyncTask é uma classe que tem por objetivo tirar todo o processamento lento da thread principal, e por esse motivo, é indicada para trabalhar com Web Services.

2.1.3 LocationManager

A plataforma Android disponibiliza está classe para ser implementada em aplicações que necessitam trabalhar com coordenadas geográficas. Segundo Android Developers (2014), o principal objetivo dessa classe é fornecer acesso aos serviços de localização do sistema. Esses serviços, por sua vez permitem que a aplicação receba atualizações periódicas da localização geográfica do sistema, ou aciona outra aplicação quando o dispositivo entra em determinada localização geográfica.

Para se trabalhar com a classe na plataforma Android é preciso adicionar a permissão *ACCESS_COARSE_LOCATION* que permite a localização geográfica derivadas de fonte de localização da rede, como torres de celulares e *Wireless Fidelity* (Wi-fi), ou a permissão

ACESS_FINE_LOCATION que possui como fonte de localização o GPS, e também pode captar por torres de celulares e Wi-fi.

Ainda, explorando o assunto sobre localização geográfica, o tópico seguinte aborda o georreferenciamento, possibilitando uma melhor compreensão funcionamento do GPS.

2.2 GEORREFERENCIAMENTO

Georreferenciar pode ser definido como o ato de obter as coordenadas de pontos de imagens ou mapas, tais como latitude e longitude conhecidas. Para a aquisição destas coordenadas, a metodologia mais aplicada consiste no uso de uma importante tecnologia que é obtida com facilidade na atualidade que é o GPS.

O Sistema de Posicionamento Global foi criado pelo departamento de defesa dos Estados Unidos da América na década de 1970, com finalidade de uso militar, sendo liberado para utilização civil a partir do ano de 2005. O sistema é constituído por uma constelação de 28 satélites sendo 4 sobressalentes em 6 planos orbitais que ficam a 20.200 km de altitude, passando sobre o mesmo ponto da terra duas vezes por dia (HUERTA (2005), citado por CANALLE, 2011, p.11). A Figura 9 demonstra o processo desempenhado para obtenção dos pontos.

Figura 9: Sistema de Triangulação.

Fonte: OFICINA DA NET (2011)

O sistema funciona da seguinte forma (HUERTA (2005), citado por CANALLE, 2011, p.11):

a) os satélites emitem sinais de rádio que são captados pelos aparelhos receptores GPS, os quais utilizam sinais de no mínimo 4 satélites e calculam a distância (chamada pseudodistância) entre a recepção e cada um dos satélites pelo intervalo de tempo entre o instante local e o instante em que os sinais foram enviados.

b) desta forma é possível efetuar a triangulação das posições e identificar a coordenada geográfica em que o aparelho receptor se encontra.

As características descritas sobre aplicações Android, além das potencialidades do sistemas de georreferenciamento podem ser explorados por meio da utilização dos Web Services. Assim, a seguir são apresentados os conceitos de fundamental importância para a implementação do trabalho, pois os serviços web permitem a disponibilização de dados na rede de forma que a aplicação Android possa consumir seus serviços.

2.3 WEB SERVICES

Web Services são tecnologias de integração de sistemas, empregada principalmente em ambientes heterogêneos, o que significa que pode desenvolver softwares ou componentes de software capazes de interagir, seja, enviando ou recebendo informações, com outros softwares, não importando a linguagem de programação em que estes foram desenvolvidos, o sistema operacional em que estes rodam e o hardware que é utilizado (GOMES, 2010).

Para exemplificar essa situação, Pamplona (2005), cita o seguinte:

> A comunicação entre os serviços é padronizada possibilitando a independência de plataforma e de linguagem de programação. Por exemplo, um sistema de reserva de passagens áreas feito em Java e rodando em um servidor Linux pode acessar, como transparência, um serviço de reserva de hotel feito em Net rodando em um servidor Microsoft.

Para trocar informações entre clientes e servidor, é necessário formatar tais dados. Um exemplo disso é a utilização de XML ou JSON que se tornaram

padrões em Web Services, facilitando assim a integração entre sistemas heterogêneos, no qual ambos conhecem a representação dos dados trocados entre si (CHAPPELL & JEWELL, 2002, citado por CAMPOS, J. 2013).

Dois são os principais padrões de desenvolvimento de Web Services, sendo o SOAP (*Simple Object Application Protocol*) e o padrão REST (*Representational State Transfer).* Segundo Rubbo (2013), os Web Services do padrão REST são considerados muito mais leves, por possuir conteúdo normalmente menor que os tradicionais envelopes SOAP. Esse fator é muito importante para o desenvolvimento de aplicações para dispositivos móveis, já que os dispositivos são menos favorecidos quando se refere a acesso à internet, processamento e memória.

Segundo Hamad, Saad e Abed (2010), SOAP mostrou-se muito verboso, devido ao fato, principalmente, de sua base estar em padronização XML(*eXtensible Markup Language*), enquanto que o paradigma *RESTful*, empregando a serialização de objetos utilizando JSON (*Javascript Object Notation*) que mostrou-se mais eficiente nestes dispositivos.

2.3.1 Arquitetura REST

Para que se possa ter um melhor entendimento do *RESTful* é de fundamental importância entender antes sua arquitetura, o REST. REST é um estilo arquitetural para sistemas multimídia distribuídos que enfatiza a generalização das interfaces, a escalabilidade da integração entre os componentes e a instalação independente dos mesmos. Este estilo foi descrito por Roy Fielding em sua tese de doutorado em 2000 (Fielding R., 2000 citado por MATTEUSSI, 2010).

O REST utiliza os recursos existentes no protocolo HTTP (*Hyper Text Transfer Protocol*) como meio para prover os serviços, e aplica uma série de restrições que farão com que os princípios da Web sejam respeitados (NGOLO, 2009, citado por SILVA, A. e COELHO, J.H. de S., 2010).

Os princípios de arquitetura incluem (HANSEN, 2007):

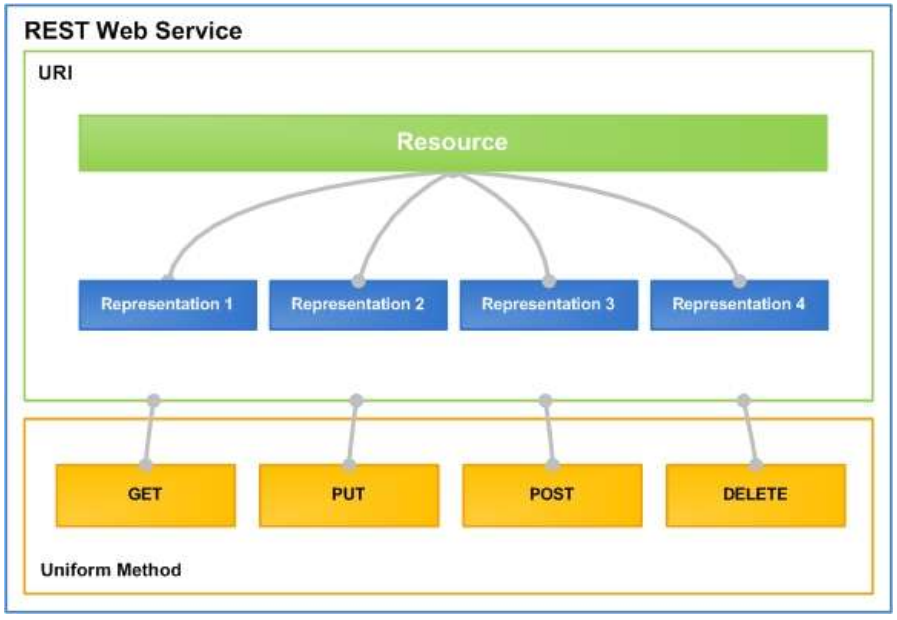
- Serviços *RESTful* são auto-contidos, ou seja, cada requisição de um cliente ao servidor deve conter toda a informação necessária para entender a requisição, e não

pode se beneficiar de qualquer contexto armazenado no servidor.

- Serviços *RESTful* possuem interface uniforme. Isso significa os recursos possuem a mesma interface que o cliente e que as únicas operações permitidas são operações HTTP: GET, POST, PUT e DELETE.
- Arquiteturas baseadas em REST são construídas com recursos unicamente identificados pelos URIs (*Unified Resource Identification*). Por exemplo, em um sistema *RESTful* de compras, cada compra possui uma URI única.
- Componentes REST manipulam recursos através de troca de representações de recursos. Por exemplo, um recurso de compra pode ser representado por um documento XML. Em um sistema de compras *RESTful*, uma compra poderia ser atualizada através do envio de um documento XML contendo a alteração da compra para sua URI.

Para exemplificar, o funcionamento dessa arquitetura, a Figura 10 apresentar estrutura de Web Services REST.

Figura 10: Estrutura *web service* REST.



Fonte: APACHE, 2009.

Analisando a Figura 10 apresentada anteriormente, tem-se (APACHE 2009):

a) URI: serviços web são organizados em recursos, e é por meio dos URIs que são identificados esses recursos, pois a mesmo contém seu nome e localização. Para acessar, inserir, ou manipular um recurso é chamada uma operação HTTP em um dos URIs do recurso.
b) *Uniform Method:* a arquitetura REST utiliza o protocolo HTTP como seu protocolo padrão, ou seja, possui interface uniforme. Por meio do HTTP, o REST tem acesso as suas operações que podem ser aplicadas a todos os seus recursos. As operações básicas são: GET que permite realizar a busca de recursos, PUT que permite criar ou atualizar o conteúdo do recurso, POST que também permite criar um novo recurso, e o DELETE que permite apagar o conteúdo do registro.

Sendo o REST uma arquitetura para o Web Services, foi necessário a criação de uma tecnologia que implantasse essa arquitetura. Dessa forma surgiu o *RESTful*, conceito esse que será apresentado no próximo tópico.

2.3.1 RESTful

O termo *RESTful*, também foi definido por Roy Fielding, referindo-se aos sistemas que seguem os princípios REST, também apresentam alguns princípios como (Fielding R., 2000 citado por MATTEUSSI, 2010):

- Suporte a escalabilidade.
- Cliente servidor.
- Apoio a sistemas em *cache*.
- Requisições para o servidor devem conter todas as informações para a requisição ser atendida.

Na arquitetura REST, as solicitações e respostas ao servidor podem ser estruturadas de diversas maneiras, o XML, JSON, HTML, PLAIN, dentre

outros. Para efeito de aplicação e desenvolvimento nesse trabalho utilizou-se JSON, por ser formato de fácil uso, e baixo custo de transferência, sendo apresentado.

2.3.2 JSON

JSON (*JavaScript Object Notation*) é uma das forma de recuperar os resultados da operações realizadas pelo *RESTful*. O JSON é um formato de troca de dados independente de linguagem, e leve, o que o torna de fácil entendimento tanto humanos quanto para máquinas analisarem e gerarem.

Vale lembrar que o JSON pode ser apresentado em diversas formas, tais como, objetos, membros, pares, e *arrays*. E um dos principais motivos da escolha do JSON está no seu tamanho reduzido em relação ao XML, pois acaba tendo uso mais propício em cenários em que a largura de banda, ou seja, a quantidade de dados que pode ser transmitida em determinado intervalo de tempo, é um recurso crítico (SAUDATE, 2013). Para exemplificar, o Quadro 8 apresenta um exemplo de objeto JSON.

Quadro 8: Exemplo JSON

```
{"Acadêmicos": [
        { "primeiro nome": "Andréia", "sobrenome": "Gualberto"},
        { " primeiro nome": "Bruno ", " sobrenome ": "Moraes" },
        { " primeiro  nome": "João Paulo ", " sobrenome ": "Paiva"}
        { " primeiro nome": "Leydinaldo", " sobrenome ": "Miranda" }
    ]
```

Fonte: Adaptado de W3SCHOOLS.

No Quadro 8, pode-se perceber que o JSON é fácil de compreender, no exemplo a estrutura é baseada em quatro objetos, e esses objetos são formados por um conjunto de pares/valores. Cada objeto possui dois conjuntos de pares/valores que são os atributos primeiro nome e sobrenome com seus respectivos valores, por exemplo, no primeiro objeto tem o primeiro atributo com o valor ‘Andréia’, e o segundo atributo com o valor ‘Gualberto’, e assim sucessivamente.

2.3.3 JAX-RS

Para dar suporte ao REST no desenvolvimento de serviços *RESTful* foi criada a especificação Java API for *RESTful* Web Services (JAX-RS). A JAX-RS é uma linguagem de programação Java projetada para tornar mais fácil o desenvolvimento de aplicativos que usam a arquitetura REST. E para simplificar esse processo a JAX-RS usa anotações específicas de HTTP para definir os recursos e as ações que podem ser realizadas nesses recursos (ORACLE, 2013).

JAX-RS é uma especificação para desenvolvimento de Web Services baseados na arquitetura REST para a linguagem Java. É também uma especificação definida pelo JCP (*Java Community Process*) na JSR (*Java Specification Request)* 339 e encontra-se na versão 2.x.

Para facilitar o acesso aos recursos, a especificação JAX-RS define um conjunto de anotações. As principais anotações estão descritas no Quadro 9.

Quadro 9: Principais Anotações JAX-RS

Anotação	**Descrição**
@Path	Recebe uma string como parâmetro e indica qual é o *path* da URL.
@GET	Indica que corresponde ao método GET do HTTP, e serve para recuperar um recurso.
@POST	Indica que corresponde ao método POST do HTTP, e serve para criar um recurso.
@PUT	Indica que corresponde ao método PUT do HTTP, e serve para atualizar um recurso.
@DELETE	Indica que corresponde ao método DELETE do HTTP, e serve para excluir um recurso.
@PathParam	Indica qual parâmetro do método se refere ao parâmetro no *path*.
@QueryParam	Indica o nome do parâmetro que será recebido.
@Consumes	Indica qual o *mime-type* do conteúdo da requisição.
@Produces	Indica qual o mime-type do conteúdo da resposta que será enviada ao cliente.

Fonte: Adaptado de K19 TREINAMENTOS (2010).

As anotações exibidas no Quadro 9, são comumente utilizadas durante a implementação de Web Services *RESTful*, facilitando assim a manipulação dos recursos.

2.3.5 JERSEY

É uma *framework open source*, implementação de referência da especificação JAX-RS, que pode ser utilizado para desenvolver serviços web *RESTful*. Segundo (Bezerra, 2013), o Jersey tem basicamente um lado servidor e um lado cliente, e quando o lado cliente "*core-client*" comunica com o servidor, o Jersey usa um servlet que "escaneia" as classes pré-definidas para identificar os recursos REST. Atualmente a última versão estável do Jersey é a 2.12.

Todas as tecnologias e conceitos apresentados consistem na base de desenvolvimento do trabalho, sendo, para tanto, adotada metodologia de trabalho com ferramentas e demais elementos importantes para finalização do trabalho.

3. METODOLOGIA

Para elaboração do presente trabalho foi adotada a seguinte metodologia:

I. Primeiramente foi realizada uma pesquisa sobre diversos pontos relacionados ao trabalho, tais como, a plataforma de desenvolvimento Android, Georreferenciamento, e Web Services para tanto, realizadas consultas a livros e principalmente artigos disponíveis na internet. O resultado da pesquisa foi apresentado anteriormente no arcabouço teórico;

II. Após entender os conceitos e funcionamento desses assuntos, foi dado início a instalação e configuração das ferramentas de desenvolvimento e tecnologias utilizadas durante o projeto. A instalação e configurações foram realizadas em ambiente Windows 7 Professional, sendo as seguintes ferramentas: Kit Development Java (JDK) versão

7, Netbeans 7.4 com suporte aos plugins JavaServer Faces(JSF) e PrimesFaces, MySQL 5.6, MySQL Workbench 6.0 CE, Apache Tomcat 7.0.41, Jersey 1.17.1, Eclipse Kepler com o conjunto de *plugins* do Android Development Tools (ADT), e o Android Software Development Kit (SDK);

III. Com as ferramentas devidamente instaladas e configuradas, foi dado início ao desenvolvimento e consequentemente testes unitários. Durante esse período foram desenvolvidas pequenas aplicações utilizando a plataforma Android para teste e melhor entendimento das funcionalidades de seus componentes; aplicações envolvendo geolocalização; e, a implementação do *web service RESTful*.

Para um melhor entendimento, vale salientar que a ferramenta Netbeans foi utilizada para o desenvolvimento da aplicação web, utilizando a linguagem Java, mais especificamente JSF, *framework* que permite a criação de interfaces web juntamente com a biblioteca PrimeFaces, que realiza a modificação da aparência dos componentes em tempo de execução.

O MySQL e MySQL Workbench foram utilizados para o desenvolvimento do banco de dados para armazenamento das informações cadastradas pela aplicação web. Além disso, também foi utilizado o Apache Tomcat tanto na aplicação web, quanto no web service, sendo esse um servidor baseado em Java responsável pela execução de aplicações web.

Por fim, e não menos importante, foi o Eclipse, ferramenta em que foram implementados o *Web Service* e a aplicação principal desse trabalho, o aplicativo Nita Propagandas & Promoções. O eclipse permitiu o desenvolvimento do web service, porém, para que fosse possível programar em Android foi necessário adicionar o ADT Plugin.

Tudo devidamente configurado, foi permitido a Maquina Virtual Android, conhecida como o emulador Android, que o computador emule um aparelho celular. Apesar de existir o emulador do Android dependo das configurações do computador a execução da aplicação pode torna-se demorada, porém também

é permitido executar em qualquer aparelho que possui sistema operacional Android.

Para o desenvolvimento desse trabalho foi utilizado para execução e teste da aplicação celular Moto G da Motorola, e para que o computador conectar corretamente com o celular foi instalado ao computador o Motorola Device Manager que permite essa conexão por meio de cabo *Universal Serial Bus* (USB). Assim como a aplicação web, para o desenvolvimento da aplicação móvel também é necessário o JDK.

Para o desenvolvimento do *Web Service* foi criado projeto web dinâmico, e adicionados os *jars* do Jersey dentro da pasta WebContent/WEB-INF/lib do projeto. Nessa pasta também foi adicionado o *driver Java Database Connectivity* (JDBC) para MySQL, que permite que o *Web Service* envie instruções ao banco de dados.

Por fim, para o desenvolvimento do aplicativo foi criado um projeto Android chamado Nita. Ao criar o projeto foi definido o *minSdkVersion Application Programming Interface* (API) Nivel 13: Android 3.2(Honeycomb), o qual significa que a mínima a aplicação pode ser executada, e o targetSdkVersion que o nível de API que a aplicação foi destinada, sabendo que foi escolhida a API Nível 2:Android 4.4W (KitKat Wear).

Mais detalhes da implementação dessas aplicações serão descrito no próximo capítulo, que trata do processo de construção e resultados de cada aplicação desenvolvida.

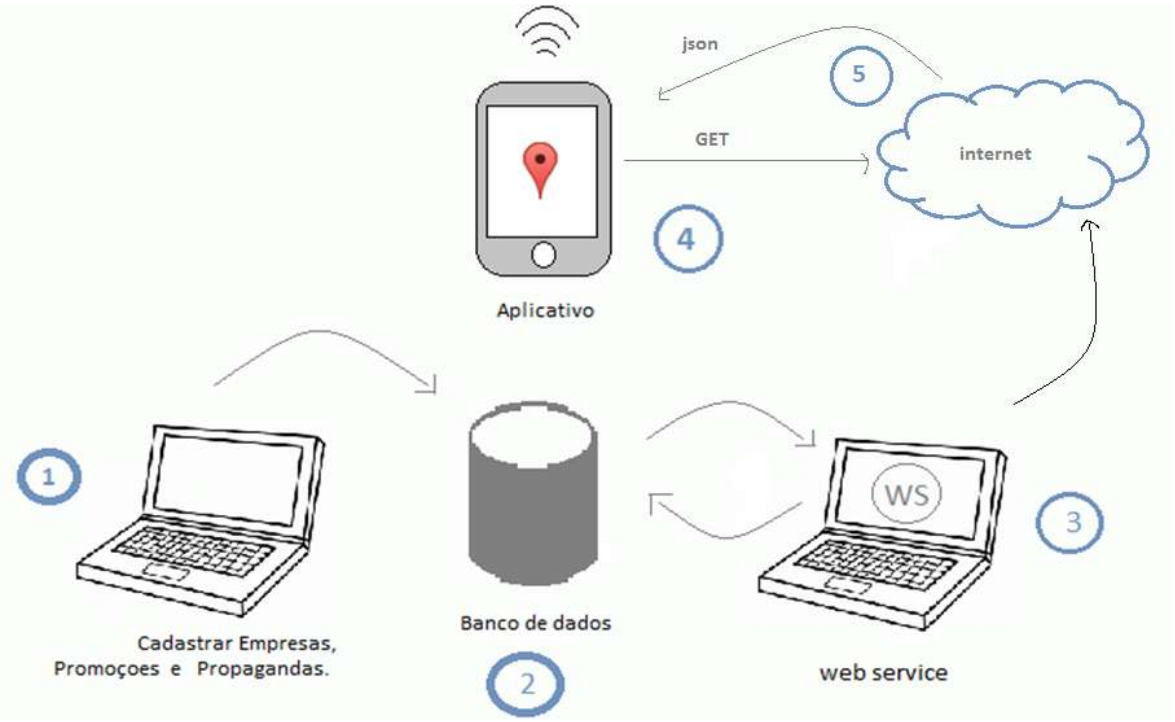
O trabalho foi desenvolvido nas dependências da Fundação Universidade do Tocantins, no Laboratório de Hardware. O equipamento utilizado consistiu em notebook DELL com 4 GB de memória e processador Core i3.

4. DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS

Este capítulo apresenta as partes do desenvolvimento do trabalho, descrevendo o processo de construção, as dificuldades, e soluções propostas para a resolução de cada situação.

Para um melhor entendimento da integração das diversas tecnologias e ferramentas a Figura 11 apresenta a arquitetura do sistema.

Figura 11: Arquitetura e Integração das Aplicações



Fonte: Elaborada pela autora.

Conforme o esquema apresentado na Figura 11, o processo de integração inicia-se no cadastro das empresas, suas propagandas e promoções na aplicação web. Ao cadastrar os dados, esses são armazenados no banco de dados.

A partir desse momento os dados podem ser mapeados e disponibilizados pelo serviço web de forma que dispositivos móveis possam fazer requisições por meio do método GET do protocolo HTTP, e de acordo com a localização geográfica os dados são retornados em formato JSON, manipulados e devidamente apresentados ao usuários da aplicação.

De acordo com a ordem apresentada no esquema, torna-se importante apresentar no próximo tópico o processo de implementação da aplicação web. Vale lembrar que serão descrita apenas as partes mais relevantes do desenvolvimento, isso ainda ao se considerar que essa não é a aplicação objetivo do trabalho.

4.1 Implementação Aplicação Web

Para que os dados das empresas estivessem prontos para serem disponibilizados na web, foi necessário que estes fossem cadastrados em um banco de dados. Devido a essa necessidade, foi implementado um sistema web que permitisse a manipulação de empresas, propagandas e promoções.

Para obter-se uma melhor compreensão do funcionamento desse sistema no Quadro são apresentados os requisitos funcionais e não funcionais nos quais a aplicação foi baseada.

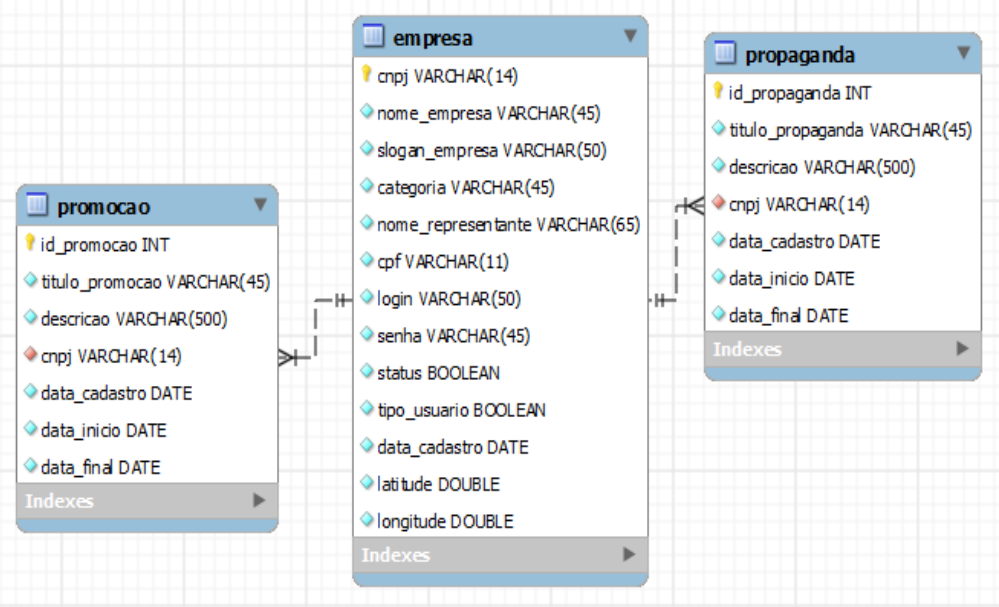
Quadro 10: Requisitos Funcionais e Não Funcionais

Funcionais:
RF01 – Permitir ao administrador cadastrar empresas;
RF02 - Permitir login ao usuário;
RF03 – Permitir ao administrador editar empresas cadastradas;
RF04 – Permitir ao administrador listar empresas cadastradas;
RF05 – Permitir ao administrador excluir empresas cadastradas;
RF06 - Permitir ao usuário empresa cadastrar propagandas;
RF07 – Permitir ao usuário empresa editar propagandas cadastradas;
RF08 – Permitir ao usuário empresa listar propagandas cadastradas;
RF09 – Permitir ao usuário empresa excluir propagandas cadastradas;
RF10 - Permitir ao usuário empresa cadastrar promoções;
RF11 – Permitir ao usuário empresa editar promoções cadastradas;
RF12 – Permitir ao usuário empresa listar promoções cadastradas;
RF13 – Permitir ao usuário empresa excluir promoções cadastradas;
Não Funcionais
RFN01 - O sistema será operado em ambiente Windows;
RFN02 - O sistema foi desenvolvido na Linguagem de Programação Java.
RFN03 – O sistema utiliza o banco de dados MySQL e MySQL Workbench.

RFN04 – O usuário deve está cadastrado para ter acesso ao sistema;

Após realizar o levantamento dos requisitos, foi desenvolvido o banco de dados para armazenar as informações das empresas e suas respectivas promoções e propagandas, conforme apresentado na Figura 12.

Figura 12: Diagrama do Banco de dados da Aplicação Web



Fonte: Elaborada pela autora

Como a aplicação Android tinha como objetivo apresentar as empresas de acordo com a sua localização geográfica, ao cadastrar uma empresa é obrigatório cadastrar suas coordenadas geográficas, para isso pode-se observar na Figura 12 que na tabela Empresa foi adicionado os campos latitude e longitude.

Para o preenchimento dessas duas colunas do banco de dados foi adicionado ao formulário de cadastro o componente <p:gmap> do PrimeFaces, como apresentado no Quadro 9, que cria um Google Maps dentro da página, que permite a captura aproximadamente da latitude e longitude de determinada empresa.

Quadro 11: Código de inserção do GMap

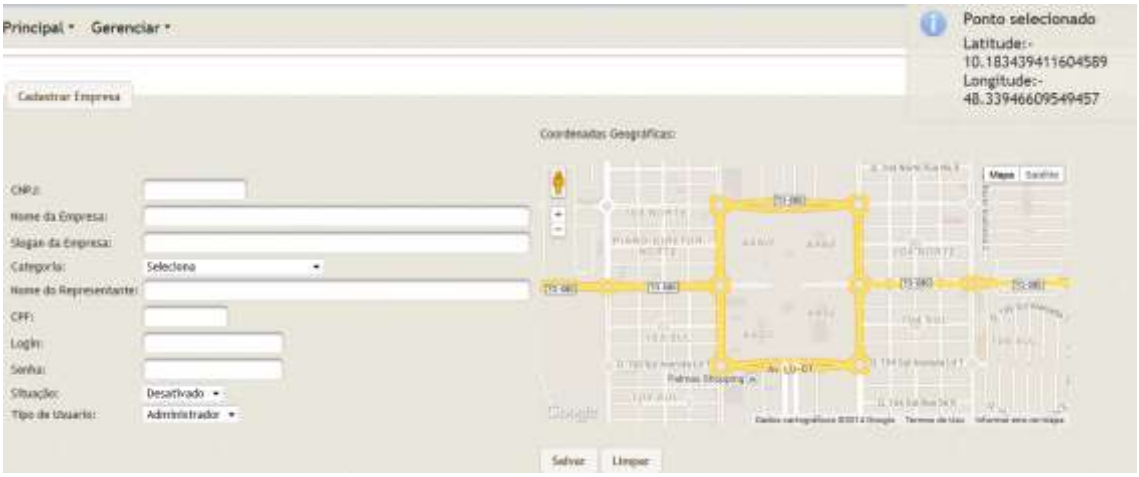
```
<p:gmap model="#{empresacontroller.emptyModel}" center="-10.18644405505313,-
48.327559250000036" zoom="15" type="ROADMAP" style="width:600px;height:400px"
id="map">
<p:ajax event=”pointSelect” listener=”#{empresacontroller.PontoSelecionado}”
update=”growl”/>
```

No Quadro 11 é apresentado o código para criação do campo do formulário de entrada dos dados geográficos com algumas propriedades, que para melhor entendimento serão explicados. A propriedade *model* é responsável por informar quem receberá esse valor na classe EmpresaController. A propriedade *center* define onde o mapa inicialmente vai abrir, por meio da coordenadas geográficas definidas. O código atual possui coordenadas geográficas da cidade de Palmas.

Outra propriedade importante é o *type* que define o tipo de mapa. O tipo *roadmap* determina que o mapa apresente apenas o mapa, levando-se em consideração que outras possibilidades de apresentação de mapas em uma página web podem existir, sendo elas: mapa e satélites, somente satélites, somente terrenos, dentre outras.

Ainda no Quadro 11, a *tag* <p:ajax> é composta pelas propriedades *listener* e *update*. A primeira propriedade chama o método que guarda a latitude e longitude do ponto selecionado, como está devidamente apresentado no Quadro 12. A segunda propriedade atualiza o *growl*, mensagem que aparece no canto superior da tela, como pode-se observar na Figura 13, informando ao usuário que um ponto geográfico foi selecionado.

Figura 13: Tela de cadastro das Empresas



Neste formulário de cadastro da empresa, além do campo das coordenadas geográficas, existem também outros campos, que apesar de serem básicos, também são de preenchimento obrigatório. Dentre eles pode-se citar o CNPJ da empresa, o nome da empresa, o slogan, a categoria, o *login* e senha para acesso a página de cadastro das propagandas e promoções. Ainda referindo-se coordenadas geográficas, o Quadro 11 apresenta como captura o ponto selecionado.

Quadro 12: Método para captura de latitude e longitude

```
public void PontoSelecionado(PointSelectEvent event) {
   LatLng latlng = event.getLatLng();
   emptyModel = new DefaultMapModel();
   empresaModel.setLatitude(latlng.getLat());
   empresaModel.setLongitude(latlng.getLng());
   FacesContext.getCurrentInstance().addMessage(
           null,
           new FacesMessage(
                   FacesMessage.SEVERITY_INFO,
                   "Ponto selecionado",
                   "Latitude:"+latlng.getLat()+"Longitude:"+latlng.getLng()
           )
   );
 }
```

Além de alterar os valores das varáveis latitude e longitude para as coordenadas atuais, o método presente no Quadro 12 também faz com que a mensagem com qual ponto geográfico foi selecionado apareça, por meio da classe *FacesMessage*.

O GMap que é uma biblioteca da Google, que fornece o componente <g:map> e para que esteja disponível na página é necessário adicionar uma API JavaScript do Google Maps, conforme pode-se observar no Quadro 13.

Quadro 13: API JavaScript Google Maps

```
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=true" />
```

Esse código em sua maioria das vezes deve ser colocado dentro da *tag* <h:header> do formulário.

Após cadastrar o usuário, se este estiver com o seu status ativo, o mesmo já pode realizar a manipulação de propagandas e promoções. Para isso, basta que seja realizado o login utilizando o usuário e senha que fora

escolhido ao cadastrar a empresa, por meio das telas apresentadas na Figura 14.

Figura 14: Tela de cadastro de Propagandas e Promoções

Na Figura 14, pode-se observar que ao cadastrar uma propaganda ou uma promoção o usuário deverá entrar com o título, a descrição, a data de início e data final. Esses dois últimos campos citados são de fundamental importância para definir se determinada propaganda ou promoção deverá aparecer quando o usuário realizar a busca por meio da aplicação móvel.

O campo data início definirá a partir de que dia essa promoção ou propaganda será válida, e o campo data fim definirá até que dia a propaganda ou promoção poderá ficar visível, ou seja, até quando esta será válida.

Vale salientar que essa aplicação web possui outras páginas, como pode perceber ao ver os requisitos funcionais da aplicação, porém foram descritas apenas as mais importantes.

Após a conclusão da construção da aplicação web, foi dado início ao desenvolvimento do *Web Service*, o processo de desenvolvimento encontra-se no próximo tópico, e é de fundamental importância, pois permite a disponibilização dos dados cadastrados na web a serem consumidos.

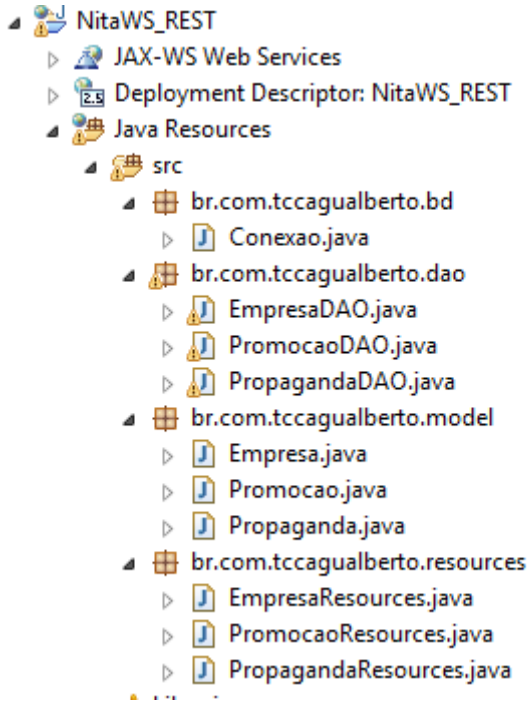
4.2 Implementação *Web Service RESTful*

Após o cadastramento dos dados por meio da aplicação web, esses dados foram trabalhados de forma que ficassem disponíveis para serem consumidos pela aplicação móvel. Devido a essa necessidade, foi desenvolvido o Web Service, sistema este que permite a interoperabilidade de

sistemas, capaz de acessar o banco de dados com informações sobre as empresa.

Como pode-se observar na Figura 15, foram criadas as classes Conexão, responsável por realizar a conexão com o banco de dados, bem como as classes EmpresaDAO, PromocaoDAO, e PropagandaDAO, no qual estão localizados os métodos que manipulam os dados gravados no banco. Por fim, e certamente as classes mais importantes por trazerem um conceito novo, se tem o pacote resources com as classes EmpresaResources, PromocaoResources, e ropagandaResources.

Figura 15: Classes e importações para criar o web service



Além das classes apresentadas na Figura 15, foi alterado o arquivo XML web.xml que existe na pasta WebContent/WEB-INF/ como pode-se observar no Quadro 14.

Quadro 14: Configuração do servidor

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"                id="WebApp_ID"
version="2.5">
```

```
  <display-name>NitaWS_REST</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>
  com.sun.jersey.spi.container.servlet.ServletContainer
</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>
    br.com.tccagualberto.resources
  </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

O xml apresentado no Quadro 14 é composto várias *tags*. Dentre elas a <display-name> no qual é definido o nome do web service, a *tag* <servlet-class> que define o servlet do Jersey, a *tag* <init-param> onde é passado como parâmetro a pasta onde estão localizados os Web Services, que no caso, é a pasta br.com.tccagualberto.resources. Dentro da *tag* de mapeamento está a *tag* <url-pattern>, por meio dela é declarada a URL padrão para localizar os recursos, de acordo com a configuração, o caminho para a localização dos recursos é realizado através da URL: http://endereçoIP/NitaWS_REST/rest/. Para chegar aos recursos pretendidos é necessário concatenar o caminho especifico de cada recurso a essa URL.

Após configurar o *web service* para receber as requisições do HTTP, foram criadas as classes para a manipulação dos recursos. No Quadro 15 é exposto o método da classe EmpresaResources.

Quadro 15: Método que retorna JSON com empresas

```
@Path(“empresastxt”)
@GET
@Produces("application/json ; charset=UTF-8")
public List<Empresa>getEmpresasTXT(){
       return EmpresaDAO.buscarEmpresas();
}
```

O Quadro 15 inicia-se com a anotação @Path que informa qual caminho deve ser concatenado a URL padrão para acessar o serviço que esse método retorna. Logo em seguida tem a anotação @GET, que informa qual o método HTTP será utilizado, ou seja, informa ao método getEmpresasTXT que

é pra realizar uma busca de dados. Quando esse método é solicitado a busca é realizada e retorna todas as empresas cadastradas que estão com o status ativo.

Vale salientar que nas classes DAO(*Data Access Object*) fora implementado somente métodos que realiza-se busca de dados, pois a aplicação Android não fará nenhum tipo de inserção, atualização, e remoção dos dados armazenados.

Sabe-se que para acessar e manipular os serviços disponíveis no Web Service, faz se o uso dos métodos do protocolo HTTP. Porém, pelo mesmo motivo citado anteriormente, foi utilizado apenas o método GET, mais precisamente com auxílio da anotação @GET, como pode-se observar no Quadro 16.

Quadro 16: Classe PromocaoResources

```
@Path("/promocoes")
public class PromocaoResources {
	@Path("{cnpj}")
	@GET
	@Produces("application/json ; charset=UTF-8")
	public List<Promocao> getPromocaos
	( @PathParam("cnpj") String cnpj){
		return PromocaoDAO.buscar(cnpj);
	}
}
```

O Quadro 16 tem por objetivo retornar uma lista de promoções cadastradas as quais possuem o CNPJ igual ao passado para anotação @PathParam. O resultado dessa solicitação, assim como as demais que serão realizadas pelo aplicativo desenvolvido, será no formato JSON como pode-se notar na anotação @Produces presente no Quadro 16.

Na declaração do @Produces, além de conter qual o formato dos dados a serem retornados, também está declarado o *charset*=UTF-8. Apesar do banco de dados, o próprio *Web Service* como pode ser observado na configuração do servidor no Quadro 14, e aplicação móvel que consome esses dados estarem com o encoding definido como UFT-8, foi necessário acrescentar o *charset* para que informações buscadas por este método não apareçam distorcidas, como pode-se ver na Figura 16.

Figura 16: Sem configurar o *charset*

Na Figura 16, é exposta uma lista de empresas, e como pode-se observar as palavras que contêm acento ou cedilha estão todas com seus caracteres distorcidos.

Vale ressaltar que outros recursos também foram retornados para o real funcionamento do aplicativo móvel, sendo eles, a lista de empresas com seus respectivos dados de acordo com a localização geográfica, e as propagandas de acordo com a empresa selecionada, para isso cada um possui o seu método específico semelhante ao exposto no Quadro 15.

Como foi citado anteriormente, na implementação do *Web Service* existem as classes DAO para realizar busca no banco de dados. No Quadro 17, é apresentado o método buscar chamado na classe PromocaoResources exposto no Quadro 16.

Quadro 17: Método buscar promoções

```
public static ArrayList<Promocao> buscar(String cnpj) {
		Conexao con = new Conexao();
		ArrayList<Promocao> res = new ArrayList<Promocao>();
		String query = "SELECT id_promocao,titulo_promocao, descricao,cnpj,"
				+ " DATE_FORMAT(data_cadastro, '%d-%m-%Y') AS
data_cadastro,"
				+ " DATE_FORMAT(data_inicio, '%d-%m-%Y') AS data_inicio,"
				+ " DATE_FORMAT(data_final, '%d-%m-%Y') AS data_final
FROM promocao"
				+ " WHERE data_inicio<=now() and data_final>=now() and
cnpj = ?";
		con.preparar(query);
		try {
			con.getPstmt().setString(1, cnpj);
			ResultSet rs = con.executeQuery();
			while (rs.next()) {
				res.add(new Promocao(
```

```
                                        rs.getInt("id_promocao"),
                                        rs.getString("titulo_promocao"),
                                        rs.getString("descricao"),
                                        rs.getString("cnpj"),
                                        rs.getString("data_cadastro"),
                                        rs.getString("data_inicio"),
                                        rs.getString("data_final")));
                        }
                } catch (SQLException ex) {
                        Logger.getLogger(PromocaoDAO.class.getName()).log(Level.SEVERE,
                                        null, ex);
                } finally {
                        con.fechar();
                        return (res);
                }
        }
```

No Quadro 17, é realizada uma busca que seleciona promoções cadastradas no banco de dados e que possui o CNPJ igual ao CNPJ que foi passado como parâmetro. Após a busca, esses arrays de dados são retonados para a classe PromocaoResources.

Com os dados das empresas disponíveis para serem consumidos, foi dado início a implementação da aplicação móvel. A mesma será detalhada no próximo tópico.

4.3 Implementação Aplicativo Móvel

Para a implementação da proposta principal desse trabalho, o desenvolvimento do aplicativo móvel, foi realizado um levantamento de requisitos da aplicação, os quais estão descritos no Quadro 18.

Quadro 18: Requisitos Funcionais e Não Funcionais

Requisitos Funcionais:
RF01 – O aplicativo busca de dados em web service.·.
RF01 – O aplicativo busca empresas dentro de determinado raio entre um e 50 quilômetros;
RF03 – O aplicativo busca empresas escolhendo ou não a categoria;
RF04 – O aplicativo busca propagandas de determinada empresa;
RF05 – O aplicativo busca promoções de determinada empresa;
RF06 – O aplicativo permite compartilhar propagandas;
RF07 – O aplicativo permite compartilhar promoções;
Requisitos Não Funcionais:

RNF01 - O aplicativo é compatível com o Sistema Operacional Android versão 3.2 e 4.4;
RNF02 - O aplicativo foi desenvolvido na linguagem de programação Java;
RFN03 – O aparelho que o aplicativo vai executar precisa ter GPS;
RFN04 – O aparelho precisa ter acesso à internet;

O aplicativo recebeu o nome de Nita Propagandas & Promoções e tem por objetivo realizar a integração de tudo que já foi desenvolvido anteriormente. A integração acontece pelo fato de que a aplicação móvel irá consumir os dados cadastrados pela aplicação web, e para consumir precisa que o *Web Service* disponibilize os dados.

Ao criar o projeto do aplicativo, o Eclipse cria automaticamente o *AndroidManifest* para adicionar as configurações do projeto, sendo que estes por sua vez está exposto no Apêndice I.

No *AndroidManisfest*, estão presentes todas as *Activities* que fazem parte do projeto com suas devidas configurações. Vale lembrar que todas as *activities* estão com a screenOrientation (orientação de telas) definidas como *portrait* (retrato). Isso significa que ao movimentar o dispositivo móvel em sentido horário ou anti-horário do aplicativo irá permanecer da mesma forma, com orientação de tela retrato.

O aplicativo permite que de acordo com a localização geográfica o usuário possa ver as empresas em determinado raio de distância, por esse motivo surgiu à necessidade da aplicação pegar a localização atual do dispositivo móvel, o código de como isso é capturado é apresentado no Quadro 19.

Quadro 19: Captura das coordenadas geográficas

```
...

if (isGPSEnabled) {
            if (location == null) {
                  locationManager.requestLocationUpdates(
                              LocationManager.GPS_PROVIDER,
                              MIN_TIME_BW_UPDATES,
                              MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
                        if (locationManager != null) {
                        location = locationManager
                  .getLastKnownLocation(LocationManager.GPS_PROVIDER);
                        if (location != null) {
                              latitude = location.getLatitude();
```

```
                                        longitude = location.getLongitude();
                                    }
                                }
                            }
                        }
        ...
```

Antes de pegar a localização é preciso verificar se o GPS está ativo, e caso não esteja, oferece-se a opção para o usuário ativar. O código completo da busca da localização atual e configuração do GPS estão presentes no Anexo I.

Após se obter a latitude e longitude, é realizado um calculo da distância entre as coordenadas atuais e as coordenadas das empresas armazenadas no banco de dados. Para facilitar o cálculo, todas as empresas cadastradas e ativas são salvas em um arquivo de extensão “.txt”, para que no momento do cálculo não haja a necessidade de buscar a localização das empresas uma de cada vez, reduzindo dessa forma o tempo de processamento.

Esse cálculo está exposto no Quadro 20, na biblioteca DistanciaCoordenadas criada especialmente para resolver essa situação.

Quadro 20: Biblioteca que calcula distância entre coordenadas

```
package br.com;
public class DistanciaCoordenadas {
/**
 * Titulo: DistanciaEntreCoordenadas
 * Descrição: Programa utiliza a fórmula
 * de Harvesine parar retorna a quantidade de metros existentes entre dois
 * pontos geográficos. Para realizar o cálculo é preciso informar a latitude
 * e longuitude dos pontos de origem e destino, o resultado será retornado
 * em metros.*
 *
 * @author Andréia Gualberto Pereira
 * @version 1.0
 * @created 16 de setembro de 2014
 */
static double raio_terra = 6371;
double diferencaLongitude = 0, diferencaLatitude = 0, senoLatitude = 0,
             senoLongitude = 0;
double latitude1_radiano = 0, latitude2_radiano = 0,
             longitude1_radiano = 0, longitude2_radiano = 0;
double dMetros = 0, vrX = 0, vrY = 0;

public DistanciaCoordenadas() {
}

public int calcularDistancia(double latitude_1, double longitude_1,
             double latitude_2, double longitude_2) {
      /* converte para radiano */
```

```
        latitude1_radiano = Math.toRadians(latitude_1);
        longitude1_radiano = Math.toRadians(longitude_1);
        latitude2_radiano = Math.toRadians(latitude_2);
        longitude2_radiano = Math.toRadians(longitude_2);

        /* diferença entre coordenadas */
              diferencaLatitude = (latitude2_radiano - latitude1_radiano);
              diferencaLongitude = (longitude2_radiano - longitude1_radiano);

              senoLatitude = Math.sin(diferencaLatitude / 2);
              senoLongitude = Math.sin(diferencaLongitude / 2);

              vrX = (senoLatitude * senoLatitude) + Math.cos(latitude_1)
                           * Math.cos(latitude_2) * (senoLongitude *
senoLongitude);
              vrY = 2 * Math.asin(Math.min(1.0, Math.sqrt(vrX)));

              dMetros = raio_terra * vrY * 1000;
              return (int) dMetros;
        }
}
```

Fonte: Elaborado pela autora baseado na fórmula de Haversine

Esse é certamente um dos principais pontos do trabalho, já que está entre seus objetivos existe realizar o processo de georreferenciamento. Para chegar à solução não foi um processo fácil, pois tem a necessidade de considerar a curvatura da terra. A solução mais adequada foi implementar uma biblioteca baseada na fórmula de Haversine, sendo esta uma equação bastante utilizada quando em navegações. O Quadro 21 apresenta a Fórmula de Haversine.

Quadro 21: Fórmula de Haversine

$$\text{haversin}\left(\frac{d}{R}\right) = \text{haversin}\,(\Delta\emptyset) + \cos(\emptyset 1)\cos(\emptyset 2)\,\text{haversin}(\Delta\lambda)$$

Fonte: LONGITUDESTORE (2008)

Na fórmula exposta no Quadro 21, o ‘R’ refere-se ao raio da terra que equivale a 6378 quilômetros, $\Delta\emptyset = \emptyset 1 - \emptyset 2$ refere-se a diferença entre latitude do ponto inicial e a latitude do destino, e $(\Delta\lambda)$ refere-se a diferença entre longitudes. Ainda no mesmo quadro pode-se observar que para obter o resultado da fórmula é preciso aplicar a lei dos cossenos.

A biblioteca apresentada no Quadro 20 retorna em metros a distância entre os dois pontos geográficos passados por parâmetros. A classe que

recebe esse valor quando necessário converte para quilômetros o valor retornado.

Vale ainda ressaltar que o arquivo ".txt" é atualizado toda vez que for manipular o sistema, de forma que se outros dados forem cadastrados o usuário terá acesso.

Sabendo a distância que existe entre as coordenadas atuais, e a coordenadas da empresa, é possível definir se determinada empresa está ou não dentro do raio de busca que o usuário escolheu, e assim o usuário poderá ter acesso às propagandas e promoções cadastradas por essas empresas.

A estrutura utilizada para fazer a listagem das empresas é semelhante à utilizada para fazer a listagem das promoções e propagandas. O que muda são apenas as variáveis, e o fato de que na listagem da empresa é realizado o cálculo da distância.

Para realizar essa listagem é preciso acessar o *Web Service* utilizando a classe AsyncTask. De forma a facilitar a compreensão será explicado como fazer isso usando como exemplo a listagem de promoções.

Enquanto são realizadas as buscas, é apresentada uma mensagem informando que está sendo realizado, essa mensagem é configurada no método onPreExecute(). O método onPreExecute() da *Activity* que lista as promoções é apresentado no Quadro 22.

Quadro 22: Método onPreExecute()

```
@Override
protected void onPreExecute() {
       super.onPreExecute();
       pDialog = new ProgressDialog(CPromocoes.this);
       pDialog.setMessage("Buscando Promoções ...");
       pDialog.setIndeterminate(false);
       pDialog.setCancelable(true);
       pDialog.show();
}
```

A mensagem "Buscando Promoções..." é exibida até quando a execução da URL é finalizada, nesse momento o método doInBackground que está exposto no Quadro 23 é chamado.

Quadro 23: Método doInBackground()

```
 @Override
```

```
	protected JSONObject doInBackground(String... params) {
		AcessarWS ws = new AcessarWS();
		String url2 = url.concat(cnpjBuscar);
		String jsonStr = ws.consumirDados(url2);
		if (jsonStr != null) {
			JSONObject jsonObj = null;
			try {
				jsonObj = new JSONObject(jsonStr);
				return jsonObj;
			} catch (JSONException e) {
				e.printStackTrace();
			}
		}
		return null;
	}
```

No método doInBackground() é criada uma instância da classe AcessarWS, que será apresentada no Quadro 25, para onde é passado por parâmetro uma URL para acessar um recurso.

Para acessar os dados disponíveis no *Web Service* o aplicativo passa uma URL de acordo com o recurso que deseja consumir. No Quadro 23 é apresentada a URL que é executada para buscar as promoções cadastradas por determinada empresa.

Quadro 24: Exemplo de URL para acessar recursos

```
http://192.168.43.211:8080/NitaWS_REST/rest/promocoes/12345634234562
```

O endereço IP (*Internet Protocol*) é onde está disponível o Web Service, e o número “12345634234562” é o CNPJ da empresa. Para a execução dessa URL é chamado o método consumirDados() da classe AcessarWS que encontra-se exposta no Quadro 25.

Quadro 25: Classe de acesso ao *Web Service*

```
public class AcessarWS {
	static String resposta = null;
	HttpEntity httpEntity = null;
	HttpResponse httpResponse = null;

	public AcessarWS() {
	}
	public String consumirDados(String url) {
		try {
			DefaultHttpClient httpClient = new DefaultHttpClient();
			HttpGet httpGet = new HttpGet(url);
			httpResponse = httpClient.execute(httpGet);
			httpEntity = httpResponse.getEntity();
			resposta = EntityUtils.toString(httpEntity);
		} catch (UnsupportedEncodingException e) {
			e.printStackTrace();
		} catch (ClientProtocolException e) {
```

```
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
                return resposta;
        }

}
```

Dessa forma, ao realizar essa requisição será retornado um JSON com o recurso solicitado, como se pode observar no exemplo exibido Quadro 26.

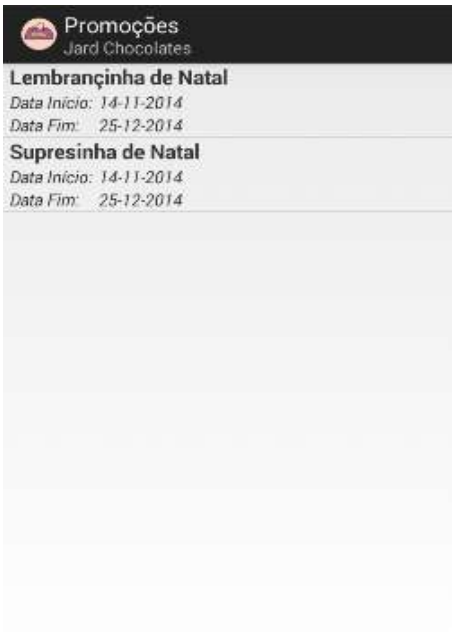
Quadro 26: Resposta da solicitação


```
{"promocao":[{"cnpj":"12123211234234","data_cadastro":"14-11-
2014","data_final":"25-12-2014","data_inicio":"14-11-2014","descricao":"Chegou
o Natal! Que tal presentear quem você ama como délicias em chocoloate, a
promoção #lembraçinhadeNatal tem descontos especiais. Não perca tempo, venha
comprar.","id_promocao":"6","titulo_promocao":"Lembrançinha de
Natal"},{"cnpj":"12123211234234","data_cadastro":"14-11-
2014","data_final":"25-12-2014","data_inicio":"14-11-2014","descricao":"Chegou
o Natal! Que tal presentear quem você ama como délicias em chocoloate, a
promoção #supresinhadeNatal tem descontos especiais. Não perca tempo, venha
comprar.","id_promocao":"7","titulo_promocao":"Supresinha de Natal"}]}
```


No Quadro 26, a solicitação realizada retornou uma array com todos os dados de duas promoções cadastradas pela empresa. Na implementação do AsyncTask, após a execução do doInBackground(), o método onPostExecute() recebe os dados que o primeiro retorna, e quando esse retorno não é vazio os dados são colocados numa ListView e apresentados ao usuário.

O resultado desse processo é apresentado na Figura 17. O código do método onPosExecute() por ser extenso pode ser analisado no Apêndice II. Quando esse retorno é vazio, ou seja, a empresa não possui nenhuma promoção cadastrada, aparece uma mensagem informando isso ao usuário.

Figura 17: Listagem de Promoções



Na listagem da promoção verifica-se o título da promoção, juntamente com sua data de início e data final. Para saber mais informações, basta selecionar a promoção. Vale ressaltar que as promoções e propagandas podem ser compartilhadas. No Quadro 27, é exposto o código que permite esse o compartilhamento de uma promoção.

Quadro 27: Código Fonte para compartilhar promoções

```
	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		getMenuInflater().inflate(R.menu.main, menu);
		MenuItem item = menu.findItem(R.id.menu_item_share);
		share_action_provider = (ShareActionProvider)
item.getActionProvider();
		share_action_provider

	.setShareHistoryFileName(ShareActionProvider.DEFAULT_SHARE_HISTORY_FILE_
NAME);
		share_action_provider.setShareIntent(compartilharPromocao());
		return true;
	}
	private Intent compartilharPromocao() {
		Intent intent = new Intent(Intent.ACTION_SEND);
		intent.setType("text/plain");
		intent.putExtra(
			Intent.EXTRA_TEXT,
			"Venha conferir a promoção da "+ nomeEmpresa+ "!!\n"+
"Titulo da Promoção:"
			+ titulo_promocao+ "\n"+ " Válida de "+ dataInicio+ " até
"+ dataFinal
			+ "\n (Baixe o app Nita Propagandas & Promoções e"
			+ " confira as novidades mais próxima da sua
localização!)");
		return intent;
	}
```

O compartilhamento das promoções e propagandas podem ser realizado para diversos outros aplicativos, isso é nativo da plataforma Android, e que enriquece as possibilidades da aplicação.

Um dos problemas enfrentados durante o desenvolvimento do trabalho foi o fato de muitas das vezes ao construir algo em XML, como por exemplo, um layout, os mesmos não serão referenciados na Classe R da aplicação. Devido essa situação o projeto apresentava erros e não permitia ser executado.

Para resolver essa situação, sempre era necessário ir até a barra de ferramentas do Eclipse em *Project->Clean*, e posteriormente *Project->Build Project.* Ao fazer isso, a classe R era apagada, e construída devidamente quando não tinha nenhum erro nos XML.

Para um melhor entendimento do funcionamento da aplicação foi criado o tópico seguinte. Nele as telas do aplicativo Nita Propagandas & Promoções são exploradas passo a passo.

4.3.1 Funcionamento do Aplicativo

Para uma melhor visualização dos resultados obtidos será descrito a seguir todos os procedimentos necessários à utilização da aplicação. A Figura 18 apresenta a primeira tela do aplicativo. Na mesma encontram-se dois botões que permite a entrada ao sistema.

Figura 18: Tela de Entrada

O primeiro botão leva o usuário à tela de informações do aplicativo, onde podem ser visualizadas as informações do sistema. As informações do sistema se divide em duas telas, uma com informações gerais do sistema, tais como, versão, desenvolvedor, e as informações de como o sistema funciona, que pode ajudar o usuário a manusear o aplicativo.

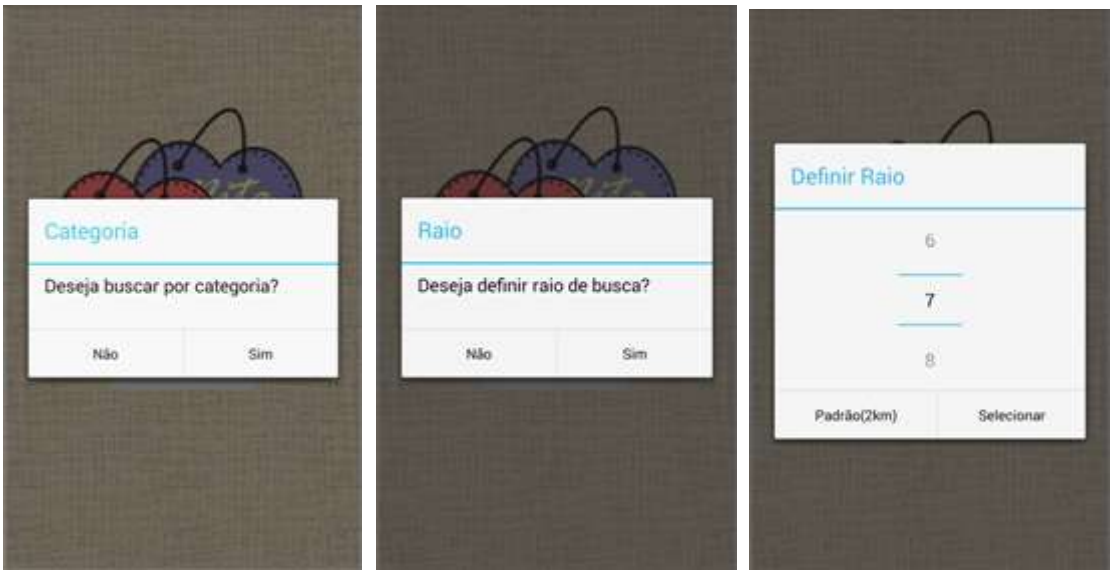
O outro botão da tela inicial do aplicativo apresentado é a entrada para a pesquisa de empresas, propagandas, promoções, e demais informações relativo a essas citadas anteriormente. A pesquisa por empresas somente pode ser realizada se o dispositivo estiver conectado a internet, caso contrário uma *intent* fará com que a aplicação volte para a tela inicial.

Além da internet também é necessário o acesso ao GPS, caso esse não esteja ligando o aplicativo solicita a configuração do GPS. Essa duas situações são apresentadas na Figura 19.

Figura 19: Tela de Entrada e Tela de Configuração do GPS.

Antes de realizar a busca o aplicativo solicita ao usuário para definir algumas configurações da busca como pode-se visualizar na Figura 20.

Figura 20: Configuração da Busca.



Primeiramente aparece uma *pop-up* verificando se o usuário deseja realizar uma busca de empresas por categoria ou não. Logo em seguida surge outra *pop-up* verificando se o usuário deseja definir o raio de busca, e caso ele opine por definir o raio de busca surge então um NumberPicker aonde o usuário poderá optar por um raio de busca padrão que é de 2 quilômetros, ou escolher de entre 1 e 50 quilômetros.

Se o usuário escolher realizar busca por categoria, o mesmo poderá selecionar qual categoria, como pode-se observar na Figura 21.

Figura 21: Seleção da Categoria de Busca.

Realizar busca por categoria pode ser mais vantajoso, pois ao selecionar a categoria é realizado um filtro, aparecendo dessa forma somente às empresas que o usuário deseja procurar naquele momento.

Após definir as configurações da busca as empresas são apresentadas de acordo com essas configurações. Na Figura 22, são apresentados dois casos, na primeira tela uma busca sem categoria e com raio padrão, e a segunda uma busca com a categoria saúde e raio de 30 quilômetros.

Figura 22: Listagem das Empresas.

Cada empresa pode conter propagandas e promoções cadastradas, para verificar basta selecionar a empresas. Na Figura 23, é apresentada a lista de propagandas e promoções da empresa Espaço BellaVita.

Figura 23: Propagandas e Promoções da empresa BellaVita

Na tela que lista as propagandas ou promoções, como pode-se observar na Figura 23, é apresentado o nome, data início e data final. Caso o usuário queria saber mais detalhes, basta selecionar a propaganda ou promoção desejada e essa será apresentada de maneira mais detalhada, como pode-se observar na Figura 24.

Figura 24: Promoção Selecionada

Após a selecionada, as propagandas e promoções podem ser compartilhadas. Para tanto, basta clicar no ícone compartilhar que aparece no canto superior da tela. Na Figura 25 é exposto um exemplo de compartilhamento de promoção.

Figura 25: Compartilhar Promoção.

Com isso, a aplicação possibilita que sejam trabalhados aspectos de marketing direcionado, sem que para tanto seja feito qualquer tipo de esforço, sendo que assim, clientes realmente interessados possam descobrir vantagens de empresas locais.

5. CONSIDERAÇÕES FINAIS

Este trabalho apresentou o desenvolvimento de três aplicações, sendo elas uma aplicação web, um *Web Service* e um aplicativo para dispositivos com Sistema Operacional Android o qual o recebeu o nome de Nita Propagandas & Promoções. Assim, os objetivos propostos foram alcançados pelo projeto. Vale salientar que as duas primeiras foram desenvolvidas com a finalidade de proporcionar a total funcionalidade desta última.

Os sistemas desenvolvidos como um todo permite o cadastro de empresas com sua localização geográfica, e depois desse cadastro realizado a empresa tem a possibilidade de cadastrar propagandas e promoções. Essas empresas e suas respectivas propagandas e promoções são disponibilizados pelo *web service RESTful*, e de acordo com a localização geográfica um usuário do aplicativo Android pode definir um raio de busca de até 50 quilômetros, e dessa forma ter acesso a propagandas e promoções cadastradas pelas empresas. Além disso, o usuário também tem a possibilidade de realizar o compartilhamento das mesmas.

Ao realizar este trabalho, pode-se perceber que o estudo da plataforma Android consiste de uma atividade interessante, pois não se trata de tecnologia complexa, principalmente para profissionais que tem um conhecimento prévio sobre a linguagem de programação Java. A mesma possui várias classes importantíssimas que possibilitam o desenvolvimento de aplicações úteis para o dia-a-dia, com facilidades que os dispositivos oferecem atualmente, tais como GPS, acesso a redes 3G e WI-FI, etc.

O *web service RESTful* e a API Jersey também foi de grande importância para a construção desse trabalho, pois essas tecnologias permitiram a integração dos sistemas e disponibilização de recursos para serem consumidos pela aplicação móvel e, além disso, são de fácil compreensão.

O principal diferencial do aplicativo Nita Propagandas & Promoções é certamente a capacidade de buscar empresas de acordo com a localização geográfica. Para conseguir isso, foi desenvolvida uma biblioteca baseada na fórmula de Haversine que permite o calcular a distância entre coordenadas

geográficas e retorna-la em metros. O desenvolvimento dessa funcionalidade foi um dos pontos mais trabalhosos, porém gratificante, pois a mesma poderá facilitar o trabalho de pessoas que venham a querer trabalhar com cálculo de distância entre coordenadas, a biblioteca desenvolvida para solucionar essa questão é apresentada durante o trabalho e será disponibilizada juntamente com o esse trabalho.

5.1 TRABALHOS FUTUROS

O aplicativo Nita Propaganda & Promoções tem a possibilidade de ser ainda muito explorado, de forma a facilitar e satisfazer ainda mais os seus usuários. Como trabalho futuro pretende-se adicionar a funcionalidade na aplicação web de cadastrar logomarca da empresa, para ser apresentada junto com o nome e slogan ao fazer a busca dessas.

Pretende-se também ao cadastrar uma promoção adicionar imagens de produtos e banners sobre as promoções de forma que quando o usuário for ver o detalhe das promoções, ele possa identificar melhor o produto.

Outro incremento ao aplicativo pode ser que ao cadastrar propagandas poderia cadastrar vídeos de curta duração que traria uma melhor explicação sobre a loja. Esses vídeos poderão ser os mesmos vídeos que são apresentados pelas empresas na televisão.

Além disso, também podem ser adicionadas rotas do ponto atual até o ponto que está localizado determinada empresa, facilitando a localização da empresa para os usuários que não conhecem a região.

Se incluído estas funcionalidades e incluso ao mercado de trabalho o aplicativo poderá ser muito bem aproveitado, uma vez que a quantidade de pessoas que possuem dispositivos móveis com plataforma Android é muito grande, e as maiorias dessas pessoas estão à procura de meios para facilitar suas atividades cotidianas. Além dos consumidores, os comerciantes também serão beneficiados com esse aplicativo, pois o Nita Propaganda & Promoções será um meio de propagação de marketing muito acessível e que pode expor aos consumidores o que a empresa tem a oferecer, gerando assim lucros.

6. REFERÊNCIAS

ANDROID DEVELOPER. **API Guides.** Android Developers. 2014. Disponível em: <http://developer.android.com/guide>. Acesso em: 10 mar. 2014.

APACHE. **Introdução ao Apache Wink,** 2009. Disponível em:<http://wink.apache.org/documentation/1.0/html/1%20Introduction%20to%20Apache%20Wink.html>. Acesso em: 03 mai. 2014.

BEZERRA, Luana. **Desenvolvendo Web Services RESTFUL utilizando a API JAX-RS 2.0 e Jersey,** 2013. Disponível em:< http://www.devmedia.com.br/desenvolvendo-web-services-restful-utilizando-a-api-jax-rs-2-0-e-jersey/27141>. Acesso em 10 set. 2014.

CAMPOS, J. **RESTMB: API RESTful para Android.** 68f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) - Centro Universitário Eurípides de Marília, 2013. Disponível em:< http://aberto.univem.edu.br/bitstream/handle/11077/988/Monografia%20Final.pdf?sequence=1>.Acesso em: 23 mai. 2014.

CANALLE, A.L. **Empregando tecnologia Java, Android e Geoprocessamento em aplicativos móveis.** 45f. Monografia de Especialização (Pós-Graduação em Tecnologia Java) - Universidade Tecnológica Federal do Paraná, 2011, Curitiba. Disponível em:< http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/493/1/CT_JAVA_VI_2010_03.PDF>. Acesso em: 23 mai. 2014.

GOMES, D. A**. Web Services SOAP em Java - Guia Prático para o Desenvolvimento de Web Services em Java.** 2ª ed. São Paulo: Novatec. 2010.

GOMES, F.J.L. et al. Desenvolvimento de Aplicações para Plataforma Google Android**.** In: SANTANA, A.M. et al**. Livro texto dos minicursos**. Teresina. p.100-123, 2011. Disponível em: <

http://www.die.ufpi.br/ercemapi2011/minicursos/EBOOK.pdf>. Acesso em: 01 mar. 2014.

HAMAD, H.; SAAD, M.; ABED, R. Performance Evaluation of RESTful Web Services for Mobile Devices. **International Arab Journal of e-Tecnology**, Palestina, Janeiro 2010.

HANSEN, M. **Basic SOA Using REST.** WebReference. 2007. Disponível em: <http://www.webreference.com/programming/basic_soa/index.html>. Acesso em: 01 mar. 2014.

ITU. **ITU releases 2014 ICT figures Mobile-broadband penetration approaching 32 per cent Three billion Internet users by end of this year.** ITU. 2014. Disponível em:<http://www.itu.int/net/pressoffice/press_releases/2014/23.aspx#.VEjM8vnF_Z1 >. Acesso em: 30 mai. 2014.

K19 TREINAMENTOS. **Criando um web service RESTful em Java**. 2010. Disponível em: <http://www.k19.com.br/artigos/criando-um-webservice-restful-em-java/>. Acesso em: 02 jun. 2014.

LECHETA, Ricardo R. **Google Android**. 2ª ed. São Paulo: Novatec, 2010.

LECHETA, Ricardo. R. **Google Android: aprenda a criar aplicações para dispositivos móveis com Android SDK.** 3ª. ed**.** São Paulo: Novatec Editora, 2013.

LEMOS A. **Comunicação e práticas sociais no espaço urbano: as características dos Dispositivos Híbridos Móveis de Conexão Multirredes (DHMCM).** 2007. Disponível em: <

http://www.facom.ufba.br/ciberpesquisa/andrelemos/DHMCM.pdf>. Acesso em: 03 mai. 2014.

LONGITUDESTORE. **The Haversine Fórmula**, 2008. Disponível em: < http://www.longitudestore.com/haversine-formula.html>. Acesso em 02 set. 2014.

MATTEUSSI, 2010. **Protótipo de Interface Web com Php para Gerenciamento de Banco de Dados Couchdb**. Universidade Comunitária da Região de Chapecó – Unochapecó. Área de Ciências Exatas e Ambientais. Curso de Ciência da Computação. Chapecó – SC, jul. 2010. 81p. Disponível em:< https://s3.amazonaws.com/elton/docs/monografia_kassiano.pdf >Acesso em: 02 mai. 2014.

NGOLO, 2009, citado por SILVA, A. e COEHO, J.H. de S. **Uma Arquitetura Semântica para a Interoperabilidade de Sistemas de E-Saúde.**140f. Trabalho de Conclusão de Curso (Tecnólogo em Banco de Dados) - Faculdade de Tecnologia de São José dos Campos, 2010.Disponível em:< http://fatecsjc.edu.br/trabalhos-de-graduacao/wp-content/uploads/2012/04/Uma-Arquitetura-Sem%C3%A2ntica-para-a-Interoperabilidade-de-Siste.pdf>. Acesso em: 12 jun. 2014.

OFICINADANET. **Como funciona o GPS?**. Oficina da Net. 2011. Disponível em:< http://www.oficinadanet.com.br/post/12406-como-funciona-o-gps>. Acesso em: 03 jun. 2014.

ORACLE. **The Java EE 6 Tutorial ; Part III Web Services; Chapter 20 Building RESTful Web Services with JAX-RS**. 2013. Disponível em:< http://docs.oracle.com/javaee/6/tutorial/doc/gilik.html>. Acesso em: 21 mai.2014.

PAMPLOMA, V.F. **Web Services. Construindo, disponibilizando e acessando Web Services via J2SE e J2ME**. JavaFree.org. 2010.

Disponível em: <http://javafree.uol.com.br/artigo/871485/Web-Services-Construindo-disponibilizando-e-acessando-Web-Services-via-J2SE-e-J2ME.html#ixzz2lzPiTE00>. Acesso em: 07 de out. 2013.

PEREIRA, Lúcio Camilo Oliva; SILVA, Michel Lourenço da Silva. **Android para Desenvolvedores.** 1. ed. Rio de Janeiro: Brasport Livros e Multimídia Ltda., 2009.

RUBBO, Fernando. **Construindo RESTful Web Service com JAX-RS 2.0,** 2013. Disponível em:< http://www.devmedia.com.br/construindo-restful-web-services-com-jax-rs-2-0/29468>. Acesso em: 21 mai. 2014.

SAUDATE, Alexandre. **REST: Construa API's inteligentes de maneira simples**. 1ªed. São Paulo: Casa do Código, 2013.

TOSIN, C.E.G. **Conhecendo o Android**, 2011. Disponível em:< http://www.softblue.com.br/blog/home/postid/11/CONHECENDO+O+ANDROID >. Acesso em: Acesso em: 23 mai. 2014.

TUTORIALSPOINT. **Android Tutorial,** 2013. Disponível em:<http://www.tutorialspoint.com/android/android_tutorial.pdf >. Acesso em: 01 jun. 2014.

VOGEL, Lars. **REST with Java (JAX-RS) using Jersey - Tutorial**, 2014. Disponível em: <http://www.vogella.com/tutorials/REST/article.html>. Acesso em: 21 Set. 2014.

W3SCHOOLS**. JSON Tutorial.** W3schools.com. Disponível em:<http://www.w3schools.com/json/> Acesso em: 01 jun. 2014.

7. APÊNDICES

7.1 Apêndice I

Este apêndice apresenta o *AndroidManifest* do aplicativo desenvolvido. Por meio deste é possível verificar as configurações do aplicativo e suas permissões.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.tccagualberto"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="13"
        android:targetSdkVersion="20" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".CCapa"
            android:label="@string/app_name"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.Holo.Light.NoActionBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".CPropagandas"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".CInformacoes"
            android:label="@string/informacoes"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".COpcao"
            android:label=""
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.Holo.Light.NoActionBar" >
        </activity>
        <activity
            android:name=".CListaESCategoria"
            android:label="@string/empresas"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".CListaEmpresasCategoria"
            android:label=""
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".CPropagandaSelecionada"
            android:screenOrientation="portrait" >
        </activity>
        <activity
```

```
            android:name=".CPromocoes"
            android:screenOrientation="portrait" >
        </activity>
        <activity
            android:name=".CPromocaoSelecionada"
            android:screenOrientation="portrait" >
        </activity>
    </application>

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE"
/>
    <uses-permission android:name="android.permission.READ_INTERNAL_STORAGE"
/>
    <uses-permission android:name="android.permission.WRITE_OWNER_DATA" />

</manifest>
```

7.2 Apêndice II

Este apêndice apresenta o método onPostExecute(), método da Classe AsyncTask, que recebe um *array* de promoções de determinada empresa e adiciona essas promoções em um ListView para que se o usuário se interessar o mesmo possa selecioná-la para ver todos os seus detalhes.

```
@Override
protected void onPostExecute(JSONObject json) {
if (json != null) {
       pDialog.dismiss();
       try {
       promocoes = json.getJSONArray(TAG_PROMOCAO);
              for (int i = 0; i < promocoes.length(); i++) {
                     JSONObject c = promocoes.getJSONObject(i);

                     tituloPromocao = c.getString(TAG_TITULO);
                     dataInicio = c.getString(TAG_DATA_INICIO);
                     dataFinal = c.getString(TAG_DATA_FIM);
                     descricao = c.getString(TAG_DESCRICAO);
                     dataCadastro = c.getString(TAG_DATA_CAD);


                     HashMap<String, String> map = new HashMap<String, String>();
                     map.put(TAG_TITULO, tituloPromocao);
                     map.put(TAG_DATA_INICIO, dataInicio);
                     map.put(TAG_DATA_FIM, dataFinal);
                     map.put(TAG_DESCRICAO, descricao);
                     map.put(TAG_DATA_CAD, dataCadastro);

                     oslist.add(map);
                     listView = (ListView) findViewById(R.id.list);
                     ListAdapter adapter = new SimpleAdapter(
                                   CPromocoes.this, oslist,
                                   R.layout.list_v_promocao, new String[] {
                                                 TAG_TITULO, TAG_DATA_INICIO,
                                                 TAG_DATA_FIM }, new int[] {
                                                 R.id.tituloPromocao,
R.id.datainicio,
                                                 R.id.datafim });
                     listView.setAdapter(adapter);
                     listView.setOnItemClickListener
                     (new AdapterView.OnItemClickListener() {
                            @Override
                            public void onItemClick(AdapterView<?> parent,
                                          View view, int position, long id) {

                                                 Intent intent = new
Intent(CPromocoes.this,
                                                 CPromocaoSelecionada.class);

                                                 tituloPromocao =
oslist.get(+position).get("titulo_promocao");
                                                 dataInicio =
oslist.get(+position).get("data_inicio");
                                                 dataFinal =
oslist.get(+position).get("data_final");
                                                 descricao =
oslist.get(+position).get("descricao");
                                                 dataCadastro =
oslist.get(+position).get("data_cadastro");
```

```
                                                System.out.println("Deus é
maiiis"+descricao+"dataa:: "+dataCadastro);

                                                intent.putExtra("nome_empresa",

       nomeEmpresaSelecionada);
                                                intent.putExtra("slogan_empresa",

       sloganEmpresSelecionada);
                                                intent.putExtra("titulo",
tituloPromocao);
                                                intent.putExtra("descricao",
descricao);
                                                intent.putExtra("data_cadastro",
dataCadastro);
                                                intent.putExtra("data_inicio",
dataInicio);
                                                intent.putExtra("data_fim",
dataFinal);
                                                startActivity(intent);
                                }
                        });
                }
                } catch (JSONException e) {
                        JSONObject c = null;
                        try {
                                c = json.getJSONObject(TAG_PROMOCAO);

                                tituloPromocao = c.getString(TAG_TITULO);
                                dataInicio = c.getString(TAG_DATA_INICIO);
                                dataFinal = c.getString(TAG_DATA_FIM);
                                descricao = c.getString(TAG_DESCRICAO);
                                dataCadastro = c.getString(TAG_DATA_CAD);
                        } catch (JSONException e1) {
                                e1.printStackTrace();
                        }

                        HashMap<String, String> map =
                                        new HashMap<String, String>();
                        map.put(TAG_TITULO, tituloPromocao);
                        map.put(TAG_DATA_INICIO, dataInicio);
                        map.put(TAG_DATA_FIM, dataFinal);

                        oslist.add(map);
                        listView = (ListView) findViewById(R.id.list);

                        System.out.println(titulo);
                        ListAdapter adapter = new SimpleAdapter(CPromocoes.this,
                                        oslist, R.layout.list_v_propagandas,
                                        new String[] { TAG_TITULO, TAG_DATA_INICIO,
                                                        TAG_DATA_FIM }, new int[] {
                                                        R.id.titulopropaganda,
R.id.datainicio,
                                                        R.id.datafim });
                        listView.setAdapter(adapter);
                        listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
                                @Override
                                public void onItemClick(AdapterView<?> parent,
                                                View view, int position, long id) {

                                        Intent intent = new Intent(CPromocoes.this,
                                        CPropagandaSelecionada.class);

                                        intent.putExtra("nome_empresa",
```

```
                                                nomeEmpresaSelecionada);
                                    intent.putExtra("slogan_empresa",
                                                sloganEmpresSelecionada);
                                    intent.putExtra("titulo", tituloPromocao);
                                    intent.putExtra("descricao", descricao);
                                    intent.putExtra("data_cadastro", dataCadastro);
                                    intent.putExtra("data_inicio", dataInicio);
                                    intent.putExtra("data_fim", dataFinal);
                                           startActivity(intent);
                                    }
                             });
                      }
               }else{
                      pDialog.cancel();
                      setContentView(R.layout.layout_sem_promocao);
               }
        }
          }
```

8. ANEXO

8.1 Anexo I

Este anexo apresenta um código fonte de uma classe de grande relevância para o projeto, a classe GPSTracker, responsável por ativar o GPS do aparelho caso esteja desligado, e realizar a busca das coordenadas geográficas latitude e longitude atual do dispositivo móvel , e este por sua vez, foi devidamente adaptada para este trabalho, pois a versão atual encontra-se em inglês.

```
public class GPSTracker extends Service implements LocationListener {
	private final Context mContext;
	boolean isGPSEnabled = false;
	boolean isNetworkEnabled = false;
	boolean canGetLocation = false;
	Location location;
	double latitude;
	double longitude;
	private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10;
	private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1;
	protected LocationManager locationManager;
	public GPSTracker(Context context) {
		this.mContext = context;
		getLocation();
	}
	public Location getLocation() {
		try {
			locationManager = (LocationManager) mContext
					.getSystemService(LOCATION_SERVICE);

			isGPSEnabled = locationManager

	.isProviderEnabled(LocationManager.GPS_PROVIDER);

			isNetworkEnabled = locationManager

	.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

			if (!isGPSEnabled && !isNetworkEnabled) {
			} else {
				this.canGetLocation = true;
				if (isNetworkEnabled) {
					locationManager.requestLocationUpdates(

	LocationManager.NETWORK_PROVIDER,
							MIN_TIME_BW_UPDATES,
							MIN_DISTANCE_CHANGE_FOR_UPDATES,
this);
					Log.d("Network", "Network");
					if (locationManager != null) {
						location = locationManager

	.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
						if (location != null) {
							latitude = 
location.getLatitude();
							longitude = 
location.getLongitude();
						}
					}
```

```
                        }
                        if (isGPSEnabled) {
                              if (location == null) {
                                    locationManager.requestLocationUpdates(

      LocationManager.GPS_PROVIDER,
                                                MIN_TIME_BW_UPDATES,

      MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
                                    Log.d("GPS Enabled", "GPS Enabled");
                                    if (locationManager != null) {
                                          location = locationManager

      .getLastKnownLocation(LocationManager.GPS_PROVIDER);
                                          if (location != null) {
                                                latitude                =
location.getLatitude();
                                                longitude               =
location.getLongitude();
                                          }
                                    }
                              }
                        }
                  }

            } catch (Exception e) {
                  e.printStackTrace();
            }
            return location;
      }
      public void stopUsingGPS(){
            if(locationManager != null){
                  locationManager.removeUpdates(GPSTracker.this);
            }
      }
      public double getLatitude(){
            if(location != null){
                  latitude = location.getLatitude();
            }
            return latitude;
      }
      public double getLongitude(){
            if(location != null){
                  longitude = location.getLongitude();
            }
            return longitude;
      }
      public boolean canGetLocation() {
            return this.canGetLocation;
      }
      public void showSettingsAlert(){
            AlertDialog.Builder          alertDialog          =          new
AlertDialog.Builder(mContext);

        alertDialog.setTitle("Configurações do GPS");

        alertDialog.setMessage("GPS não está ligado. Você deseja ir ao menu de
Configurações do GPS?");

        alertDialog.setPositiveButton("Configurar",                         new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,int which) {
             Intent                intent                =                new
Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
             mContext.startActivity(intent);
            }
        });
```

```
        alertDialog.setNegativeButton("Cancelar",                                 new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
            }
        });
        alertDialog.show();
      }
      @Override
      public void onLocationChanged(Location location) {
      }
      @Override
      public void onProviderDisabled(String provider) {
      }
      @Override
      public void onProviderEnabled(String provider) {
      }
      @Override
      public void onStatusChanged(String provider, int status, Bundle extras)
{
      }
      @Override
      public IBinder onBind(Intent arg0) {
            return null;
      }

}
```