UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
ESPECIALIZAÇÃO EM JAVA

VANDERLEI MATOS ANDRÉ

DESENVOLVIMENTO DE UM PROTÓTIPO DE APLICAÇÃO PARA UM DISPOSITIVO COM SISTEMA OPERACIONAL ANDROID PARA A GESTÃO DE UM EVENTO POR UM PRODUTOR

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA – PR
2012

VANDERLEI MATOS ANDRÉ

DESENVOLVIMENTO DE UM PROTÓTIPO DE APLICAÇÃO PARA UM DISPOSITIVO COM SISTEMA OPERACIONAL ANDROID PARA A GESTÃO DE UM EVENTO POR UM PRODUTOR

MONOGRAFIA DE ESPECIALIZAÇÃO

Monografia de especialização apresentada ao Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Especialista em Tecnologia Java”.

Orientador: Prof. Leandro Batista de Almeida.

Co-orientador: Prof. Nelson Kashima

CURITIBA – PR
2012

AGRADECIMENTOS

Agradeço, primeiramente, a Deus pela generosidade de colocar obstáculos em meu caminho que eu possa superá-los e me poupar dos insuperáveis.

À minha mãe, pela educação, pelos conselhos, pelo colo, pelas longas conversas de ensinamentos da vida e por conseguir criar, educar e formar seus três filhos. Graças a tudo que a senhora me ensinou, eu pude conquistar várias coisas boas em minha vida e, hoje, posso dar um pouco de conforto à minha família.

Ao meu pai que, mesmo longe, torce pelo meu sucesso.

Aos meus sócios e, principalmente, amigos Adenir Junior, Caio César e Lucas Dall'Agnol por me apoiarem nesta etapa tão importante da minha vida e por cobrir minha ausência na Goma IT. Pelos momentos divertidos que sempre passamos juntos trabalhando até de madrugada, pelo compartilhamento de conhecimento, crescimento profissional que vocês têm me proporcionado e pela honra de poder tê-los como amigos, meu muito obrigado.

Aos meus grandes amigos e mestres Fernando Camargo Alves, Marcelo Szostak e Pedro Rodolfo Kalva, a quem devo muitas das minhas conquistas e com quem aprendi muito nestes quase quinze anos de amizade e pela cobrança para eu terminar de uma vez esta especialização.

Ao meu amigo Polo Ottoni pelo prazer de sua amizade e por suas sábias palavras de conforto e motivação e à amiga Janaína por seu carinho e companheirismo e ao amigo e irmão Fabio Camargo Alves a quem tenho muito carinho e pelas cobranças de nos vermos mais.

Ao amigo Amarildo, sempre disposto a ajudar.

Aos meus tios e tias que sempre me ajudaram com conselhos e bons exemplos de conduta.

Às minhas irmãs Tuca, Juce e Preta que, mesmo de longe, me ajudaram com sua torcida pela conquista deste projeto e por seu amor incondicional.

Às minhas irmãs Marcia e Juceli que sempre estiveram comigo em todos os momentos e torceram por mim em todos os desafios que já enfrentei e por seu amor e compreensão.

Aos amigos Vladimir Torres e Wellington Correia à quem devo muito pelo meu sucesso na vida profissional e por sua amizade.

À minha avozinha, a quem sou muito grato pela educação que me deu, além do seu amor e colo nos momentos difíceis.

À minha filha Amanda que compreendeu minha ausência em alguns momentos de sua vida e que me apoiou neste momento.

Ao professor Leandro, por sua imensurável disponibilidade e ajuda nas dúvidas, mesmo as mais básicas, e por sua notável paixão em compartilhar conhecimento.

Ao professor Nelson Kashima, pela preocupação em nos ensinar, em pouco tempo, o máximo que pode sobre Android.

A todos os professores da especialização que, de alguma forma, me ajudaram na conquista de projeto.

Aos meus amigos Ronaldo e Elizangela Albuquerque, muito importantes na minha vida, que sempre torceram por mim e por compreenderem o motivo da minha ausência em vários eventos.

À Penelope, pelo simples fato de existir em minha vida, pelas brincadeiras, por escutar, calada, minhas reclamações e por seu amor incondicional e sem medida.

Ao amigo distante Marco Antônio Penteado que, através de sua indicação para eu conseguir um emprego, me proporcionou este destino maravilhoso e de sucesso.

Ao amigo Maurício Pagani com quem passei vários momentos bons em minha vida.

Agradeço, principalmente, à minha esposa, amiga e parceira de todos os momentos, Fabiana, a quem devo muito. Com toda certeza, sem o seu apoio, compreensão e carinho nas horas difíceis, esta conquista não teria se concretizado.

O prazer de compartilhar minha vida com você é imensurável e as alegrias que você me proporcionou nestes dezoito anos são incontáveis.

A você, minha eterna gratidão.

RESUMO

ANDRÉ, Vanderlei Matos. Desenvolvimento de um protótipo de aplicação para um dispositivo com sistema operacional Android para gestão de um evento por um produtor – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Curitiba, 2012.

A busca crescente de usuários por tecnologias que provê a mobilidade e a demanda sistemas que atendam esta necessidade motivou o estudo da plataforma Android.

Esta monografia apresenta um estudo da plataforma e as várias formas de implementação e integração com sistemas web através do uso de REST e JSON e, como resultado final, apresenta um protótipo de aplicação para gestão de eventos pelo produtor com o objetivo de gerenciar as vendas de ingresso do evento.

Palavras-chave: Android. Dispositivos Móveis. Mobilidade. Integração entre sistemas. RESTfull. JSON.

ABSTRACT

ANDRÉ, Vanderlei Matos. Development of a prototype application for a device with Android operating system for managing an event by the producer – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Curitiba, 2012.

The growing search for technologies that provides users mobility and demand systems that meet this need motivated the study of the Android platform.

This monograph presents a study of the platform and the various forms of implementation and integration with the web through the use of REST and JSON, and as a final result presents a prototype application for managing events by the producer in order to manage sales entry of the event.

Keywords: Android. Mobiles. Mobility. Integration between systems. RESTfull. JSON.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

GPS	Sistema de posicionamento global
OHA	Open HandSet Alliance
SDK	Kit de desenvolvimento padrão
UML	Linguagem de Modelagem Unificada
VM	Máquina Virtual
REST	Representational State Transfer
URI	Uniform Resource Identifier
CRUD	Create, Retrieve, Update, Delete
JSON	JavaScript Object Notation
HTTP	Hipertext Transfer Protocol
XML-RPC	Extensible Markup Language – Remote Procedure Call
MIME	Multipurpose Internet Mail Extensions
IP	Internet Protocol

SUMÁRIO

1. INTRODUÇÃO

A necessidade de estar o tempo todo *on-line* onde quer que se esteja, já é uma realidade atualmente. Quer seja para diversão, com jogos online, ou para interação em redes social e, cada vez maior, o uso de aplicações do mundo corporativo, onde se possa administrar o negócio sem estar, obrigatoriamente, à frente de um computador no escritório.

Esta tendência ressalta a afirmação de Vaz (2008) onde “a concorrência tem exigido cada vez mais esforços das empresas de quaisquer portes no sentido de manterem-se competitivas em seu ambiente de atuação”.

Para o uso corporativo, é necessário que as aplicações sejam mais “enxutas”, onde o conteúdo das informações e a sua visualização estejam voltados para a necessidade da mobilidade, ou seja, mostrar o que realmente interessa.

Lecheta (2010, p. 19) destaca o crescimento do mercado corporativo e a busca das empresas em integrar as aplicações móveis com seus sistemas de *backend.*

Com base nesta afirmação, o produtor de um evento deve poder tomar decisões sobre a estratégia de gestão de um evento onde quer que esteja.

O dinamismo das vendas de um evento, a intensa interação do produtor com um sistema de gestão e a necessidade de mobilidade soma-se às características necessárias para que se desenvolva uma aplicação em um dispositivo móvel e robusto que permita ao produtor, no menor tempo possível, executar ações relevantes ao andamento do evento.

A finalidade desta pesquisa é propor um sistema para aplicativos móveis baseados em *Android* onde o produtor de um evento possa interagir com o sistema de gestão mais facilmente, com objetivos voltados à gestão de um evento específico.

1.1 JUSTIFICATIVAS

A escolha de propor um protótipo de sistema para *smartphone* com o sistema operacional *Android* se deu pelo fato de que a busca de usuários por este tipo de tecnologia aumenta cada vez mais, dia após dia, e que a busca por um celular comum diminui em relação ao *smartphone* (IBOPE, 2012).

“14% da população brasileira já tem um *smartphone.* Na América Latina, essa percentagem só é superada na Argentina (24%) e no México (20%)” (TERRA, 2012).

Com o grande número de usuários de *smartphone*, a popularização do sistema operacional *Android* só aumentou e já é o mais utilizado no Brasil (TERRA, 2012).

Ainda sobre o aumento expressivo da busca por dispositivos móveis como o *smartphone:*

> “O mercado de celulares está crescendo cada vez mais. Estudos mostram que hoje em dia mais de 3 bilhões de pessoas possuem um aparelho celular, e isso corresponde a mais ou menos metade da população mundial.
>
> Hoje em dia, os usuários comuns estão procurando cada vez mais celulares com diversos recursos como câmeras, músicas, Bluetooth, ótima interface visual, jogos, GPS, acesso a internet e e-mails, e agora ainda temos a TV digital.
>
> O mercado corporativo também está crescendo muito, e diversas empresas estão buscando incorporar aplicações móveis a seu dia-a-dia para agilizar seus negócios e integrar as aplicações móveis com seus sistemas de *back-end*. Empresas obviamente visam lucro e mais lucro, e os celulares e smartphones podem ocupar um importante espaço em um mundo onde a palavra mobilidade está cada vez mais conhecida” (Lecheta, 2010, p. 19).

A plataforma Android evolui muito rapidamente, na mesma velocidade que novos dispositivos são ativados com esta tecnologia, permitindo que as aplicações desenvolvidas tenham uma interface gráfica cada vez mais poderosa, além de proporcionar a integração com outros aplicativos.

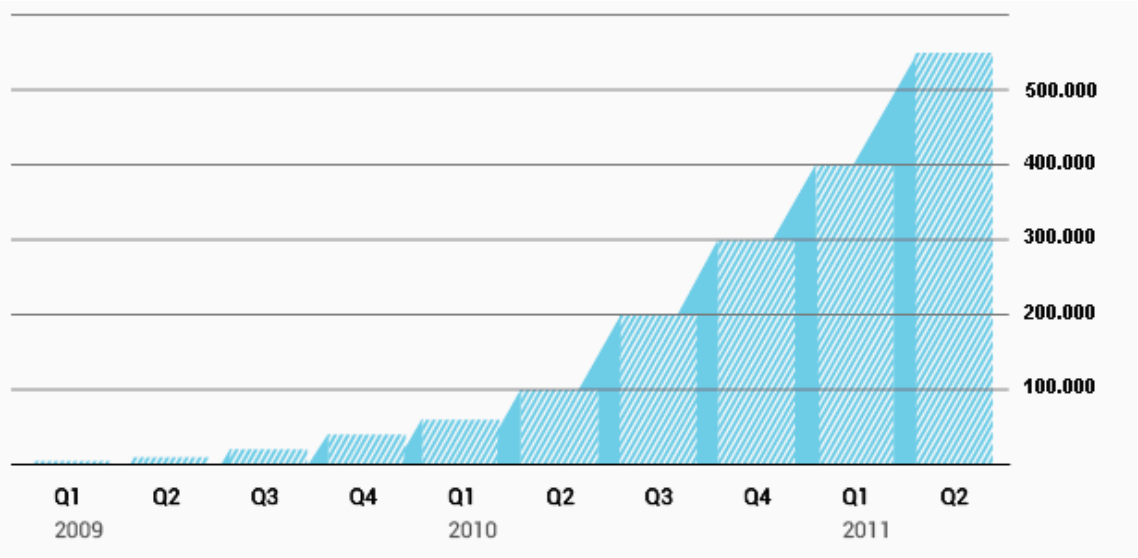

Figura 1 - Crescimento de ativações da plataforma Android por dia no mundo.
Fonte: ANDROID DEVELOPER.

No âmbito científico, a escolha de um projeto cuja tecnologia seja o *smartphone* com *Android* faz com que surjam novos desafios a serem superados e que este estudo sirva para futuros projetos com base nesta tecnologia.

Por outro lado, no âmbito social, a tecnologia cada vez mais robusta, completa e de fácil acesso aos mais diversos usuários traz para o presente o benefício de ter às mãos um dispositivo discreto, mas com a seriedade de poder tomar decisões do mundo corporativo onde quer que se esteja.

A existência de um sistema de *backend,* desenvolvido para a web, para a gestão do evento por parte do produtor faz com que surja a necessidade de tomadas de decisões aliadas à mobilidade e este estudo serve para suprir esta lacuna.

1.2 OBJETIVO

Este trabalho visa levantar as necessidades básicas do produtor para administrar seu evento através de um dispositivo móvel com *Android*.

1.2.1 Objetivo Geral

O projeto deste estudo tem como meta principal projetar e desenvolver um protótipo de aplicação que interaja com um sistema de *backend* e possibilite ao produtor de evento a gestão básica para a tomada de decisões utilizando, para isso, um dispositivo móvel com o sistema operacional *Android*.

Como resultado do estudo, pretende-se ter uma versão estável do protótipo do sistema rodando no dispositivo móvel, além da documentação do sistema baseada em diagramas da Linguagem de Modelagem Unificada (UML): Use Cases, Diagrama de Classes, Diagrama de Seqüência, Diagrama de Atividades e o código fonte do sistema.

1.2.2 Objetivo Específico

Desenvolver os módulos mínimos necessários para que o produtor possa consultar e administrar um evento através de um dispositivo móvel com sistema operacional *Android*.

São elas:

- Autenticação: Validação do usuário pelo canal *mobile* no sistema de *backend;*
- Consulta de Eventos: Listagem dos eventos ativos da produtora a qual pertence o produtor;
- Consulta de ingressos: Listagem dos ingressos criados para o evento selecionado;
- Agendar lote: Ação na qual o produtor pode agendar a virada de um novo lote de ingressos;
- Virada de lote: Ação na qual o produtor cria um novo lote de ingressos e determina a virada imediata deste lote;

1.3 CONTEÚDO DO TRABALHO

O conteúdo deste trabalho está organizado da seguinte forma:

No capítulo 2, é apresentado um estudo da plataforma Android, onde é possível verificar os principais componentes e as formas como são utilizados em uma aplicação para dispositivo móvel.

Também é apresentado o estudo das tecnologias utilizadas para integração entre sistemas na web.

No capítulo 3, são apresentados os diagramas de caso de uso, de classes e de sequência desenvolvidos para a implementação do protótipo, além das telas do protótipo sendo executado no emulador.

No capítulo 4, estão as considerações finais sobre o estudo desta monografia.

2. REVISÃO BIBLIOGRÁFICA

2.1 GESTÃO DE EVENTO

A gestão de um evento se dá pela união de várias atividades que envolvem pessoas ligadas à administração do evento, que vão desde o produtor até a empresa responsável pela venda do ingresso.

Um sistema de gestão de evento é responsável por suportar todas estas atividades, que inclui o suporte às atuações, em tempo real, sobre um evento em andamento.

As atividades envolvidas na administração do evento, no que diz respeito à venda de ingressos, são complexas e dinâmicas e requer um controle criterioso das ações para garantir que a venda transcorra da melhor forma possível, evitando transtornos, tanto para o cliente que compra o ingresso, quanto para o produtor do evento.

No que diz respeito à venda de um evento, existem vários tipos de ingressos e, para cada ingresso, são criados lotes, onde cada lote representa um valor diferente de venda para o mesmo ingresso e que deve ser controlado pelo sistema de gestão a liberação de cada um dos lotes.

De acordo com a estratégia de venda do evento, à qualquer momento o produtor ou a empresa responsável pela gestão podem alterar a quatidade de ingressos de um determinado lote ou, ainda, criar um novo lote de venda.

A **Figura 2** demontra o fluxo macro das atividades envolvidas em um evento, no que se refere à administração da venda de ingresso.

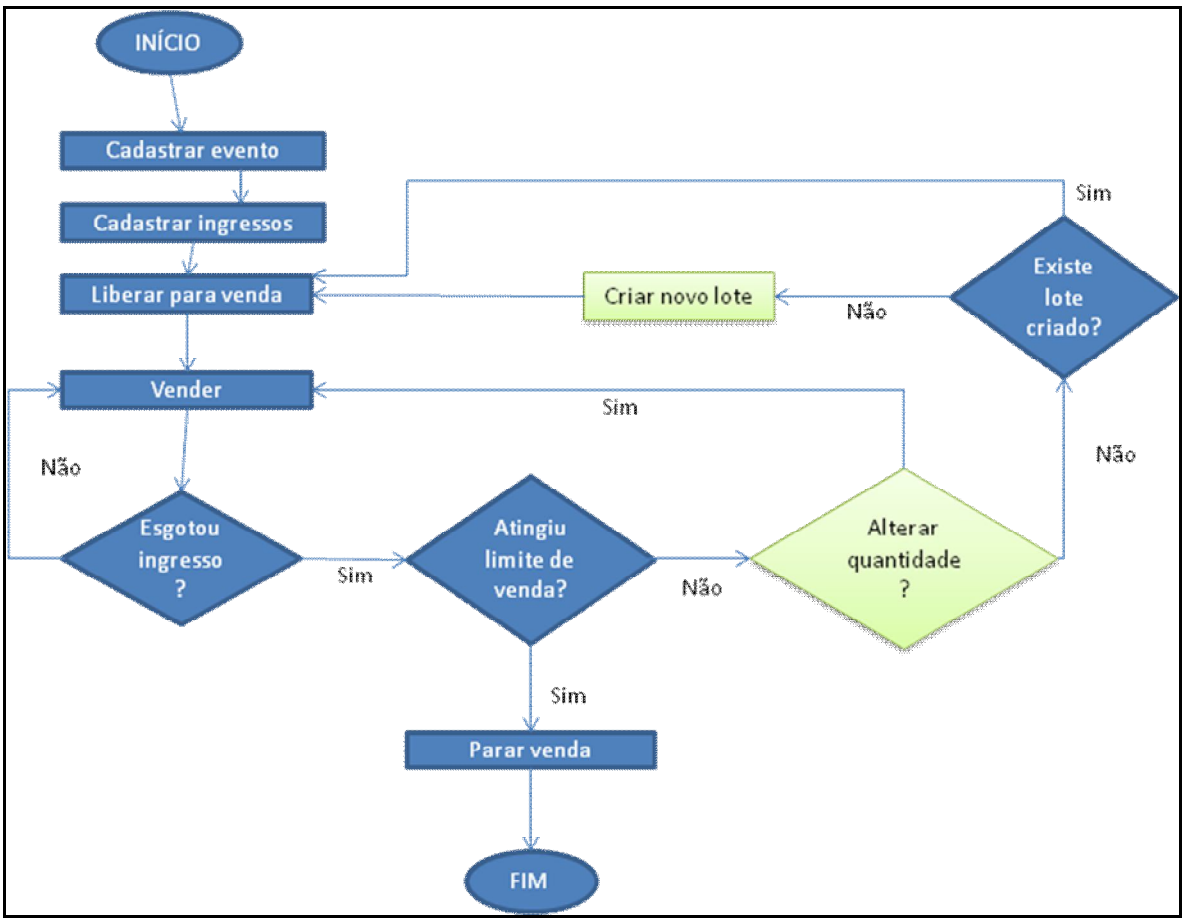

Figura 2 - Fluxograma de Vendas de Ingresso
Fonte: Autoria própria.

Vaz (2008) destaca a importância da informação como diferencial competitivo de empresas para a tomada de decisão em resposta às necessidades de clientes, ressaltando que a tecnologia da informação é uma ferramenta importante neste contexto, capaz de transformar dados em informações valiosas.

Baseando-se nesta afirmação, é indiscutível o ganho que a mobilidade pode proporcionar ao sistema de gestão de eventos.

2.2 TECNOLOGIA ANDROID

O Android nasceu em novembro de 2007, quando foi anunciado pelo Google a *Open Handset Alliance* (OHA) e, ao mesmo tempo, teve a primeira versão *beta* do Kit de Desenvolvimento Padrão (SDK) liberada para *downloads*. “Poucos meses depois, mais de 1 milhão de pessoas já tinham feito download dos *kits* no site do Google” (ROGERS, 2009, p. 3).

A criação do Android atende aos anseios de várias categorias de usuários:

Os usuários comuns, que ansiavam por um dispositivo móvel mais moderno, com vários recursos e com navegação amigável e das empresas, que buscavam por um dispositivo que pudesse oferecer os requisitos necessários para o desenvolvimento de aplicações corporativas (LECHETA, 2010, p. 20);

O número de usuários de telefone móvel é quase o dobro quando comparado ao número de aparelhos de televisão em uso no mundo, chegando a quase 3 bilhões e as grandes empresas do ramo de tecnologia e telefonia participantes da OHA compartilham a idéia de que pode melhorar a vida desses usuários e empresas oferecendo-lhes uma tecnologia inovadora e completa, a plataforma Android (OHA OVERVIEW, 2012).

2.3 PLATAFORMA ANDROID

Para Lecheta (2010, p. 19), o Android é “a nova plataforma de desenvolvimento para aplicativos móveis como *smartphones* e contêm um sistema operacional baseado em Linux”, onde seus componentes são escritos em C ou C++ e as aplicações dos usuários e até mesmo as aplicações internas são escritas em Java (ABLESON, 2012, p. 4).

“Não é uma plataforma de hardware” (ABLESON, 2012, p. 4).

O Android não distingue aplicações internas das aplicações escritas pelo SDK, o que permite desenvolver aplicações robustas utilizando os recursos disponíveis pelo dispositivo (ABLESON, 2012, p. 4).

Além disso, é possível integrar a aplicação desenvolvida com vários outros aplicativos disponibilizados pelo Google, como o serviço de mapas, por exemplo.

Diferente de outros ambientes de desenvolvimento para dispositivos móveis como Windows Mobile e Apple iPhone, a licença do Android é *open-source,* o que ajuda na maturidade da plataforma, uma vez que qualquer desenvolvedor da comunidade pode contribuir identificando problemas e propondo soluções.

Ableson (2012, p. 4) ilustra a relação entre o Android e o *hardware* no qual ele é executado.

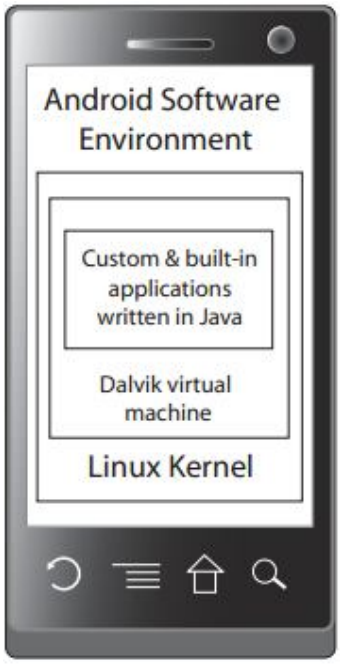

Figura 3 - Plataforma Android
Fonte: ABLESON (2012, p. 4).

2.4 COMPOENTES DE UMA APLICAÇÃO ANDROID

Os principais componentes da arquitetura Android são descritos a seguir.

Para se construir uma aplicação é preciso ter, pelo menos, um destes quatro tipos de componentes, podendo, ainda, existir todos em uma única aplicação:

2.4.1. Activity

A classe *Activity,* que se encontra no pacote `android.app`, é uma das principais classes da arquitetura Android e representa o meio de interação da aplicação com o usuário.

Uma aplicação pode ter uma ou mais *Activity* e "cada *Activity* é responsável por controlar os eventos da tela e definir qual *View* será responsável por desenhar a interface gráfica com o usuário" (LECHETA, 2010, p. 93).

Basicamente, quanto mais complexa a aplicação, ou seja, quanto mais tela tiver, mais *Activity* terá.

Segundo Lecheta (2010, p. 94), "uma *activity* representa uma atividade, ação ou funcionalidade que o usuário pode realizar dentro de sua aplicação".

Para utilizar a classe *Activity* é preciso estendê-la e implementar o método `onCreate,` que é chamado pelo ambiente de execução do Android no momento de criação da *Activity* e é onde o programador define qual tela será mostrada para o usuário.

A **Figura 4** ilustra o esqueleto básico para a implementação de uma *Activity.*

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

Figura 4 - Implementado uma Activity
Fonte: Autoria própria.

Para que a classe responsável por mostrar a tela ao usuário execute com sucesso, não basta somente implementar os métodos que a classe *Activity* exige. É necessário que toda e qualquer classe que estenda de *Activity* esteja registrada no arquivo de configuração do Android, o `AndroidManifest.xml.`

Neste arquivo, são definidas configurações como as permissões que o aplicativo terá, além de versão mínima necessária para executar e também é declarada a classe principal do aplicativo.

A **Figura 5** mostra o arquivo `AndroidManifest.xml` com algumas configurações típicas.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="br.com.goma.android.produtor"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="13" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".ProdutorActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
    <uses-sdk android:minSdkVersion="7" />

    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
</manifest>
```

Figura 5 - Arquivo AndroidManifest.xml
Fonte: Autoria própria.

A *Activity* possui um ciclo de vida controlado pelo sistema operacional onde, para cada mudança de estado, existe um método que o programador pode sobrescrever para executar alguma ação referente à aplicação em execução.

Método	Descrição	Pode matar o processo?	Próximo
`onCreate()`	Este método 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.	Não	`onStart()`
`onRestart()`	O método `onRestart()` é chamado quando uma *activity* foi parada temporariamente e está sendo iniciada outra vez.	Não	`onStart()`

`onStart()`	É chamado quando a *activity* está ficando visível ao usuário e já tem uma view.	Não	`onResume()` ou `onStop()`
`onResume()`	O método `onResume()` é chamado quando a *activity* está no topo da pilha "*activity stack*" e, dessa forma, já está executando como a *activity* principal e pronta para interagir com o usuário.	Não	`onPause()`
`onPause()`	Se algum evento ocorrer, como o celular entrar em modo de espera/dormir para economizar energia ou uma `Intent` estar sendo processada, a *activity* do topo da pilha que está executando pode ser temporariamente interrompida. Para isso, o método `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()`.	**Sim**	`onResume()` ou `onStop()`
`onStop()`	Chamado quando a *activity* está sendo encerrada e não está mais visível ao usuário, o que pode ocorrer quando outra *activity* é iniciada, por exemplo. Caso isto ocorra, o método `onRestart()`	**Sim**	`onRestart()` ou `onDestroy()`

	é chamado para reiniciar a *activity* Caso a *activity* fique muito tempo parada em segundo plano e o sistema operacional do Android precise limpar os recursos para liberar memória, o método `onDestroy()` pode ser automaticamente chamado para remover completamente a *activity* da pilha.		
`onDestroy()`	Este método literalmente encerra a execução de uma *activity* e 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 seu processo no sistema operacional também é completamente encerrado.	**Sim**	-

Quadro 1- Um resumo dos métodos do ciclo de vida da Activity
Fonte: Adaptado de Lecheta (2010, p. 98).

A coluna “Pode matar o processo?” indica em que momento o sistema operacional pode encerrar o processo, ou seja, somente depois da execução destes métodos é que o aplicativo será encerrado efetivamente.

No clico de vida da *Activity,* o método `onPause()` é o único que, seguramente, é chamado antes que a aplicação seja encerrada, portanto, neste

ponto, como mostra o **Quadro 1**, é importante implementar algum código que salve o estado da aplicação, porém, deve-se tomar cuidado com a informação que será retida durante a execução deste método porque o procedimento pode ser lento e a próxima *activity* somente será mostrada quando este processo estiver encerrado, o que pode prejudicar a velocidade de navegação do usuário (ANDROID ACTIVITIES, 2012).

A **Figura 6** ilusta o fluxo do ciclo de vida de uma *activity.*

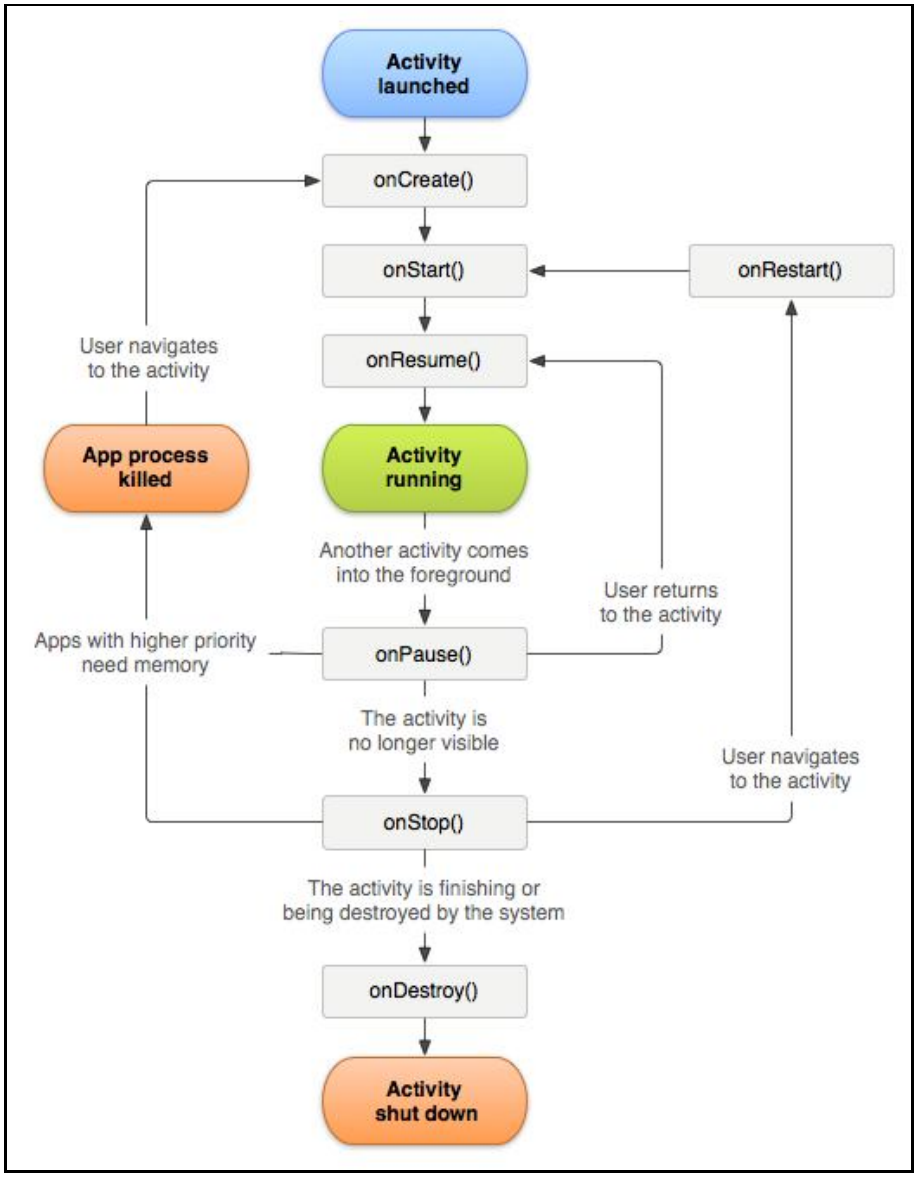

Figura 6 - Ciclo de vida de uma Activity
Fonte: ANDROID ACTIVITIES.

2.4.2. Service

"A classe *Service* é utilizada para executar um serviço em segundo plano, geralmente vinculado a algum processo que deve executar por tempo indeterminado e tem um alto consumo de recursos, memória e CPU" (LECHETA, 2010, p. 317).

Por se tratar de um processamento sendo executado em segundo plano, este tipo de componente não requer uma interface gráfica e é executado a partir de outro componente, como uma *Activity,* por exemplo.

Um serviço pode ser executado de duas formas:

Método	Descrição
`startService(intent)`	Método utilizado para iniciar um serviço que fica executando por tempo indeterminado, até que o método `stopService(intent)` seja chamado ou até que o próprio serviço termine sua própria execução chamando o método `stopSelf()`.
`bindService(intent, com, flags)`	Este método pode iniciar um serviço se ele ainda não está executando ou simplesmente conectar-se a ele. Ao estabelecer a conexão é possível recuperar uma referencia de uma classe ou interface que pode manipular o serviço. Por exemplo, um player mp3 executando em segundo plano poderia expor uma interface com os métodos para controlar a reprodução das músicas.

Quadro 2- Métodos que podem ser usados para iniciar o serviço
Fonte: Lecheta (2010, p. 319).

A classe *Service* possui um ciclo de vida parecido com uma *Activity* e, assim como tal, é conhecida e controlada pelo *Android,* que decide quando um processo deve ser encerrado para liberar memória e recursos.

Alguns métodos essenciais no ciclo de vida de um serviço necessitam ser sobrescritos para que se possa controlar o serviço em um determinado ponto do seu ciclo de vida (ANDROID SERVICES, 2012).

- **`onStartComand()`** – Este método é chamado pela VM quando um outro componente executa o método **`startService()`** da classe *Context.* Uma vez executado este método, o serviço é iniciado e pode executar em segundo plano indefinidamente, até que seja executado o método **`stopSelf()`**.
- **`onBind()`** – Este método é executado quando um outro componente aciona o método `bindService()` da classe *Service.* Sua implementação é obrigatória e retorna a interface `Ibind,` que deve ser implementada pelo cliente para a comunicação com o serviço. Nos casos em que o serviço não permite interação por parte do cliente, a implementação deste método retorna `null.`
- **`onCreate()`** – Este método é chamado pelo sistema operacional do *Android* uma única vez, quando o serviço é criado, antes de chamar um dos métodos `onStartComand()` ou `onBind(),` dependendo do modo como o serviço está sendo criado.
- **`onDestroy()`** – Este é o último método a ser executado antes que o serviço seja completamente removido da pilha de execução. O responsável por sua chamada pode ser um dos seguintes métodos: `stopSelf()`, da própria classe *Service* ou `stopService()`, da classe *Context*, se o serviço foi criado através do `startService()` e `onUnbind()`, se o serviço foi criado através do `bindService()`. O método `onUnbind()` somente será executado quando todos os clientes tiverem se desconectado do serviço, ou seja, o último cliente tiver chamado o método `unbindService().`

Assim como uma *Activity* e outros componentes do *Android,* o serviço também deve ser declarado no `AndroidManifest.xml` e qualquer outro componente do *Android* pode utilizá-lo, até mesmo componentes de outras

aplicações, entretanto, é possível declará-lo como privado, bloqueando o acesso de outras aplicações (ANDROID SERVICES, 2012).

A **Figura 7** mostra alguns detalhes importantes de configuração do serviço:

```
<service android:enabled=["true" | "false"]
         android:exported=["true" | "false"]
         android:icon="drawable resource"
         android:label="string resource"
         android:name="string"
         android:permission="string"
         android:process="string" >
    . . .
</service>
```

Figura 7 - Configuração do serviço no AndroidManifest.xml
Fonte: ANDROID SERVICES.

- *enabled* - Define se o serviço pode ou não ser instanciado pelo sistema. O valor padrão é *true*.
- *exported* – Define se componentes de outras aplicação podem usar o serviço. Caso o valor seja *false*, somente componentes da mesma aplicação ou aplicações com o mesmo *ID user* poderão utilizar o serviço.
- *icon* - Referência ao nome do recurso contido no arquivo de configuração que define uma imagem como ícone do serviço.
- *label* – Nome do serviço que será mostrado ao usuário.
- *name* – Nome da classe que implementa o serviço, propriamente dito.
- *permission* – Tipo de permissão que a entidade deve ter para executar o serviço.
- *process* – Nome do processo onde o serviço será executado.

O serviço não cria sua própria *thread* e também não executa em um processo separado, pelo contrário, é executado na mesma *thread* que o processo principal, a menos que seja especificado. Isto significa que, se o serviço a ser executado consume muito processamento ou bloqueia algum tipo de recurso como um tocador de MP3 ou rede, é aconselhável criar uma *thread* dentro do serviço para executar tal

tarefa, evitando, assim, erros de aplicações que travam, deixando o processamento principal somente para interações com o usuário (ANDROID SERVICES, 2012).

2.4.3 BroadcastReceivers

Lecheta (2010, p. 290) descreve a classe `BroadcastReceiver` como “uma das classes mais importantes da arquitetura do Android, utilizada para que aplicações possam reagir a determinados eventos gerados por uma *intent*”.

“Se um aplicativo quer receber e responder a um evento global, como um toque de telefone ou uma mensagem de texto recebida, deve registar-se como `BroadcastReceiver`" (ABLESON, 2012, p. 19).

Um *BroadcastReceiver* é um “escutador” de eventos disparados pelo sistema operacional e, para que a aplicação possa responder a estes eventos, deve registrá-los no `AndroidManifest.xml`.

Desta forma, não é necessário que a aplicação esteja executando para que possa interceptar o evento, uma vez que o sistema operacional se encarrega de ativá-la.

Assim como o serviço, um *BroadcastReceiver* não possui interface e é executado em segundo plano, sendo recomendado para executar rotinas rápidas e simples. Quando um *BroadcastReceiver* necessita executar algum processamento mais pesado como a persistência de dados ou um outro processo mais demorado, é aconselhável que este inicie um serviço para executar tal rotina, uma vez que o serviço é desenhado para este propósito e, por outro lado, o *BroadcastReceiver* pode estar escutando vários eventos (ABLESON, 2012, p. 20).

2.4.4. Content Providers

Compartilhar arquivos e banco de dados não é uma boa prática no Android e isto é reforçado pela política de segurança de acessos imposta pelo sistema

operacional linux, que impede acessos a arquivos de uma aplicação por outra sem permissão concedida explicitamente (ABLESON, 2012, p. 22).

A única maneira de compartilhar dados entre aplicações é através dos provedores de conteúdos, onde, se uma aplicação precisa acessar dados de outra aplicação, ela o fará acessando o provedor de conteúdo da outra aplicação, como, por exemplo, quando uma aplicação acessa a agenda de contatos do Android.

O provedor de conteúdo é uma camada de dados que provê a abstração necessária para a persistência e requisição dos dados, centralizando estas tarefas em um único lugar, podendo utilizar uma das várias formas de persistência de dados disponíveis no Android, que incluem arquivos, banco de dados SQLite e armazenamento em memória (ABLESON, 2012, p. 22).

Para que um provedor de conteúdo seja visível por outras aplicações do Android, é necessário registrá-lo no `AndroidManifest.xml` com a tag `<provider>`.

```
<provider android:name="com.example.autos.AutoInfoProvider"
          android:authorities="com.example.autos.autoinfoprovider"
          . . . />
</provider>
```

Figura 8 - Configuração do provedor no AndroidManifest.xml
Fonte: ANDROID PROVIDERS.

- `android:name` define o nome da classe que implementa o `ContentProvider`, incluindo o pacote o qual pertence a classe.
- `android:authorities` define o nome simbólico que identifica o provedor dentro do sistema.

Contudo, a criação de provedor de conteúdo só é viável quando se deseja tornar algum tipo de informação pública à outras aplicações. Do contrário, se utiliza o banco de dados, onde a informação é privada ao escopo da aplicação.

2.5. BANCO DE DADOS

O Android possui suporte ao banco de dados relacional SQLite, que não é tão robusto como os grandes banco de dados, mas que possui recursos suficientes para persistir dados de uma aplicação.

Este pequeno banco de dados foi implementado como uma biblioteca da linguagem C e incluido como parte da pilha de softwares do Android.

Por ser implementado como uma biblioteca, ele executa no mesmo processo da aplicação que o criou e isso diminui problemas como dependências externas, transações bloqueadas e sincronismo (MEIER, 2010, p. 210).

De acordo com Lecheta (2010, p. 368), “um banco de dados é visível somente à aplicação que o criou e cada aplicação pode criar um ou mais banco de dados, que ficam localizados na seguinte pasta, relativa ao nome do pacote do projeto”:

```
/data/data/nome_pacote/databases/
```

Um banco de dados pode ser criado das seguintes formas:

- Utilizando a API do Android para o SQLite.
- Usando um cliente do SQLite.
- Utilizando o aplicativo SQLite3 pelo console do emulador.

A documentação do Android recomenda a criação de um novo banco de dados SQLite através da extensão da classe `SQLiteOpenHelper`, onde o método `onCreate()` deve ser sobrescrito para incluir o comando de criação de uma tabela no banco de dados (ANDROID SQLITE, 2012).

A partir da execução do método `onCreate()`, é possível obter uma instância de `SQLiteDatabase` através dos métodos `getWritableDatabase()` e `getReadableDatabase()` onde é possível executar os comandos de manipulação de dados do banco de dados.

2.6. AMBIENTE DE EXECUÇÃO

As aplicações desenvolvidas para o Android através do SDK são todas escritas em Java e, por isso, necessitam de uma máquina virtual (VM) para interpretar e executar o código.

O sistema operacional utilizado pelo Android, o Linux, provê a interface entre as aplicações escritas em Java e o *hardware* do dispositivo móvel através de uma VM customizada, chamada Dalvik, desenhada de forma que garante o funcionamento eficiente de várias instâncias em um único dispositivo (MEIER, 2010, p. 12).

Lecheta (2010, p. 24) descreve:

> "Ao desenvolver as aplicações para o Android, você vai utilizar a linguagem Java e todos os seus recursos normalmente, mas, depois que o bytecode (.class) é compilado, ele é convertido para o formato .dex (Dalvik Executable), que representa a aplicação do Android compilada.
>
> Depois disso, os arquivos .dex e outros recursos como imagens são compactados em um único arquivo com a extensão .apk (Android Package File), que representa a aplicação final, pronta para ser distribuída e instalada."

Segundo Meier (2010, p. 15), a VM Dalvik age como uma camada intermediária e utiliza os recursos do *kernel* do Linux para lidar com funcionalidades de baixo nível, como a segurança, processos, *thread* e gerenciamento de memória. Estas tarefas fortemente ligadas ao *hardware* são tiradas do desenvolvedor, que passa a se preocupar somente com as regras ligadas ao desenvolvimento do *software*.

Mesmo com a evolução contínua de dispositivos móveis onde a capacidade de processamento e a memória disponível são cada vez maiores, quando se desenvolve uma aplicação para o Android, deve-se levar em conta que um dispositivo móvel não é um computador *desktop* e as atividades que exigem muito processamento e/ou uso excessivo de memória consomem bateria do aparelho.

2.7. RESTFUL

A disponibilidade de serviços é uma realidade quando se pensa em integração entre sistemas na web, o que faz com que a utilização de *web services* seja comum, uma vez que a aplicação cliente e a aplicação servidora não precisam estar escritas na mesma linguagem.

Um padrão para compartilhamento de informações nos serviços web que está crescendo é o REST (Representational State Transfer), que utiliza o conceito de recursos identificados por um *id* e que utiliza os métodos do protocolo HTTP para realizar operações sobre esses recursos em um servidor web (DEBUG, 2012).

O termo REST, que significa ***REpresentational State Transfer*** (Transferência de Estado Representativo, em tradução livre) foi criado por Roy Fielding em sua dissertação publicada em 2000, que define o REST, não como uma arquitetura, mas, sim, como um conjunto de restrições que, quando aplicado ao desenho de um sistema, cria um estilo de arquitetura de software (SANDOVAL, 2009, p. 7).

O URI (Uniform Resource Identifier), em um *web service* RESTful, é um link para um recurso e é o único meio de aplicações clientes e servidor trocarem representações (SANDOVAL, 2009, p. 10).

O REST utiliza o conceito de CRUD (Create, Retrieve, Update, Delete) para disponibilizar a execução de operações para aplicações cliente e, para isso, utiliza os quatro métodos do protocolo HTTP (SANDOVAL, 2009, p. 11).

São eles:

- GET – Utilizado para recuperar recursos.
- POST – Utilizado para criar recursos.
- PUT – Utilizado para atualizar recursos.
- DELETE – Utilizado para excluir recursos.

A requisição feita por uma aplicação cliente ao *web service* é mapeada por um método específico em uma classe recurso do Jersey, que é uma classe Java dentro de um contexto em uma aplicação web. “Todo recurso Jersey possui um URI apontando para ele” (SANDOVAL, 2009, p. 83).

A classe recurso contém é anotada como @Path, que define a URI do contexto.

A utilização do REST se torna bastante simples com a implementação do framework Jersey, aliado ao uso de *annotations.*

Os métodos da classe recurso, responsáveis por executar uma determinada operação, devem estar anotados com algumas das anotações abaixo:

- @GET - Define o método que irá tratar uma requisição HTTP GET.
- @PUT – Define o método que irá tratar uma requisição HTTP PUT.
- @Path - Define um caminho relativo para o método dentro da raiz de um recurso Jersey.
- @Consumes – Definida junto com @POST e @PUT, diz ao *framework* para qual método delegar a requisição de entrada, ou seja, qual o tipo MIME (Multipurpose Internet Mail Extensions) será aceito.
- @Produces – Definida junto com @GET, @POST e @PUT, diz ao *framework* o tipo de representação (MIME) que será enviado ao cliente.

As figuras 9 e 10 demonstram exemplos de anotações utilizadas no REST.

```
@GET
@Path("/getAllEvents")
@Produces(MediaType.APPLICATION_JSON + ";charset=UTF-8")
public TicketResponse getAllEvents() {

    TicketResponse resp = new TicketResponse();
    List<EventSite> events = eventDAO.getAllEvents();

    if (events == null || events.size() == 0) {
        resp.setStatus(new Status(11, "Nenhum eventos disponivel para venda."));
    } else {
        resp.setStatus(new Status(01, "Ok"));
        resp.setEvents(events);

    }

    return resp;
}
```

Figura 9 - Exemplo do recurso GET do REST
Fonte: Autoria própria.

```
@PUT
@Path("/validateTicketStock")
@Produces(MediaType.APPLICATION_JSON + ";charset=UTF-8")
@Consumes(MediaType.APPLICATION_JSON)
public TicketResponse validateTicketStock(TicketRequest request) {

    TicketResponse resp = new TicketResponse();
    List<TicketSite> tickets = new ArrayList<TicketSite>();

    if (request != null && request.getTickets() != null) {
        tickets = ticketDAO.validateTicketStock(request.getTickets());
    }

    if (tickets == null || tickets.isEmpty()) {
        resp.setStatus(new Status(41, "Erro na validacao."));
    } else {
        resp.setTickets(tickets);
        resp.setStatus(new Status(01, "ok"));
    }

    return resp;
}
```

Figura 10 - Exemplo do recurso PUT do REST
Fonte: Autoria própria.

2.8. JSON

O JSON (JavaScript Object Notation) é um padrão para chamadas de procedimentos remotos (JSON-RPC) bastante utilizado como uma alternativa ao XML-RPC (Extensible Markup Language – Remote Procedure Call) para trafegar dados complexos pela internet por ser fácil de usar e que possui suporte pela maioria dos navegadores disponíveis no mercado, o qual tem a tarefa de transformá-lo em objetos nativos do JavaScript (JSON, 2012).

Com ele, é possível enviar e receber, via protocolo HTTP (Hipertext Transfer Protocol), dados, invés de *String* passadas por parâmetro.

Sua estrutura é baseada em par chave/valor, o que simplifica a geração e torna fácil o entendimento.

“Um objeto JSON é um conjunto não ordenado de pares chave/valor onde o objeto começa com “{“ (chave esquerda) e termina com “}” (chave direita). Cada

nome de atributo é seguido de “:” (dois ponto) e o par chave/valor é separado por “,” (vírgula)” (JSON ORG, 2012).

A **Figura 11** exemplifica a estrura básica de um objeto JSON, que pode conter vários atributos, sempre separados por vírgula.

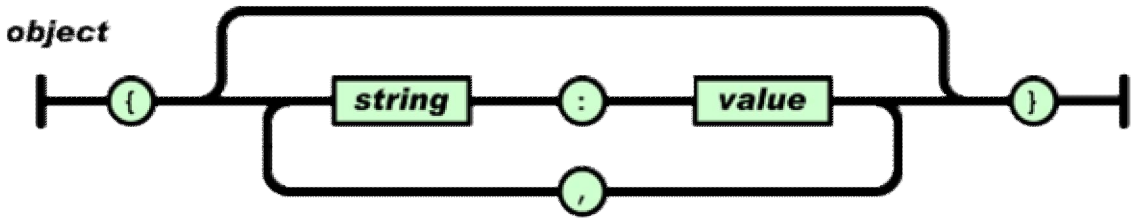

Figura 11 - Estrutura básica de um objeto JSON
Fonte: JSON ORG.

O JSON suporta a estrutura de *array,* que é uma coleção de valores e que segue o mesmo princípio básico do objeto onde os valores são separados por vírgula.

O valor contido no *array* pode ser qualquer um dos tipos suportados pelo JSON, que são: números, *string*, objeto, *array,* `true, false, null`.

A **Figura 12** mostra um exemplo de estrutura de um array no formato JSON.

```
{"status":
    {"code":"0","message":"Ok"},
    "events":[
        {
            "city":"CIANORTE",
            "dateBegin":"2012-08-11T22:00:00-03:00",
            "description1":"LUCAS E VINICIUS",
            "idEvent":"7801343152809230",
            "producer":"RANCHO URBANO EVENTOS LTDA ME",
            "state":"PR"
        },
        {
            "city":"PORTO SEGURO",
            "dateBegin":"2013-02-13T22:00:00-02:00",
            "description1":"CARNAPORTO",
            "idEvent":"1621343737686337",
            "producer":"CABANA AXE MOI LTDA",
            "state":"BA"
        }
    ]
}
```


Figura 12 - Exemplo de estrutura de array JSON
Fonte: Autoria própria.

3. DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo, será apresentada a documentação do protótipo na notação UML, bem como os módulos necessários para que o produtor possa consultar e administrar um evento através de um dispositivo móvel com sistema operacional *Android*.

Os requisitos mínimos necessários para operação da aplicação são descritos a seguir:

- Autenticação - Validação do usuário pelo canal *mobile* no sistema de *backend;*
- Consulta de Eventos - Listagem dos eventos ativos da produtora a qual pertence o produtor;
- Consulta de ingressos: Listagem dos ingressos criados para o evento selecionado;
- Agendar lote: Ação na qual o produtor pode agendar a virada de um novo lote de ingressos;
- Virada de lote: Ação na qual o produtor cria um novo lote de ingressos e determina a virada imediata deste lote;

3.1 DIAGRAMA DE CASO DE USO

O Diagrama de caso de uso abaixo mostra a descrição dos serviços oferecidos pelo sistema ao qual o usuário, no papel de Produtor, terá acesso.

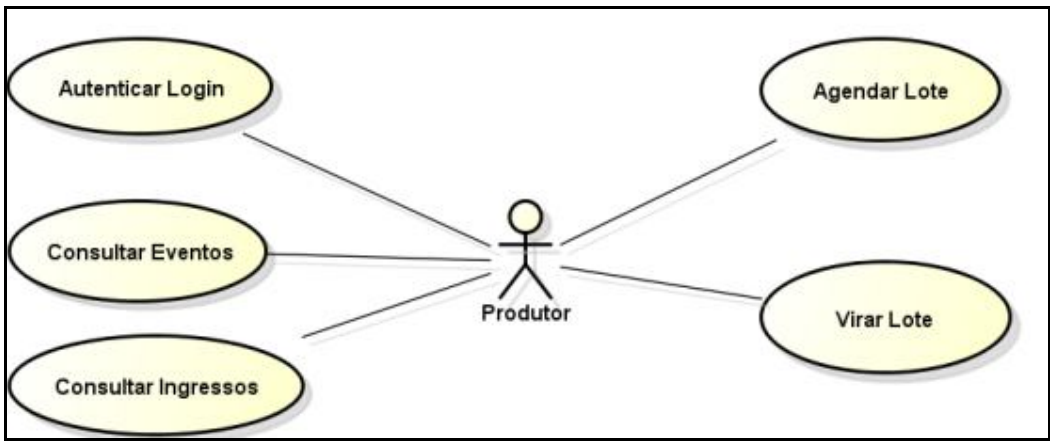

Figura 13 - Diagrama de Caso de Uso

3.2 DIAGRAMA DE CLASSES

Para ficar clara a colaboração entre as classes, tanto no lado da aplicação *backend*, quanto no lado da aplicação *Android,* o diagrama de classes foi dividido em dois e o diagrama a seguir mostra algumas das classes existentes no sistema de *backend* e que serão usadas na interação com a aplicação desenvolvida para o *Android,* utilizando como interface a classe *SiteService*, que disponibiliza os serviços.

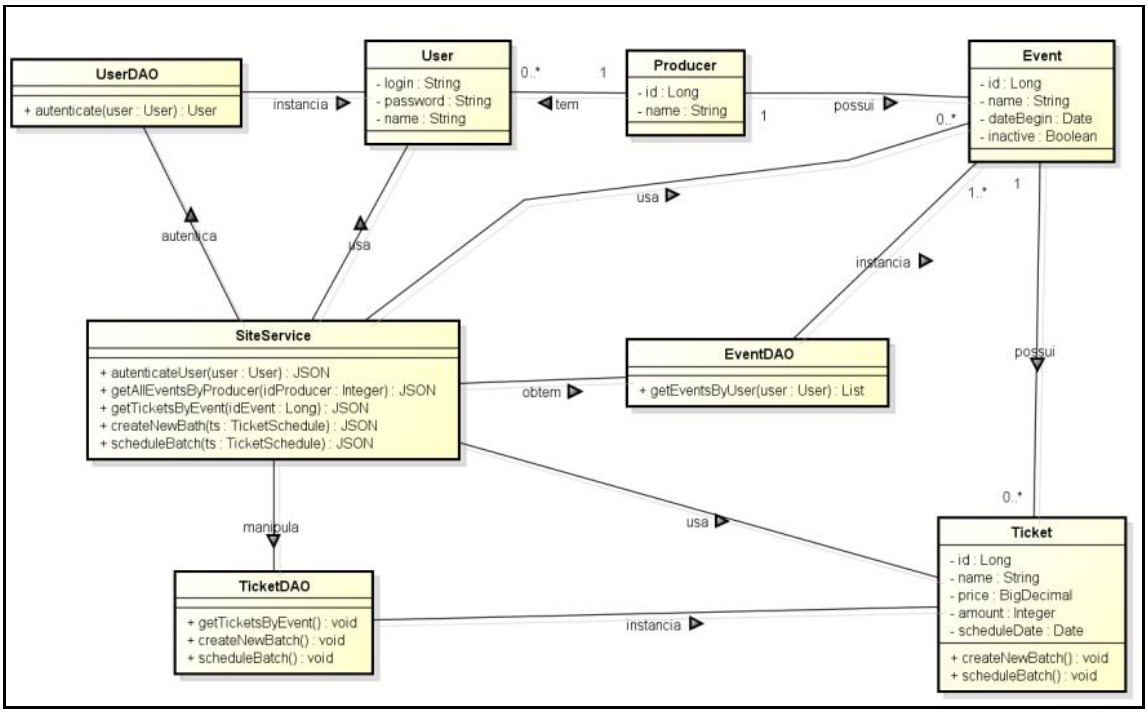

Figura 14 - Diagrama de Classes do backend

O próximo diagrama de classes ilustra a comunicação entre as classes do aplicativo desenvolvido para o *Android,* com o propósito de autenticar o usuário no sistema *backend* e, a partir daí, executar as operações descritas como requisitos da aplicação.

As classes que extendem de *Activity* são as telas da aplicação, responsáveis por montar o objeto ou a lista para a visualização no dispositivo.

A comunicação com a aplicação no servidor se dá através da classe *WebServicesClient*, que é responsável por enviar e receber requisições ao IP (Internet Procotol) onde o serviço está disponível e executar a operação disparada pelo usuário.

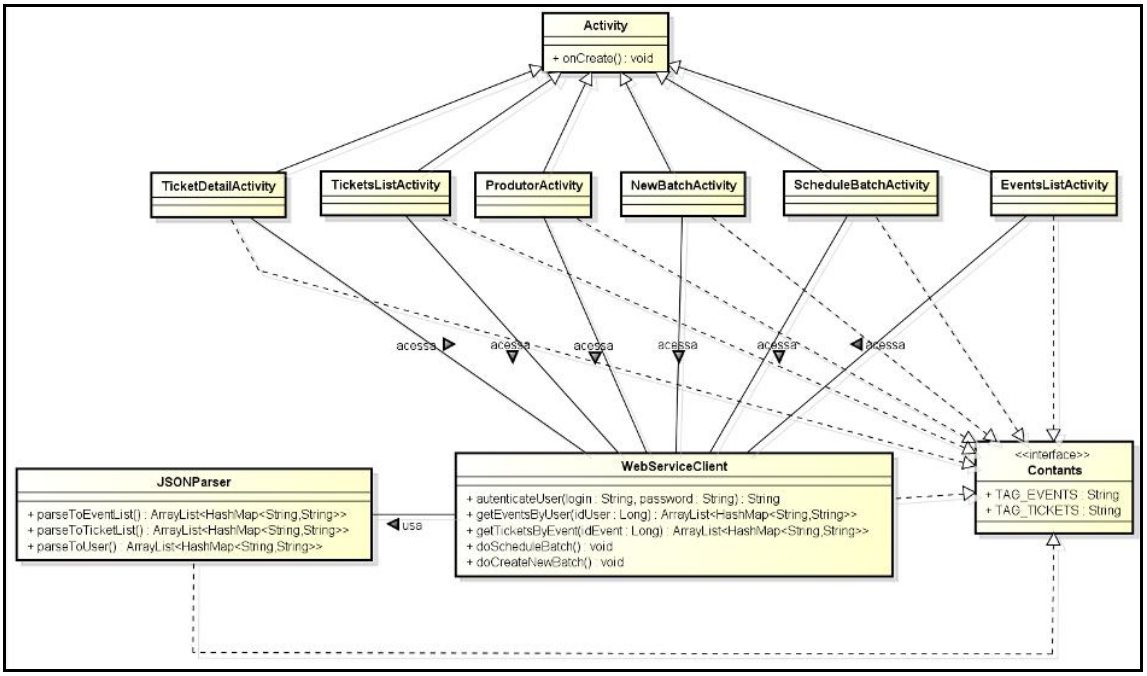

Figura 15 - Diagrama de classes da aplicação Android

3.3 DIAGRAMA DE SEQUÊNCIA

O diagrama de sequência mostra como os objetos interagem e a sequência em que isso acontece para atender a uma determinada operação solicitada pelo usuário.

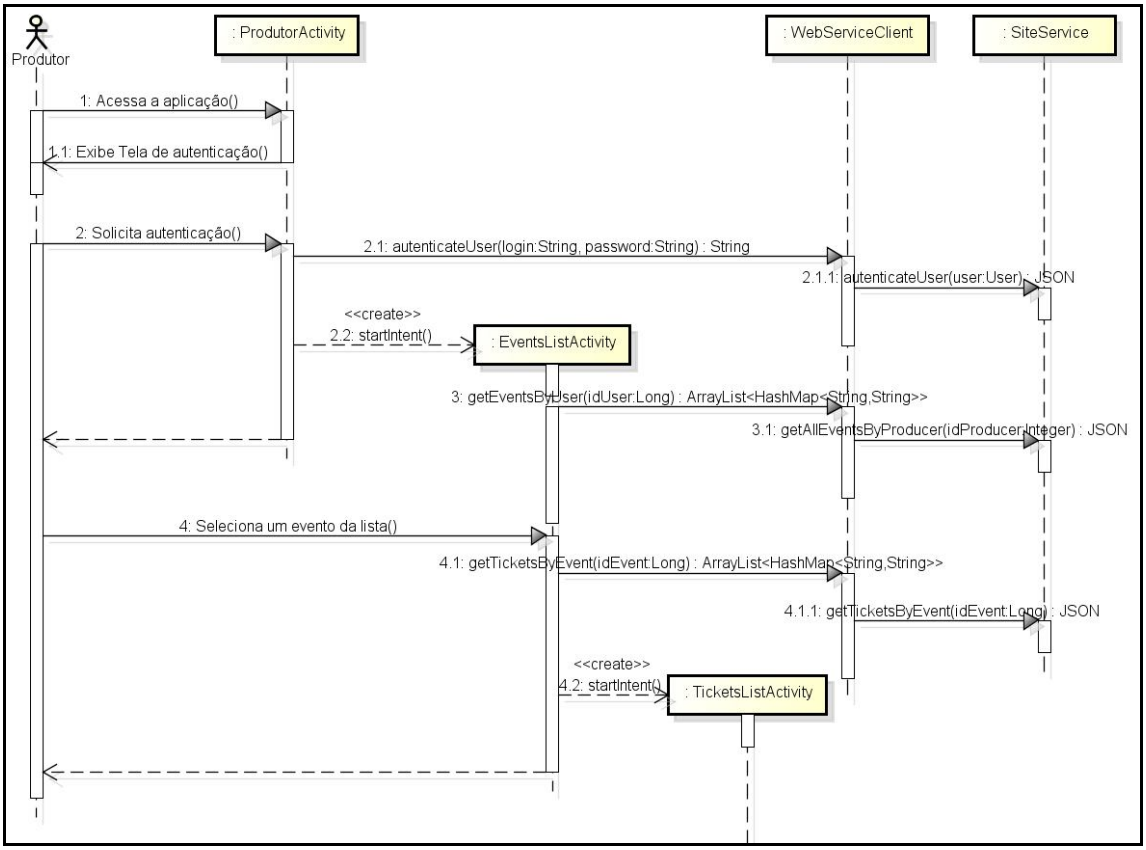

Figura 16 - Diagrama de Sequencia

Após a execução das operações descritas na **Figura 16**, o usuário tem a opção de agendar um lote ou criar um novo lote.

A sequência para executar as operações de agendamento e virada de lote é mostrada nos diagramas de sequência a seguir.

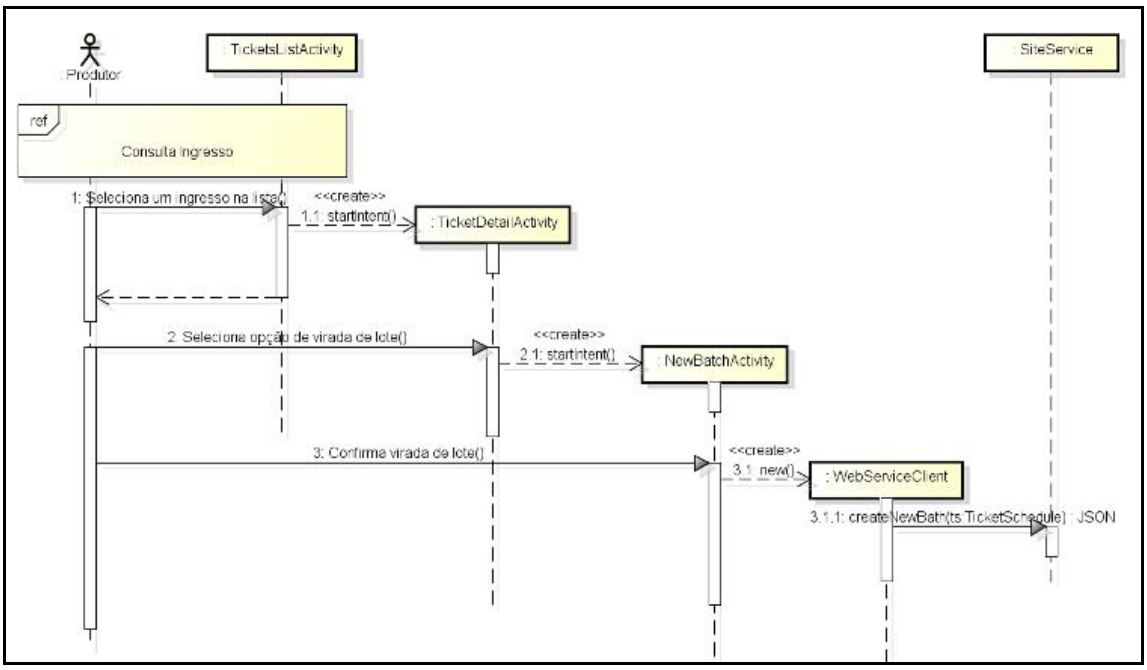

Figura 17 - Diagrama da Sequência de Virada de Lote

Estar na tela de lista de ingressos é um pré-requisito para que se possa executar a operação de agendamento.

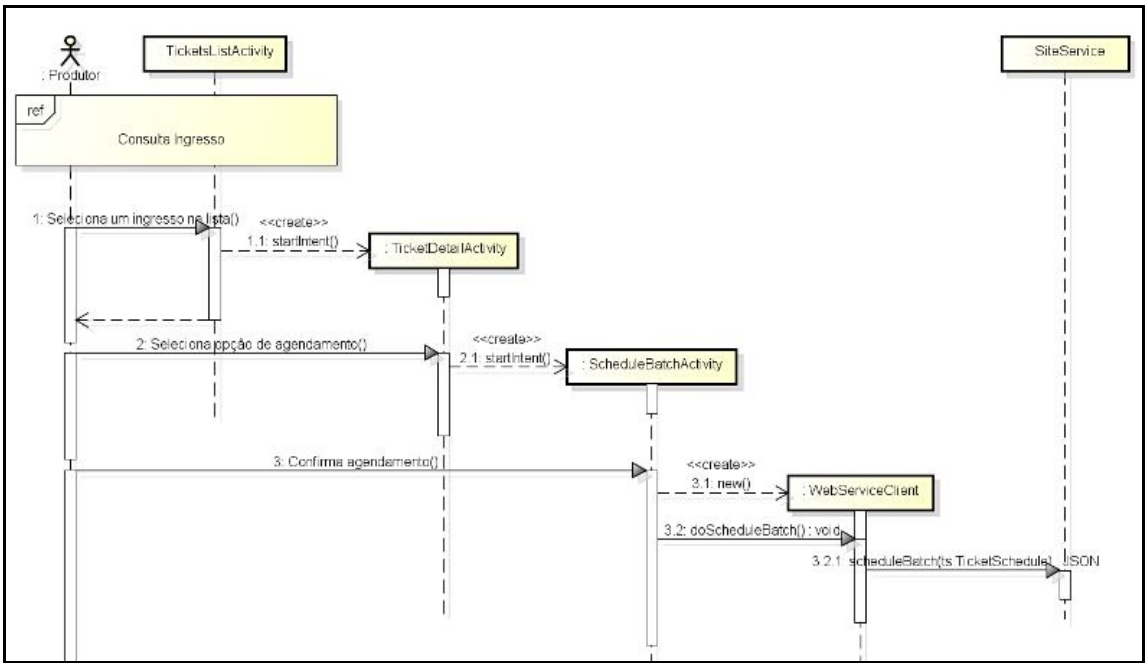

Figura 18 - Diagrama de Sequência de Agendamento de Lote

3.4 TELAS DO PROTÓTIPO

As figuras listadas a seguir são as telas do protótipo sendo executado no emulador fornecido pelo SDK do Android e contemplam os requisitos proposto no estudo.

3.4.1 Tela de autenticação do usuário

Para que o usuário possa acessar a aplicação, ele deve possuir um login válido e pertencer à uma produtora cadastrada e ativa no *backend.*

É através desta dependência usuário/produtora que a aplicação conseguirá listar os eventos permitidos para consulta pelo usuário.

A **Figura 19** mostra a tela inicial da aplicação onde o usuário deve informar os dados para solicitar a autenticação.

Uma vez autenticado, o usuário poderá navegar pela lista de eventos ativos relacionados à mesma produtora a qual o usuário está cadastrado no *backend.*

Figura 19 - Tela de autenticação do usuário

3.4.2 Tela de listagem dos eventos

Todos os eventos listados para consulta pelo usuário estão ativos no sistema, ou seja, estão em andamento.

A necessidade de poder interagir com a aplicação em qualquer lugar, de maneira simples, é um dos fatores que levaram ao estudo proposto.

As informações apresentadas ao usuário tem o objetivo de ajudá-lo na administração da venda dos ingressos de um evento, porém, sem consultas de *ranking,* já que o objetivo principal é o gerenciamento dos lotes de venda.

Figura 20 - Lista de eventos

3.4.3 Tela de listagem de ingressos do evento

A partir da tela de listagem de eventos, ao selecionar um evento, a aplicação aciona o *web service* responsável por retornar uma lista dos ingressos daquele evento, conforme demonstrado na **Figura 21**.

Figura 21 - Lista de ingressos do evento

3.4.4 Tela de detalhe do ingresso

Quando o usuário clica em um item da lista de ingressos, é mostrado à ele a tela de detalhe do ingresso, de onde é possível acionar a execução do agendamento ou criação de um novo lote de ingressos.

Figura 22 - Tela de detalhe do ingresso

3.4.5 Tela de agendamento do lote

O agendamento de lote permite ao usuário criar um novo lote de vendas, porém, deixá-lo em *stand-by* até a data da agenda.

Algumas regras são exigidas para que a operação seja realizada com sucesso, como, por exemplo, a data de agendamento deve ser maior que a data atual e o valor do ingresso deve ser maior que o valor atual do ingresso.

A simplicidade da aplicação é a chave para a aderência por parte dos usuários, visto que nem todas as pessoas que possuem *tablet* estão familiarizadas com a tecnologia.

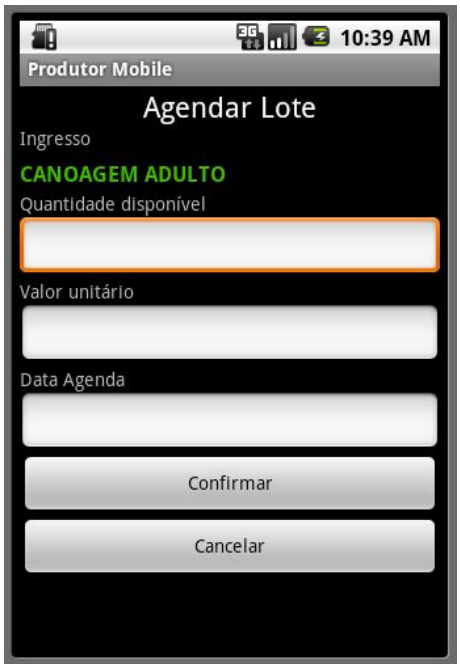

Figura 23 - Tela de agendamento de lote

3.4.6 Tela de virada de lote

Esta operação é bastante parecida com o agendamento de lote, porém, a diferença está no fato de que, no momento em que o usuário confirmar a ação, o novo lote será liberado para venda, parando a venda do lote atual.

Figura 24 - Tela de virada de lote

4 CONSIDERAÇÕES FINAIS

O estudo apresentado possibilitou verificar o crescimento constante da plataforma Android e das tecnologias utilizadas na integração entre sistemas na web.

Como a demanda deste tipo de integração aumenta muito, visto que as aplicações voltadas para uso na internet são muitas, a tendência é que surjam cada vez mais *frameworks* e tecnologias capazes de simplificar ainda mais o desenvolvimento de aplicações distribuidas.

Na mesma direção, segue o avanço da plataforma Android onde, com este estudo, foi possível conhecer as várias formas de implementação de um aplicativo e, como resultado, obter uma aplicação simples, porém, completa no sentido de utilizar os vários recursos oferecidos pela tecnologia, além de poder integrá-la com outros recursos da web, como foi o caso da utilização do RESTful e JSON para produzir e consumir recursos gerados por serviços web.

Com a difusão das tecnologias citadas neste estudo, o desenvolvimento do protótipo deu-se de forma bastante satisfatória, visto que, com uma rápida pesquisa no Google, é possível encontrar tutoriais e exemplos de aplicação das tecnologias, além de existir uma bibliografia bastante rica sobre o assunto, o que facilita bastante o conhecimento, aumentado a adesão à tecnologia.

A possibilidade de explorar e conhecer os vários recursos disponíveis no Android e de outras tecnologias envolvidas na utilização da web abre um leque bastante grande de conhecimento a ser adquirido, o que motiva outros trabalhos voltados para o uso desta tecnologia, agregrando novas funcionalidades ao protótipo desenvolvido durante o estudo.

REFERÊNCIAS

ABLESON, W. Frank et al. **Android IN ACTION.** 3. ed. New York: Manning Publications Co., 2012

ANDROID ACTIVITIES. **Android Developers - Activities.** Disponível em: <http://developer.android.com/guide/topics/fundamentals/activities.html>. Acesso em: 29 mai. 2012.

ANDROID DEVELOPER. **Android everywhere.** Disponível em: <http://www.android.com/developers>. Acesso em: 10 mar. 2012.

ANDROID SERVICES. **Android Developers - Services.** Disponível em: <http://developer.android.com/guide/topics/fundamentals/services.html>. Acesso em: 29 mai. 2012.

ANDROID SQLITE. **Android Developers – Data Storage.** Disponível em: < http://developer.android.com/guide/topics/data/data-storage.html#db>. Acesso em: 02 jul. 2012.

ANDROID PROVIDERS. **Android Developers - Providers.** Disponível em: <http://developer.android.com/guide/topics/providers/content-providers.html>. Acesso em: 03 jun. 2012.

DEBUG. **RESTful Web Services com Jersey.** Disponível em: < http:// nglauber.blogspot.com.br/2011/06/restful-web-services-com-jersey.html >. Acesso em: 27 jul. 2012.

IBOPE. **Consumidor pretende comprar mais smartphone do que celular comum.** Disponível em: < http://www.ibope.com.br/calandraWeb/servlet/CalandraRedirect?temp=5&proj=PortalIBOPE&pub=T&db=caldb&comp=IBOPE+Intelig%EAncia&docid=A57B524940DAA4B5832579BA0054101D>. Acesso em: 11 mar. 2012.

JSON. **JSON–RPC.** Disponível em: < http://json-rpc.org>. Acesso em: 19 jul. 2012.

JSON ORG. **Introducing JSON.** Disponível em: < http://json.org>. Acesso em: 20 jul. 2012.

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

MEIER, Reto. **Professional Android 2 Application Development – Wrox professional guides.** 2. ed. USA: John Wiley & Sons, 2010.

OHA OVERVIEW. **Open Handset Alliance - Overview.** Disponível em: <http://www.openhandsetalliance.com/oha_overview.html>. Acesso em: 21 mai. 2012.

ROGERS, Rick et al. **Android Application Development.** 1. ed. USA: O'Reilly Media, Inc., 2009

SANDOVAL, Jose. **RESTfull Java Web Services – Master core REST concepts and create RESTfull web services in Java.** USA: Packt Publishing Ltd., 2009.

TERRA. **Pesquisa: 14% dos brasileiros têm smartphone; Android domina.** Disponível em: < http://tecnologia.terra.com.br/noticias/0,,OI5776362-EI15606,00-Pesquisa+dos+brasileiros+tem+smartphone+Android+domina.html>. Acesso em: 16 mai. 2012.

VAZ, Giovana Aparecida et al. Sistemas de Informações Gerenciais: A importância da utilização do sistema Beta dentro dos processos decisórios e gerenciais da empresa Transportadora Alfa: Um estudo de caso**. Anais do 4 o. Encontro de Engenharia e Tecnologia dos Campos Gerais.** Ponta Grossa, Paraná - Brasil. 25 a 29 de agosto de 2008.