UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENAÇÃO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ALEX FREITAS MARIANO

CLAUDIO MARCELO DA SILVA MAIA

SISTEMA DE GERENCIAMENTO DE PEDIDOS PARA PLATAFORMA ANDROID

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2012

ALEX FREITAS MARIANO

CLAUDIO MARCELO DA SILVA MAIA

SISTEMA DE GERENCIAMENTO DE PEDIDOS PARA PLATAFORMA ANDROID

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

Orientador: Prof. Danillo Leal Belmonte

PONTA GROSSA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

SISTEMA DE GERENCIAMENTO DE PEDIDOS PARA PLATAFORMA ANDROID

por

ALEX FREITAS MARIANO
CLAUDIO MARCELO DA SILVA MAIA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado(a) em cinco de junho de 2012 como requisito parcial para a obtenção do título de Tecnólogo em Tecnologia em Análise e Desenvolvimento de Sistemas. O(a) candidato(a) foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Danillo Leal Belmonte
Prof.(a) Orientador(a)

_____
Wellton Costa de Oliveira
Membro titular

_____
Alison Roger Hajo Weber
Membro titular

_____
Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

_____
Simone de Almeida
Coordenador(a) do Curso
UTFPR - Campus Ponta Grossa

Dedicamos este trabalho a nossos familiares,
pois sem a força transmitida por eles
nada disso seria possível.

AGRADECIMENTOS

Agradeço antes de tudo a Deus por ingressarmos e adquirirmos valiosos conhecimentos durante nossa passagem pela UTFPR. Somos gratos pelo constante auxílio concedido para enfrentar o surgimento de novas dificuldades nesta etapa de nossas vidas.

Agradecemos aos nossos pais e familiares, por sempre nos apoiar e oferecer suporte durante toda nossa vida de estudos.

Agradecemos aos nossos professores que transmitiram toda a sua sabedoria a fim de nos preparar da melhor forma para a nossa vida profissional. Somos especialmente gratos ao nosso orientador, Prof. Danillo Leal Belmonte, pelo incentivo e ajuda durante a realização deste projeto.

E por fim agradecemos aos nossos amigos que em inúmeras ocasiões nos apoiaram, além de proporcionar memoráveis momentos de diversão.

RESUMO

MARIANO, Alex Freitas; MAIA, Claudio Marcelo da Silva. **Sistema de Gerenciamento de Pedidos para Plataforma Android**. 2012. Trabalho de Conclusão de Curso de Graduação – Curso de Tecnologia em Análise e Desenvolvimento de Sistemas – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

Devido a atual procura dos usuários por mobilidade surge um novo Sistema Operacional, o Android, desenvolvido pela empresa Google. Por conta disso, este trabalho destina-se a apresentar esta nova plataforma, exibindo sua arquitetura e especificações, além de projetar e desenvolver um aplicativo de Gerenciamento de Pedidos. Este *software* conta com diversas características como o uso de *Web service* para armazenagem de dados, sincronização entre banco de dados local e remoto, serviços de localização em tempo real, visualização de relatórios e gráficos, entre outras, sendo escrito na linguagem de programação Java, utilizando XML para modelagem da interface do usuário.

Palavras-chave: Android. *Web service. Google Maps*. Java. XML.

ABSTRACT

MARIANO, Alex Freitas; MAIA, Claudio Marcelo da Silva. **Order Management System for Android Platform**. 2012. Final Paper – Systems Analysis and Development Technology, Federal Technological University of Paraná. Ponta Grossa, 2012.

Due the current seek from users for mobility arises a new Operating System, the Android, developed by Google. Thus, this work is intended to present this new platform, showing its architecture and specifications, as well as design and develop an application of Order Management. This software counts on diverse characteristics like usage of Web service for storage of data, synchronization between local and remote databases, localization services in real time, visualization of reports and graphs, among others, being written in Java programming language, using XML for modelling the user interface.

Keywords: Android. *Web service. Google Maps*. Java. XML.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Em um curto período de tempo, surge uma infinidade de dispositivos eletrônicos devido a constante evolução da tecnologia. Desse modo, os sistemas de informação se beneficiam desses aparelhos para o cumprimento de tarefas do cotidiano.

As tecnologias de computação móvel encontram-se em evolução e parecem destinadas a transformarem-se no novo paradigma dominante da computação (MYERS et al, 2003).

Compartilhando dessa ideia, a empresa Google adquiriu os direitos sobre uma pequena empresa nomeada Android Inc. e posteriormente desenvolveu o Sistema Operacional (SO) Android (BUSINESSWEEK, 2005).

Baseado no SO Linux e com um ambiente de desenvolvimento flexível e poderoso, o Android é uma plataforma para dispositivos móveis que fornece as ferramentas necessárias para a criação de aplicativos, utilizando a linguagem Java, com suporte a diversos serviços e hardware, além de possuir um sistema de código aberto e software livre (LECHETA, 2009).

Comercialmente utilizada desde 2008, está cada vez mais presente no mercado, mostrando-se confiável e de fácil migração para as empresas de telefonia móvel. No seu curto período de vida, o SO do Google já assume a primeira posição no mercado mundial de smartphones.

O principal objetivo deste trabalho é o desenvolvimento de um aplicativo Android que gerencia os pedidos realizados. Além de exibir o funcionamento do SO Android, mostra como esta nova tecnologia é empregada em um sistema.

1.1 DELIMITAÇÃO DO TEMA

O presente projeto tem a finalidade de apresentar o SO Android, mostrar sua arquitetura e suas características e por fim expor as etapas de desenvolvimento de uma aplicação que realiza o gerenciamento de pedidos, foi escolhido por se tratar de um caso simples. O foco deste trabalho consistirá na apresentação da arquitetura do sistema em si (utilização do SO Android, entre outros).

1.2 ABORDAGEM DO PROBLEMA

É fácil imaginar durante o desenvolvimento de um projeto para dispositivos móveis que o usuário, operador do sistema terá, na maioria das vezes, acesso à internet. Entretanto, não se pode garantir que este serviço esteja disponível o tempo todo, podendo acarretar na perda de valiosas informações durante a comunicação com os serviços na *Web* (*Web services*).

A solução para este problema foi utilizar o gerenciador de banco de dados (SGBD) do Android, o SQLite. Ao usar este SGBD, presente nativamente no SO, garante-se o armazenamento da informação, uma vez que, mesmo não havendo comunicação com a *Web service*, todos os dados serão gravados localmente no aparelho e quando a conexão for estabelecida, estes dados serão enviados para o servidor, garantindo a integridade da informação de forma transparente para o usuário.

Outro problema é em relação ao tamanho do aparelho e de seu *display*, dificultando a exibição de um grande número de informações ao mesmo tempo, como nos *smartphones* e *tablets*. Esta dificuldade pode ser contornada ao utilizar os gerenciadores de layouts.

1.3 PREMISSAS DA APLICAÇÃO

Durante o desenvolvimento do sistema houve a necessidade de propor algumas premissas, pré-requisitos à utilização do sistema. As premissas são apresentadas a seguir:

- Impossibilidade da deleção de dados sempre que *off-line*: o usuário, independente do seu nível de acesso, não pode efetuar a deleção de dados, enquanto permanecer sem conexão com a internet, ou seja, *off-line*. Esta medida é tomada para evitar que dados sejam corrompidos enquanto outros usuários trabalham no sistema com constantes atualizações. Entretanto, as demais operações estarão disponíveis para estes usuários.
- Impossibilidade de proceder para a tela de gráfico sempre que *off-line*: o usuário, independente do seu nível de acesso, não pode entrar na tela de gráfico quando estiver sem conexão com a internet. Esta resolução é tomada para evitar

que o usuário tenha acesso a informações incorretas e desatualizadas, pelo fato dele estar *off-line* e o sistema ser constantemente atualizado.

- Impossibilidade de proceder para a tela de relatório sempre que *off-line*: o usuário, independente do seu nível de acesso, não pode entrar na tela de relatório quando estiver sem conexão com a internet. Procedimento tomado pelos mesmos motivos do tópico anterior.
- Impossibilidade de proceder para a tela de mapa sempre que *off-line*: o usuário, independente do seu nível de acesso, não pode entrar na tela de mapa quando estiver sem conexão com a internet. Este procedimento foi tomado devido à necessidade de se comunicar com os servidores da Google em tempo real, impossibilitando assim, a localização dos funcionários, clientes, checagem de rotas etc.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

O objetivo geral deste trabalho visa o desenvolvimento de um aplicativo para Android que faz o gerenciamento de pedidos, possibilitando ao usuário uma maior mobilidade, acessando o conteúdo pelo *smartphone* ou *tablet*.

1.4.2 Objetivos Específicos

- Compreender o funcionamento do SO Android e descrever suas características e arquitetura;
- Analisar o Android SDK utilizando a IDE Eclipse;
- Examinar sobre a tecnologia GPS e localização;
- Pesquisar sobre o banco de dados SQLite;
- Pesquisar sobre *Web services*;
- Modelar e implementar o sistema de gerenciamento de pedidos;
- Disponibilizar exemplos úteis, para ajudar na implementação de projetos semelhantes;
- Verificar e validar o sistema.

1.5 ESTRUTURA

O presente projeto é estruturado da seguinte maneira:

O capítulo 2 exibe todas as tecnologias, conceitos e embasamentos utilizados para o desenvolvimento do controle de pedidos.

O capítulo 3 explica a metodologia do trabalho, mostrando o desenvolvimento do projeto de controle de pedidos, desde os estudos iniciais do SO Android, apresentando a implementação e perícia adquirida até o projeto chegar a sua versão final.

O quarto e último capítulo é a conclusão do projeto, mostrando os trabalhos futuros e considerações finais.

2 EMBASAMENTO TEÓRICO

2.1 LINGUAGEM DE PROGRAMAÇÃO JAVA

Para começar o desenvolvimento de aplicativos para o SO Android é necessário ter uma noção da linguagem Java. Os conceitos mais importantes de Java são o de objeto e classe, onde aquele é qualquer coisa, real ou abstrata, na qual é possível armazenar dados e operações, e este é o agrupamento de objetos com a mesma estrutura de dados e operações, sendo um objeto a instância de uma classe (ORACLE, 2012).

Outro conceito não menos importante é o de herança, onde é permitido que uma classe seja criada a partir de outra classe e a nova classe herda todas as suas características (ORACLE, 2012).

2.1.1 Ambiente de Desenvolvimento Integrado

Ao desenvolver aplicativos para Android é necessário uma IDE (*Integrated Development Environment* ou, em português, Ambiente de Desenvolvimento) que possibilite maior agilidade no desenvolvimento, como o Eclipse, por ser a plataforma oficial de desenvolvimento Android (IBM DEVELOPER WORKS, 2010). O Eclipse é um ambiente de desenvolvimento de *software* multilinguagem, apresentando um extenso sistema de *plug-ins*, podendo ser utilizado para desenvolver vários tipos de aplicativos, usando várias linguagens (LEE, 2011).

Como foi necessário o uso de uma IDE para agilizar o desenvolvimento do aplicativo Android, não seria diferente na implementação e publicação dos *Web Services*. O NetBeans IDE apresenta mais recursos, como a criação de boa parte do *Web service* com o auxilio de interface gráfica (GOMES, 2010). O NetBeans é uma ferramenta de desenvolvimento de software gratuita e *open source* para desenvolvedores e utiliza diversas linguagens (NETBEANS, 2012).

2.2 XML

XML (*Extensible Markup Language*) é um aglomerado de regras recomendadas pela W3C (*World Wide Web Consortium*) para a codificação de informações estruturadas, com o objetivo principal de descrever dados. Segundo a W3Schools (2012), XML é o formato mais comum para transmissão de dados entre todos os tipos de aplicações.

Atualmente, o XML tem sido tão significante para a Internet quanto o HTML (*HyperText Markup Language*) foi na sua fundação, sendo este um instrumento frequentemente usado para a transmissão de dados entre aplicações heterogêneas. O XML não irá substituir o HTML, ambos foram projetados para diferentes objetivos.

É importante lembrar que o XML tem como objetivo estruturar, armazenar e transportar dados, com foco nos dados, sem misturar as informações contidas no documento com sua estrutura de formatação, facilitando qualquer leitura e edição posterior do documento por qualquer aplicação que de suporte.

Mesmo parecido com o HTML, o XML oferece ao programador muito mais liberdade na sua implementação, possibilitando ao usuário definir suas próprias *tags* e estrutura do documento, diferentemente do HTML onde as *tags* são predefinidas.

O XML é usado para a realização da GUI, interface gráfica de usuário, para requisição e repostas do *Web Service* e também para o *AndroidManifest*, mostrando ser uma linguagem de fácil manipulação.

2.3 WEB SERVICE

Na década de noventa, os modelos de computação distribuída eram muito utilizados para a integração de sistemas em ambientes de redes locais e homogêneos. Após o surgimento da Internet passou a existir a necessidade de integrar sistemas em ambientes de redes remotos e heterogêneos. Apareceu então o *Web service*, derivado de um consórcio composto por grandes empresas como IBM e Microsoft, entre outras pertencentes ao W3C (GOMES, 2010).

Segundo a W3C (2012), um *Web service* é um sistema projetado para suportar a interação inoperável máquina a máquina através de uma rede. Ele tem

uma interface descrita em um formato de máquina-processável. Outros sistemas interagem com o *Web service* de uma maneira prescrita pela sua descrição usando mensagens SOAP (*Simple Object Access Protocol*), tipicamente transmitidos utilizando HTTP (*Hypertext Transfer Protocol*) com uma serialização XML em conjunto com outros padrões web relacionados.

Gomes (2010) mostra que os *Web services* são uma tecnologia para a coesão de sistemas, empregada principalmente em ambientes distintos. Utilizando essa tecnologia podemos desenvolver softwares capazes de interagir, seja enviando ou recebendo informações com outros softwares, não importando a linguagem de programação que estes foram desenvolvidos, o sistema operacional em que rodam e o hardware que é utilizado. A única premissa é que para se comunicar com os *Web services* a troca de dados tem de ser feita no formato XML (GOMES, 2010).

A Figura 1 apresenta o funcionamento de um *Web service*, mostrando elementos e a sequência de uma chamada utilizando o padrão SOAP.

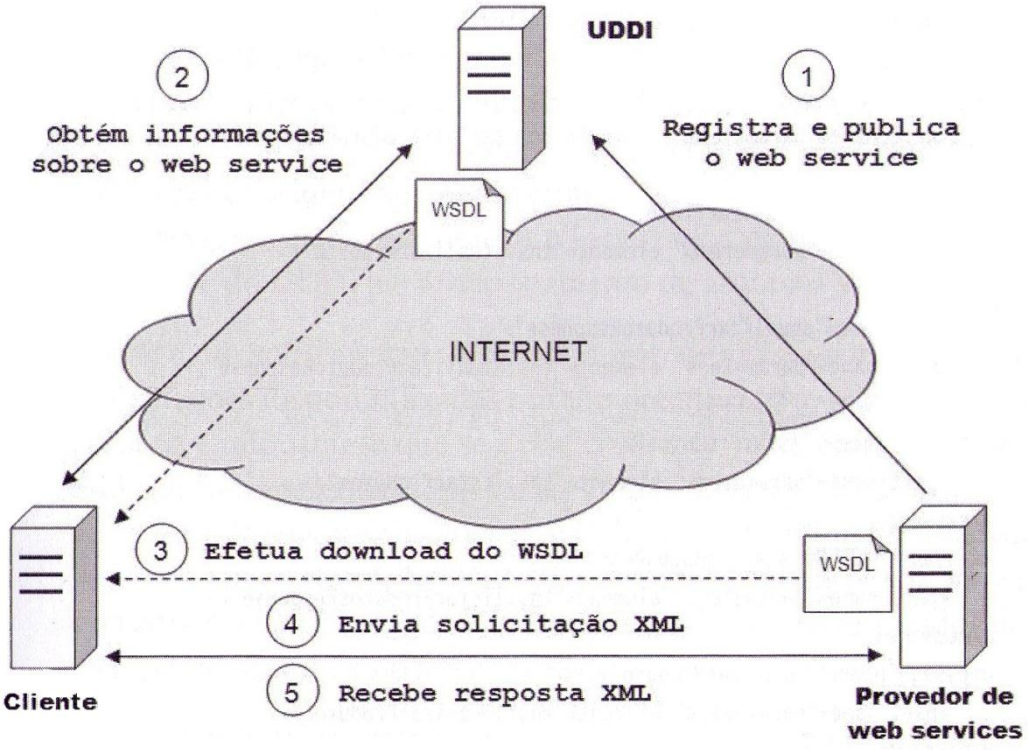

Figura 1 - Arquitetura para *Web services* SOAP criada pelo W3C.
Fonte: Gomes (2010, P. 15).

Primeiramente os detalhes dos elementos mostrados anteriormente:

- SOAP (*Simple Object Acess Protocol*): para realizar a transmissão de dados, dentro da arquitetura *Web service* proposto pelo W3C, é utilizado o protocolo-padrão SOAP, sendo este baseado no XML e no modelo "*Request-Response*" do HTTP.
- WSDL (*Web Services Description Language*): é um arquivo no formato XML, com a finalidade de descrever especificamente um *Web service*. Dentro do arquivo estão às operações do *Web service*, descritas de forma detalhada e clara como será o formato de entrada e saída de cada operação.
- UDDI (*Universal Description, Discovery and Integration*): é o mecanismo empregado para atender ambos, cliente e servidor, e tem a finalidade de armazenar arquivos WSDL. Além disso, principalmente, o UDDI fornece ao provedor de *Web services* métodos de registro e publicação para os *Web services*, possibilitando que os clientes pesquisem e localizem os *Web services* desejados.
- Cliente: é o sistema que usufruirá dos *Web services*, sendo responsável por utilizar as operações disponibilizadas através do arquivo WSDL, efetuando solicitações e recebendo os resultados do *Web service*, no formato XML.
- Provedor de *Web services*: é o local onde o *Web service* ficará armazenado. Este corresponde a um servidor de aplicações ou um web container, também armazena arquivos WSDL.

Após apresentar os elementos, apresentam-se as sequências que formam uma chamada a um *Web service* SOAP.

1) Registra e publica o *Web service*: tanto *Web services* quanto arquivos WSDL ficam armazenados juntos em provedores de *Web services*. Após a criação, o *Web service* é disponibilizado para o uso em um provedor de *Web services*, entretanto para que o cliente consuma o serviço, o *Web service* e o WSDL precisam ser localizados. Assim, segundo Gomes (2010), depois da criação e armazenagem de um *Web service* em um provedor é necessário o registro e publicação em um UDDI.

2) Obtém informações sobre o *Web service*: Para que o desenvolvedor utilize um *Web service*, primeiramente, é necessária a pesquisa em um UDDI pelo tipo de *Web service* que este pretende usar. Deste modo, o UDDI irá fornecer ao desenvolvedor o endereço do *Web service* e de seu WSDL.

3) Efetua download do WSDL: Após o desenvolvedor obter a localização, poderá fazer o download do arquivo WSDL, sendo possível posteriormente a criação do software cliente para a utilização do *Web service* desejado.

4) Enviar solicitação XML: O sistema cliente fará requisições ao *Web service*, mandando solicitações no formato XML.

5) Receber resposta XML: O *Web service* receberá a solicitação feita anteriormente, fará o processo requisitado pelo cliente e produzirá o resultado. O *Web service*, dependendo do tipo de operação, pode ou não enviar o resultado ao sistema cliente, no formato XML.

2.4 SISTEMA OPERACIONAL ANDROID

2.4.1 O que é o Android?

Como dito anteriormente, o Android é um Sistema Operacional móvel baseado numa versão modificada do Linux. Ele foi originalmente desenvolvido por uma companhia iniciante, a Android Inc. (LEE, 2011). Então em 2005, como parte de sua estratégia para penetrar no mercado de dispositivos móveis, o Google adquiriu os direitos sobre o Android e assumiu o trabalho de seu desenvolvimento.

O Google fez o SO Android ser gratuito e aberto, sendo liberado sob a Licença Apache de código aberto, mostrando que qualquer indivíduo que queira utilizar o Android poderá utilizá-lo sem receio, podendo até mesmo baixar o código-fonte completo do sistema operacional e alterá-lo, obedecendo sempre os termos de condições do software. É interessante citar o *Open Handset Alliance* (OHA), uma aliança composta por várias empresas responsáveis pela criação de padrões para a telefonia móvel. O Google está presente nesta aliança, além de Dell, Nvidia, Intel, Motorola, HTC, Samsung, LG, entre outras.

Baseado nisso o Android se tornou interessante para o mercado, despertando o interesse de vários fornecedores que poderiam usar o SO nos seus *hardwares* e ainda personalizá-los da maneira que achassem necessário, tornando seus produtos diferentes dos demais concorrentes.

Ainda é possível destacar outra grande vantagem, em relação ao desenvolvimento. Com o uso do SO Android, segundo Lee (2011), é possível ter uma abordagem unificada para o desenvolvimento de aplicativos. Portanto, ao

implementar um aplicativo para Android este deve ser apto a rodar em uma infinidade de aparelhos distintos, desde que esses aparelhos possuam o SO Android instalado.

Durante seu curto período de vida o Android passou por algumas atualizações. A Tabela 1 apresenta as várias versões do Android e seus codinomes. É importante salientar que a versão utilizada no presente projeto é a 2.2, codinome *FroYo*.

Tabela 1 - Versões do Android até o presente momento.

Versão Android	Data de Lançamento	Codinome
1.1	Fevereiro de 2009	-
1.5	Abril de 2009	Cupcake
1.6	Setembro de 2009	Donut
2.0/2.1	Outubro de 2009	Eclair
2.2	Maio de 2010	FroYo (Frozen Yogourt)
2.3	Dezembro de 2010	Gingerbread
3.0	Janeiro de 2011	Honeycomb
4.0	Outubro de 2011	Ice Cream Sandwich

Fonte: Autoria Própria.

2.4.2 Características e Arquitetura

O Android não tem configurações estabelecidas para hardware nem software, uma vez que há a possibilidade de personalização pelos fabricantes, em consequência do código-fonte aberto, todavia, segundo Lee (2011), o Android suporta as seguintes funcionalidades:

- Armazenamento – usa o SQLite, uma base de dados relacional leve, para armazenamento de dados;
- Conectividade – suporta GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, WiFi, LTE e WiMAX;
- Troca de mensagens – suporta tanto SMS quanto MMS;
- Navegador de Web – baseado no *WebKit* de código aberto, juntamente com o mecanismo de JavaScript V8 do Chrome;
- Suporte a meios – inclui suporte para os seguintes meios: H.263, H.264 (em contentor 3GP ou MP4), MPEG-4 SP, AMR, AMR-WB (em contentor 3GP), AAC, HE-AAC (em contentor 3GP ou MP4), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF e BMP;

- Suporte a hardware – sensor de acelerômetro, câmera, bússola digital, sensor de proximidade e GPS;
- *Multi-touch* – suporte a telas multi-toques;
- Multitarefa – suporte a aplicativos multitarefa;
- Suporte a Flash – suporte ao Adobe Flash;
- *Tethering* – suporte ao compartilhamento de conexões de Internet como um hotspot com e sem fios.

Do mesmo modo, a arquitetura do SO Android também é fracionada, sendo dividida em cinco partes, contendo quatro camadas fundamentais, como a figura 2 apresenta abaixo.

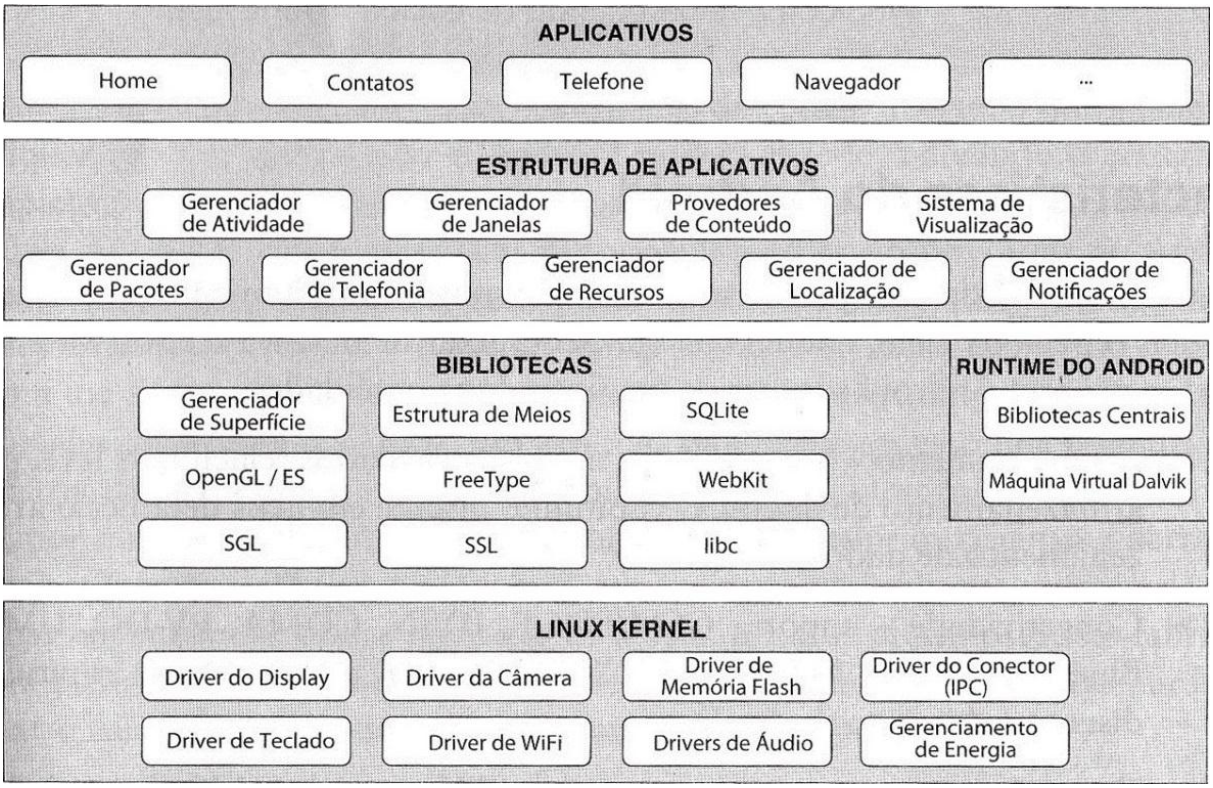

Figura 2 - Arquitetura do Android.

Fonte: Lee (2011, P. 4).

- *Kernel* Linux – é o núcleo em que o Android é baseado. Nesta camada está contida a totalidade de *drives* de baixo nível para os vários componentes de *hardware* de um dispositivo Android;
- Biblioteca – esta camada inclui todo o código que fornece as funcionalidades essenciais de um SO Android. As bibliotecas contidas nessa

camada são responsáveis pelas funcionalidades de manipulação de áudio, vídeo, gráficos, banco de dados e navegador.

- *Runtime* do Android – está presente na mesma camada das bibliotecas, possibilitando ao desenvolvedor implementar aplicativos para o Android usando a linguagem de programação Java. A *runtime* do Android também compreende a máquina virtual de *Dalvik*, que possibilita que cada aplicativo rode seu processo, na sua própria instância da máquina virtual *Dalvik*. O *Dalvik* é uma máquina virtual, idealizada especificamente para o Android e otimizada para dispositivos móveis alimentados por bateria, com memória e CPU limitadas (LEE, 2011);
- Estrutura de aplicativos – camada responsável por exibir as capacidades do SO Android para os desenvolvedores de aplicativos, mostrando a eles ser possível o uso dessas competências em seus próprios aplicativos;
- Aplicativos – é a camada mais elevada da arquitetura do sistema, onde estão presentes todos os aplicativos do Android, como e-mail, calendário, mapas, navegador, despertador, jogos, entre outros, além dos aplicativos baixados através do *Google Play* e os aplicativos implementados por desenvolvedores, como o prospecto contido neste documento, o projeto de gerenciamento de pedidos.

2.4.3 Kit de Desenvolvimento de *Software*

A linguagem utilizada para desenvolver aplicativos Android é Java, contudo é necessário outro software, o Android SDK (*Software Development Kit*, ou, em português, Kit de Desenvolvimento de Software), que fornece um conjunto rico de ferramentas, incluindo depurador, bibliotecas, emulador de telefone portátil, documentação, código de amostra e tutoriais (IBM DEVELOPER WORKS, 2010).

Para a implementação de aplicativos Android usando a IDE Eclipse é obrigatória a utilização de um *plug-in* que providencia um conjunto grande de ferramentas para o desenvolvimento, este plugin é chamado *Android Development Tools* (ADT), e suporta a criação e depuração de aplicativos para o Android (LEE, 2011).

Ainda segundo IBM Developer Works (2010) e Lee (2011), com o ADT é possível realizar as seguintes tarefas:

- Criação de novos projetos de aplicativos para Android;

- Acesso de ferramentas para utilizar simuladores e dispositivos Android;
- Compilação e depuração de aplicativos Android;
- Exportar aplicativos Android em Android Packages (pacotes do Android, arquivos .apk);
- Criação de certificados digitais para assinar o código dos aplicativos desenvolvidos;
- Utilizar recursos como assistente de conteúdo, recursos abertos, integração com JUnit;
- Visões e perspectivas distintas para o desenvolvimento de aplicativos Android;
- Abundância de widgets, semelhantes aos widgets do Java swing;
- Javadoc detalhado para ajudar durante o desenvolvimento.

Outra etapa importante no desenvolvimento de aplicativos Android é a fase de testes tornando-se necessária a criação de um AVD (*Android Virtual Devices*). Um AVD é uma instância de simulador que permite a modelagem de um dispositivo real, ou seja, possibilita emular um telefone Android, onde os aplicativos serão executados e testados (IBM DEVELOPER WORKS, 2010). A figura 3 exibe um AVD rodando, sua aparência assemelha-se a de um telefone celular real Android completo com teclado e *touchscreen*.

Figura 3 - AVD em execução.

Fonte: Autoria Própria.

Segundo Lee (2011), todo AVD é baseado em um perfil de hardware, um mapeamento da configuração do sistema e ainda um método de armazenamento simulado, semelhante a um cartão de memória.

É permitida a criação de inúmeros AVDs, para testar os aplicativos em diversas configurações de dispositivos, além de versões diferentes do SO Android. Esses testes são importantes para confirmar o comportamento de seu aplicativo quando ele for rodado em diferentes dispositivos, com diferentes capacidades (LEE, 2011).

2.4.4 Anatomia do Aplicativo

Os projetos desenvolvidos para o SO Android utilizando a IDE Eclipse apresentam alguns detalhes internos importantes para a compreensão do completo funcionamento do aplicativo. As várias pastas e arquivos contidos em um projeto Android são os seguintes, exibe Lee (2011):

- src – nesta pasta estão contidas todas as classes implementadas pelo usuário, os arquivos-fontes .java do projeto;
- gen – compreende os arquivos gerados automaticamente pelo AVD, como o R.java, que é gerado pelo compilador e referencia todos os recursos utilizados no projeto, não podendo este arquivo ser modificado;
- biblioteca – abrange todos os *plug-ins* utilizados durante o desenvolvimento da aplicação, como o android.jar, que contém as bibliotecas de classe essenciais para um aplicativo do Android;
- assets – contém os recursos usados pelo aplicativo, sendo na maioria das vezes arquivos de formatos distintos;
- res – é a pasta que contém todos os recursos utilizados no aplicativo, desde imagens e ícones de diferentes resoluções até arquivos de mapeamentos (como o strings.xml) e *layouts*. Dentro dela estão contidas determinadas subpastas como: drawable-<resolução> (destinados a arquivos no formato imagem), *layout* (destinados para a especificação, no formato XML, das telas de UI) e values (destinados para a definição de cadeias de caracteres do aplicativo).

- AndroidManifest.xml – é um arquivo de manifesto do aplicativo, compreende as permissões necessárias para o aplicativo realizar alguma ação, além de ter outras funcionalidades, como filtro de intenção, receptores, definição de atividades e suas ações, especifica a versão mínima do SO em que o aplicativo rodará, apresenta ícone utilizado, entre outras utilidades, assemelha-se ao webconfig.xml em aplicações web.

2.4.5 Atividades e Intenções

2.4.5.1 Atividades

Lee (2011) entende que uma atividade é, basicamente, uma tela que contém a interface de usuário de seus aplicativos, podendo estes aplicativos possuírem nenhuma (no caso serviços que não possui interface gráfica) ou mais atividades, sendo o seu principal objetivo a interação com o usuário.

Cada atividade apresenta um ciclo de vida, sendo esse um dos conceitos mais importantes para o correto funcionamento dos aplicativos Android. Ao criar uma atividade primeiramente se estende a classe base Java *Activity*.

Ainda segundo (LEE, 2011), a classe base *Activity* determina uma série de eventos que conduzem o ciclo de vida de uma atividade, como:

- *onCreate*() – chamado quando a atividade é criada. Vale lembrar que é neste evento que são exibidos os elementos UI da tela, isto é, chamar a *View* que será responsável por desenhar a interface gráfica da tela;
- *onStart*() – chamado quando a atividade fica visível para o usuário;
- *onResume*() – chamado quando a atividade começa sua interação com o usuário;
- *onPause*() – chamado quando a atividade presente é pausada e outra atividade está sendo retomada;
- *onStop*() – chamado quando a atividade já não está mais visível para o usuário;
- *onRestart*() – chamado quando a atividade foi suspensa e está reiniciando.
- *onDestroy*() – chamado antes da atividade ser destruída pelo sistema.

A Figura 4 apresenta o ciclo de vida de uma atividade.

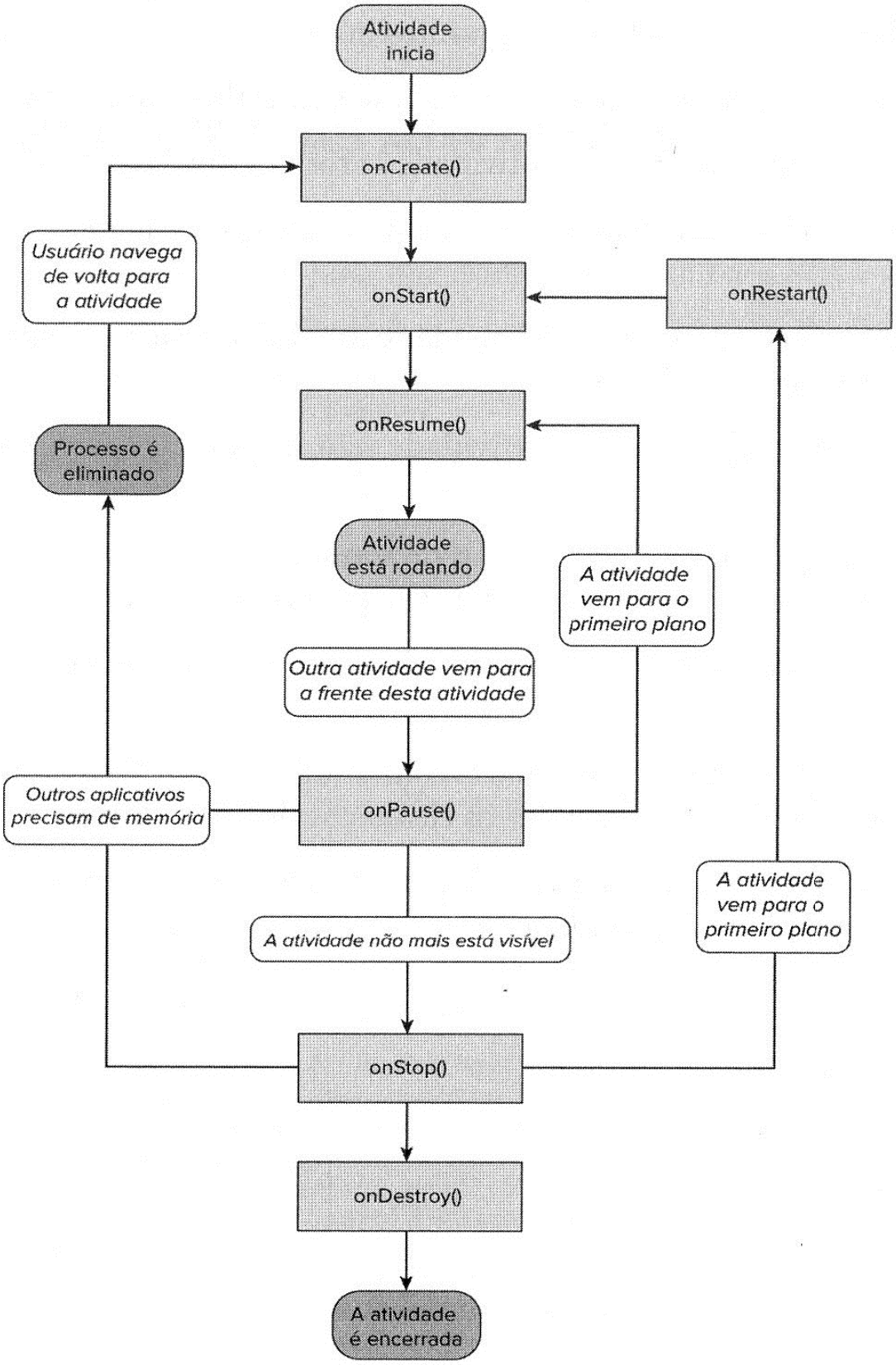

Figura 4 - Ciclo de vida de uma atividade.

Fonte: Lee (2011, P. 34).

2.4.5.2 Intenções

Após apresentar o importante conceito da atividade, é hora de mostrar outro, não menos importante, a intenção. Quando um aplicativo Android apresenta mais de uma atividade, torna-se necessário navegar de uma atividade a outra, desse modo para realizar tal tarefa é utilizada a intenção.

A intenção, grosso modo, é a "cola" que possibilita que diferentes atividades, de diferentes aplicativos, trabalhem juntas transparentemente, assegurando que tarefas possam ser realizadas como se todas permanecessem ao mesmo aplicativo (LEE, 2011).

Para navegar até outra atividade é usado o método *startActivity*(), criando uma instância da classe *Intent* e passando para esta o contexto da presente atividade e a classe da atividade desejada, o código necessário para isto é apresentado no pseudocódigo a seguir:

```
1 Intent i = new Intent(getApplicationContext(), Activity2.class);
2 startActivity(i);
```

Pseudocódigo 1 – Ligando atividades com intenções.

Vale ressaltar que atividades podem ser invocadas por qualquer aplicativo dentro do dispositivo, sendo indispensável que nomes distintos sejam dados para as atividades. A partir disso, também é possível chamar outras atividades/aplicativos para a própria aplicação, como navegador, fazer ligações, mandar e-mails, mostrar mapas, mostrar a lista de contatos, leitor de documento, entre outros.

Algumas vezes é necessário passar alguma informação de uma atividade para outra, nesse caso o objeto *Intent* também pode ser usado para passar dados à determinada atividade.

Após ter instanciado o novo objeto *Intent*, através do método *putExtra*(), são adicionados os dados. Para recuperar esses dados passados é usado o método *getIntent*(), para adquirir a intenção que iniciou a atividade e logo depois é empregado o método *getExtras*() para obter o objeto *Bundle*. Um objeto *Bundle* é basicamente um objeto dicionário que permite que você determine os dados em pares de valor/chave (LEE, 2011). E finalmente utilizando o método *getString*() é

recuperada a chave do objeto *Bundle*. Esses procedimentos são mostrados no pseudocódigo a seguir:

```
1 Activity1.java
2 Intent i = new Intent(getApplicationContext(),Activity2.class);
3 i.putExtra("projeto", "Sistema de Gerenciamento de Pedidos");
4 startActivity(i);.
5
6 Activity2.java
7 Bundle extras = getIntent().getExtras();
8 String project = extras.getString("projeto");
```

Pseudocódigo 2 – Passando e recuperando dados entre atividades.

2.4.6 Interface do Usuário – UI (User Interface)

2.4.6.1 ViewGroups

Anteriormente foi mostrado que a atividade é onde o usuário interage com a aplicação, entretanto, apenas a atividade não tem presença na tela. Para que apareça alguma informação na tela a atividade tem de desenhar a UI utilizando o *ViewGroups* (grupos de vistas) e *Views* (vistas), a partir de um arquivo XML.

A *View*, segundo Lee (2011), é um *widget* que tem uma aparência na tela, como botões, rótulos, caixa de texto, e ainda podem ser agrupadas em um *ViewGroup*. O grupos de vistas são responsáveis por organizar a sequência dos elementos que serão desenhados na tela.

Todo *View* e *ViewGroup* apresentam algumas características comuns, para especificar a altura, largura e o espaço que tal componente irá utilizar na tela, além de vários outros atributos que tornam cada *widget* único. Vale lembrar que é normal a combinação de distintos tipos de *ViewGroups* para arranjar uma interface. O Android interpreta os *ViewGroups* citados logo abaixo:

- *LinearLayout;*
- *TableLayout;*
- *AbsoluteLayout;*
- *RelativeLayout;*
- *FrameLayout;*
- *ScrollView;*

O *LinearLayout* acomoda os elementos (*Views*) em uma única coluna ou linha, onde estes podem tanto ficar na posição vertical quanto na horizontal. Para utilizar o *LinearLayout* é necessário declarar no documento XML a *tag* <*LinearLayout*>. A figura 5 mostra um exemplo do *LinearLayout*.

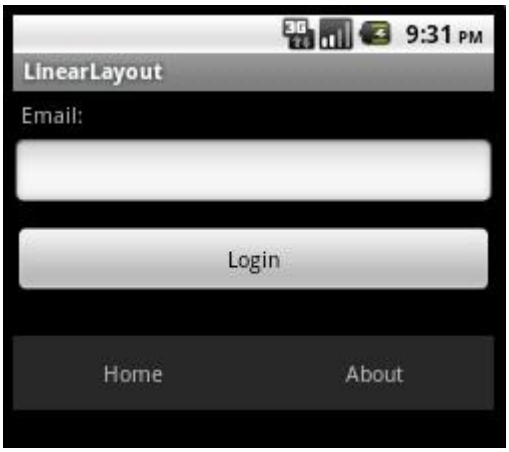

Figura 5 - LinearLayout.

Fonte: Autoria Própria.

O pseudocódigo 3 apresenta o documento XML referente ao *LinearLayout*, mostrado na Figura 5.

```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     android:layout_width="match_parent"
3     android:layout_height="match_parent"
4     android:orientation="vertical" >
5
6     <TextView
7         android:layout_width="fill_parent"
8         android:layout_height="wrap_content"
9         android:padding="5dip"
10        android:text="Email:" />
11
12    <EditText
13        android:layout_width="fill_parent"
14        android:layout_height="wrap_content"
15        android:layout_marginBottom="10dip" />
16
17    <Button
18        android:layout_width="fill_parent"
19        android:layout_height="wrap_content"
20        android:text="Login" />
21
22    <LinearLayout
23        android:layout_width="fill_parent"
24        android:layout_height="wrap_content"
25        android:layout_marginTop="25dip"
26        android:background="#2a2a2a"
27        android:orientation="horizontal" >
28
```

```
29          <TextView
30              android:layout_width="fill_parent"
31              android:layout_height="wrap_content"
32              android:layout_weight="1"
33              android:gravity="center"
34              android:padding="15dip"
35              android:text="Home" />
36
37          <TextView
38              android:layout_width="fill_parent"
39              android:layout_height="wrap_content"
40              android:layout_weight="1"
41              android:gravity="center"
42              android:padding="15dip"
43              android:text="About" />
44      </LinearLayout>
45
46  </LinearLayout>
```

Pseudocódigo 3 – Documento XML referente à interface gráfica do usuário.

O *TableLayout* também acomoda as *Views* em linhas e colunas. O elemento <*TableRow*> é usado para criar uma linha na tabela, podendo cada linha ter uma ou mais *Views*, consequentemente gerando uma célula. Para utilizar o *TableLayout* é necessário declarar no documento XML a *tag* <*TableLayout*>. A figura 6 exibe um exemplo do *TableLayout*.

Figura 6 - TableLayout.

Fonte: Autoria Própria.

O *AbsoluteLayout* admite que a localização dos elementos (*Views*) pertencentes a ele sejam especificados com precisão, onde deve-se passar a coordenada exata de sua posição na tela. Porém há um problema com esse *ViewGroup*, pois nem todos os dispositivos móveis tem a mesma resolução e por esse motivo é aconselhado que este não seja usado, a não ser que seu aplicativo

seja exclusivo para aparelhos com certa resolução e que esta resolução seja conhecida.

O *RelativeLayout* é organizado de maneira diferente, deixando que seja especificado a localização da *View*, porém levando em consideração a posição de outra *View*. São utilizados alguns atributos para posicionar a *View* de modo que fique alinhada a outra. Esses atributos possibilitam posicionar a nova *View* acima, abaixo, à direita, à esquerda e centralizado, todos levando em consideração a posição de outra *View*. Para utilizá-lo é necessário declarar no documento XML a *tag* <*RelativeLayout*>. A figura 7 mostra um exemplo do *RelativeLayout*.

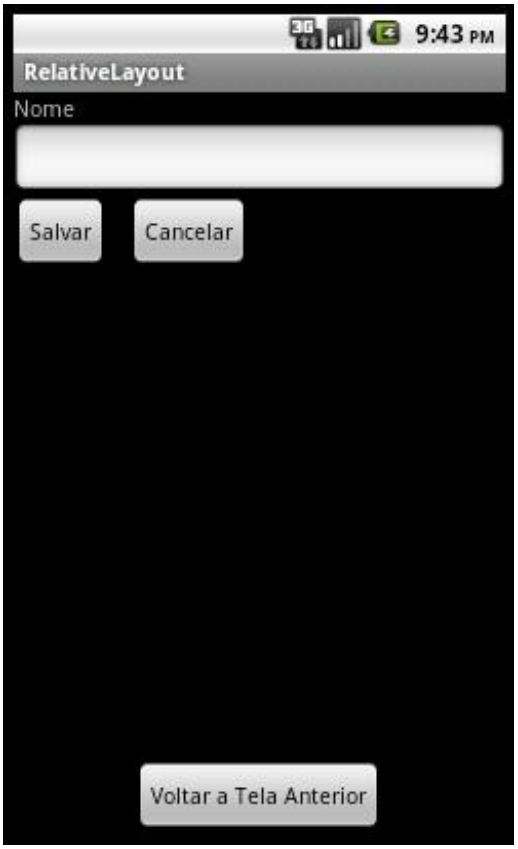

Figura 7 - RelativeLayout.
Fonte: Autoria Própria.

O *FrameLayout* funciona como uma pilha de objetos. Sempre que é adicionada uma nova *View* dentro deste *ViewGroup* cada *View* irá sobrepor a *View* anterior. É um guarda-vagas na tela que você pode usar para exibir uma única *View* (LEE, 2011). Para utilizá-lo é necessário declarar no documento XML a *tag* <*FrameLayout*>.

Outra recurso interessante é o *ScrollView,* usado para permitir que os usuários rolem por todas as *Views*, quando estas ocupam um espaço maior do que o disponível fisicamente no *display*, sendo extremamente útil quando o montante de informações para acomodar na tela é muito grande. Para utilizá-lo é necessário declarar no documento XML a *tag* <*ScrollView*>.

2.4.6.2 Views

Após mostrar no que consiste as *ViewGroups*, detalhando seus tipos, chegou a hora de apresentar os tipos de vistas (*Views*). As quais são basicamente divididas em três grupos, *Views* básicas, *Views* de seleção e *Views* em lista. *Views* de seleção não serão abordadas por não ser aplicadas no projeto.

As *Views* básicas possibilitam que informações textuais sejam exibidas, assim como fazer seleção básica. Algumas das *Views* básicas que serão explicadas logo abaixo são: *TextView, EditText, Button, ImageButton, CheckBox, ToggleButton, RadioButton, RadioGroup, ProgressBar* (ou *ProgressDialog*) e *AutoCompleteTextView*.

A *View TextView* é usada para mostrar alguma informação para o usuário, também é conhecida com *label* ou rótulo e possivelmente é a *View* mais utilizada durante o decorrer do desenvolvimento.

Entretanto essa não é a única *View* que é usada com frequência, o *EditText* e Button também são. O *EditText* é semelhante ao *TextView*, a diferença está no fato de que com o *EditText* pode-se inserir/editar textos. Já o Button e *ImageButton* são um botão de pressionar ou clicar, contudo o segundo exibe uma imagem no lugar do texto. Também presente no grupo dos botões, as *Views CheckBox* e *ToggleButton* apresentam dois estados, marcado e desmarcado.

A *View RadioButton* também apresenta dois estados: marcado ou desmarcado, contudo diferentemente dos dois anteriores este não pode ser desmarcado, podendo apenas marcar outro *RadioButton* no lugar. Quando há mais de um *RadioButton* é utilizado o *RadioGroup* para reuní-los, permitindo, assim, que somente um *RadioButton* esteja assinalado no *RadioGroup* (LEE, 2011).

A figura 8 apresenta todas as Views explanadas anteriormente, respectivamente *TextView, EditText, Button, ImageButton, CheckBox, RadioButton, RadioGroup e ToggleButton*.

Figura 8 - Views Básicas.

Fonte: Autoria Própria.

A *View ProgressBar* mostra ao usuário o progresso de alguma tarefa que está em andamento, sendo esta barra atualizada sucessivamente para representar a porcentagem da operação, fazendo com que o usuário fique ciente do processo que já foi concluído e o que ainda falta, eliminando as chances do usuário pensar que a aplicação travou. A *View ProgressBar* também pode apresentar o progresso de modo indeterminado, mostrando apenas uma animação recursiva em forma de círculo sem indicar o progresso. Por exemplo, a barra de progresso poder ser usada para mostrar o avanço de um download de uma imagem da internet, deixando o usuário a par do status da operação. A figura 9 mostra a *ProgressBar* nos seus dois modelos.

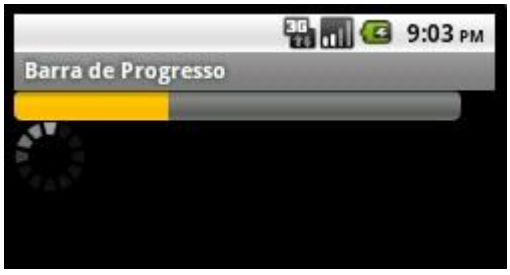

Figura 9 - ProgressBar.

Fonte: Autoria Própria.

Outra *View* básica é a *AutoCompleteTextView*, que basicamente é igual ao *TextView*, porém com a vantagem de mostrar automaticamente uma lista de sugestões enquanto o usuário está digitando (LEE, 2011). A lista de sugestões é mostrada como um menu *drop-down*, onde o usuário pode escolher o item para substituir o conteúdo do *AutoCompleteTextView*. Os dados contidos no menu vêm de um *ArrayAdapter* que controla a matriz de dados.

Um *ArrayAdapter* pode ser alterado para mostrar a informação da maneira desejada, como na figura 10, em que foi alterado o método de filtragem para procurar em toda a palavra por uma letra desejada, ao invés do método de filtragem padrão que apenas procura a palavra que começa com tal letra.

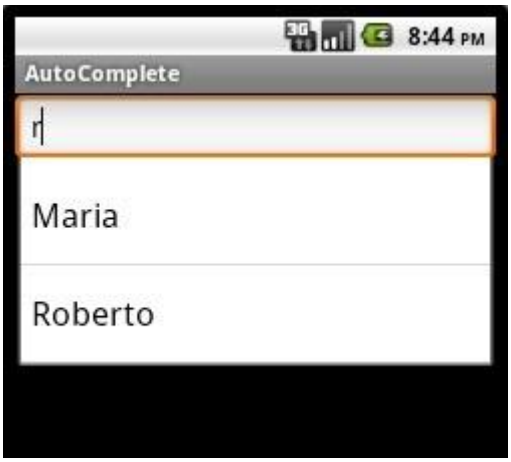

Figura 10 - AutoCompleteTextView.
Fonte: Autoria Própria.

Depois de mostrar as *Views* básicas, é indispensável apresentar também as *Views* em lista, que são *Views* que permitem que você exiba uma longa lista de itens, existindo dois tipos de *Views* em lista: *ListView* e *SpinnerView* (LEE, 2011).

A *View ListView* exibe uma lista de itens no estilo de lista de rolagem vertical e os dados contidos na lista também são controlados pelo *ArrayAdapter*. O *ListView* pode ainda ser personalizado para mostrar diferentes tipos de lista, como uma lista com múltiplos itens podendo ser selecionados ou habilitar o suporte a filtragem. A figura 11 apresenta um exemplo simples de *ListView*.

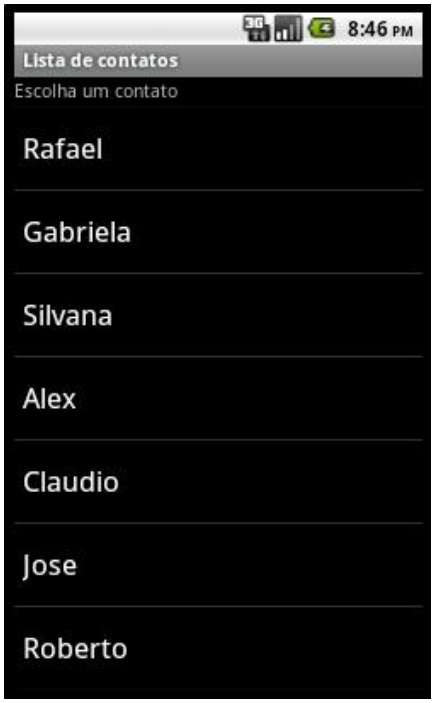

Figura 11 - ListView.

Fonte: Autoria Própria.

Para contornar a falta do componente *ComboBox* (caixa de combinação) existem as *Views AutoCompleteTextView*, já apresentada, e a *View SpinnerView*.

Figura 12 – SpinnerView

Fonte: Autoria Própria.

Para mostrar as informações de lista geralmente é utilizada a vista *ListView*, entretanto ao utilizá-lo é ocupado muito do espaço da tela, não sobrando

espaço necessário para outras vistas. Ao utilizar o *SpinnerView,* apresentado na figura 12, este contratempo não ocorre, já que é exibido um item de cada vez, a partir de uma lista, igualmente controlados pelo *ArrayAdapter*, e permite que o usuário selecione um item da lista, também sendo possível a sua customização (LEE, 2011).

2.4.7 Persistência de Dados

2.4.7.1 Persistência em Arquivos

Uma etapa importante no desenvolvimento de aplicativos é a persistência dos dados, pois permite que os usuários recuperem as informações digitadas em utilização posterior.

O Android possibilita que os dados sejam armazenados de diferentes maneiras, como por meio da preferência, uma técnica em que é possível guardar informações simples de aplicativos. Geralmente é usada para salvar as preferências do usuário, como tamanho, cor, tipo da fonte do texto, qualquer informação onde seja necessário o armazenamento de elementos do usuário.

Além dessa maneira de persistência também é possível persistir dados em arquivos. Uma forma de salvar arquivos é escrever no armazenamento interno do dispositivo, onde é usado a classe *FileOutputStream* para salvar o texto em um arquivo e para ler o conteúdo do arquivo é usada a classe *FileInputStream*, juntamente com a classe *InputStreamReader* (LEE, 2011).

Algumas vezes é necessário gravar um arquivo no armazenamento externo, cartão de memória (cartão SD), por não haver espaço suficiente na memoria interna do dispositivo ou por ser mais fácil de compartilhar informações com outros usuários, apenas tirando e inserindo o cartão em outro dispositivo. Pequenas mudanças são suficientes para armazenar os arquivos externamente, segundo Lee (2011), primeiramente é indispensável à criação de um diretório no cartão de memória pelo caminho *"/sdcard"*, sendo apenas necessário posteriormente salvar o arquivo neste diretório.

2.4.7.2 Base de dados SQLite

Os tipos de persistência de dados mostrados anteriormente são vantajosos para salvar conjuntos de dados simples, entretanto para salvar dados relacionais uma base de dados se torna muito mais eficiente pelo fato de possibilitar a realização de consultas para recuperar informações específicas, permitindo ainda reforçar a integridade dos dados, especificando os relacionamentos entre diferentes conjuntos de dados (LEE 2011).

O sistema de base de dados utilizado pelo Android é o SQLite. O SQLite é uma ferramenta ou biblioteca desenvolvida na linguagem C, que permite com que desenvolvedores possam armazenar os dados de suas aplicações em tabelas e manipular esses dados através de comandos SQL (GONÇALVES, 2007). A vantagem desta base de dados é que ela não precisa realizar acesso a um SGBD (Sistema Gerenciador de Banco de Dados).

Segundo Gonçalves (2007), o SQLite, na prática, é como um “mini-SGBD” que pode criar um arquivo em disco, além de ler e escrever diretamente sobre este arquivo, criado com o objetivo de manter diversas tabelas. A biblioteca SQLite é simples de usar, podendo-se manipular os dados contidos nas tabelas através de instruções SQL. Para criar uma tabela é utilizado o comando *CREATE TABLE*. Assim os dados presentes nas tabelas são manuseados através dos comandos *INSERT, UPDATE e DELETE* e ainda as consultas são realizadas com o uso do comando *SELECT*.

Algumas características do SQLite são:

- Software gratuito, multiplataforma, desenvolvido por D. Richard Hipp utilizando linguagem de programação C padrão;
- O banco de dados é mantido localmente, junto da aplicação, em um único arquivo;
- Não necessita de instalação, configuração ou de administração;
- Suporta a maior parte do SQL 92;
- Não oferece integridade referencial, suporte a Triggers, deleção em múltiplas tabelas, e consultas mais complexas que utiliza SQL;
- Suporta o uso de transações;
- Fácil de usar.

Vale a pena notar que a base de dados criada para um aplicativo só será acessível para ele mesmo, outros aplicativos não serão capazes de acessá-la (LEE, 2011).

2.4.8 Serviço de Localização *Google Maps*

Atualmente a maioria dos dispositivos móveis apresentam serviços baseados em localização, que determinam a posição do usuário e oferecem alguns serviços como traçar rotas, sugestões de pontos de lazer e comodidade, entre outros, além, é claro, de apresentá-los nos mapas, ponto chave deste tipo de aplicativo.

Um dos vários aplicativos fornecidos pela plataforma Android é o Google *Maps*, podendo este ser customizado de acordo com a preferência do usuário como: mudar as vistas (satélite, mapa, *StreetView* e tráfego), e recuperar informações do usuário como por exemplo: obter a latitude e longitude de localizações, realizar geocodificação e geocodificação inversa (adquirir endereço através da latitude e longitude, e vice versa), adicionar marcadores ao mapa, utilização de GPS (*Global Positioning System*) etc. (LEE, 2011).

Para iniciar o desenvolvimento de um mapa é necessário, primeiramente, de uma chave gratuita da API do *Google Maps*. Esta etapa é obrigatória, visto que será dado acesso aos dados presentes no *Google Maps*, após efetuar o registro e concordar com os termos de uso do Google. O registro consiste em assinar o aplicativo com a impressão digital MD5 do certificado, então é fornecida uma chave que está associada com o certificado do aplicativo do signatário (GOOGLE DEVELOPERS, 2012).

Depois de efetuar o registro e obter a chave do *Google Maps*, o mapa poderá ser exibido. O mapa é um tanto simples na etapa inicial em que é mostrado, entretanto o mapa pode ser personalizado para exibir controle de *zoom*, podendo mover o mapa para qualquer localização desejada dinamicamente e efetuar zoom para dentro ou para fora de uma localidade (LEE, 2011).

Outra customização que pode ser feita é a troca de vistas, que por *default* vem como vista do mapa, apresentando desenho de ruas e lugares de interesse. O *Google Maps* também pode exibir a vista do satélite, vista de rua (*StreetView*) e

ainda as condições atuais de tráfego, mostrando o estado do trânsito de lento a rápido por cores. A figura 13 mostra o mapa tanto na vista do mapa quanto satélite.

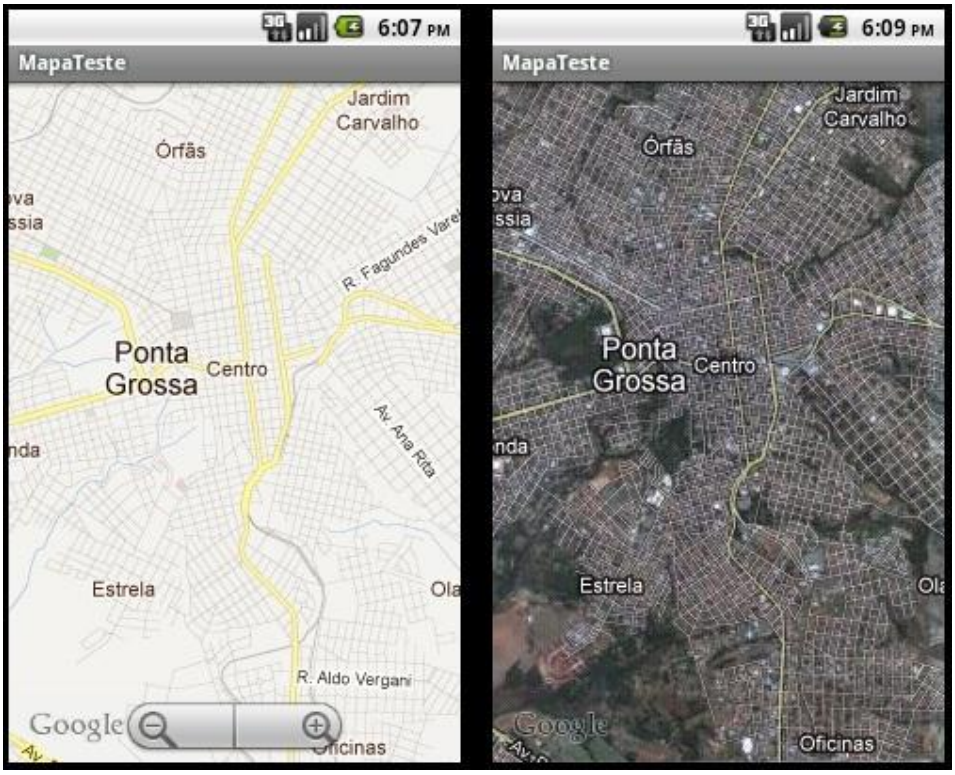

Figura 13 - Mapa com diferentes vistas.

Fonte: Autoria Própria.

Para facilitar a visualização do mapa faz-se necessário utilizar marcadores, tendo como objetivo auxiliar os usuários a localizar pontos de interesse no mapa, sendo ainda possível obter a latitude e longitude com simples toque na tela. As informações decorrentes desta ação são muito úteis para achar um endereço de uma localização desejada, procedimento conhecido como geocodificação inversa (LEE, 2011).

Segundo Lee (2011), a geocodificação é o processo para saber a latitude e longitude de um determinado endereço e geocodificação inversa é, obviamente, o processo inverso, realizado para descobrir o endereço possuindo latitude e longitude.

Para obter a localização atual dos dispositivos móveis normalmente usa-se um receptor GPS, por encontrar a localização facilmente, porém não é sempre que ele funcionará pela incapacidade do satélite penetrar determinados ambientes. A solução para este problema seria a utilização da triangulação de torres de celular ou triangulação de Wi-Fi. A triangulação de torres mantem contato constante com as

torres de celular que rodeiam o aparelho, podendo este traduzir as informações captadas em uma localização física, funcionando melhor em áreas densamente habitadas (LEE, 2011). A triangulação Wi-Fi usa o mesmo principio, contudo se conecta a uma rede Wi-Fi e compara o provedor de serviço com bases de dados para determinar a localização. Mesmo com estes aspectos, o mais preciso é o GPS (LEE, 2011).

2.4.9 Serviços

Um serviço é um aplicativo Android que roda em segundo plano, sem a necessidade de interação com o usuário, onde não é preciso a apresentação de uma UI ao usuário (LEE, 2011). Por exemplo, enquanto um usuário está utilizando um aplicativo, talvez ele queira escutar uma música enquanto realiza tal tarefa. Assim, a música de fundo não tem nenhuma necessidade de interagir com o usuário, portanto ela será executada como um serviço.

Ao implementar um serviço é importante executar esta tarefa assincronamente, permitindo que sejam realizadas as execuções em segundo plano, não causando a suspensão da atividade que o usuário está utilizando até o término do serviço. Outra questão importante, sem levar em consideração a duração do serviço está na finalização do mesmo onde este deve ser destruído para não consumir recursos valiosos do sistema desnecessariamente.

Na maioria das vezes um serviço é executado em seu próprio segmento, autônomo da atividade que o chama (LEE, 2011). Entretanto há vezes em que o serviço precisa interagir com a atividade, para mostrar que o serviço foi terminado ou para passar alguma informação que será utilizada pela atividade em seguida. A passagem e recuperação de dados é realizada da mesma maneira apresentada anteriormente, através do objeto *Intent*, empregando do método *putExtra*() e *getExtras*() (LEE, 2011). Outra maneira, e melhor, de passar e recuperar dados é ligar a atividade diretamente ao serviço, fazendo com que a atividade possa chamar qualquer membro ou método do serviço diretamente (LEE, 2011).

3 METODOLOGIA

3.1 DESCRIÇÃO GERAL

O sistema de gerenciamento de pedidos para SO Android, chamado BusinessUP, é uma aplicação para diferentes tipos de usuários, apresentando uma estrutura de nível de acesso, distinguindo os usuários em quatro grupos, Gerente, Vendedor, Fornecedor e Cliente, sendo apenas o Cliente a não ter acesso às funções do sistema.

Ao acessar o aplicativo a primeira tela mostrada será a Tela de Login, tendo o usuário duas opções, realizar o acesso ao sistema, quando já possuir uma conta, ou ir para a Tela de Registro, registrar uma nova conta. Quando o usuário tiver sua conta poderá realizar o login, sendo posteriormente apresentada a Tela do Menu Principal, onde, dependendo do seu nível de hierarquia dentro do sistema, serão apresentadas as seguintes opções: Tela de Clientes, Tela de Funcionários, Tela de Fornecedores, Tela de Produtos, Tela de Pedidos, Tela de Gráficos, Tela de Relatórios e Tela de Mapa.

As telas de Cliente, Funcionário, Fornecedores e Produtos apresentam similaridades, onde o usuário poderá cadastrar, alterar, apagar, pesquisar e navegar pelas informações contidas no banco de dados. A Tela de Pedidos, principal da aplicação, mostra o cliente e vendedor responsáveis, respectivamente, por ordenar e realizar o pedido, além de todos os produtos disponíveis para a seleção do cliente, posteriormente exibindo, após a escolha dos produtos e sua quantidade, o total do pedido.

Outras telas, não menos importantes, apresentam os resultados da interação dos usuários com o sistema. As telas de Gráficos e Relatórios mostram informações interessantes sobre todo o sistema, desde um relatório simples mostrando quais são os funcionários presentes, até um gráfico exibindo em quais meses há um maior número de pedidos.

A Tela de Mapa apresenta um mapa com diversas funções pra auxiliar o usuário durante a realização do pedido. O mapa exibe os funcionários que estão trabalhando no momento, com atualizações em tempo real, possibilitando acompanhar as atividades feitas pelos demais usuários. Outra função é a de traçar

rotas, dando ao usuário a oportunidade de traçar a rota até determinado cliente ou fornecedor.

3.2 VISÃO GERAL

O diagrama de casos de uso representado na figura 14 mostra uma visão geral do aplicativo Android, para simplificar o entendimento do sistema, apresentando o ator Usuário, sem se preocupar com seu nível dentro do sistema, interagindo com casos de uso que serão expandidos a fim de mostrar todas as funções presentes no projeto.

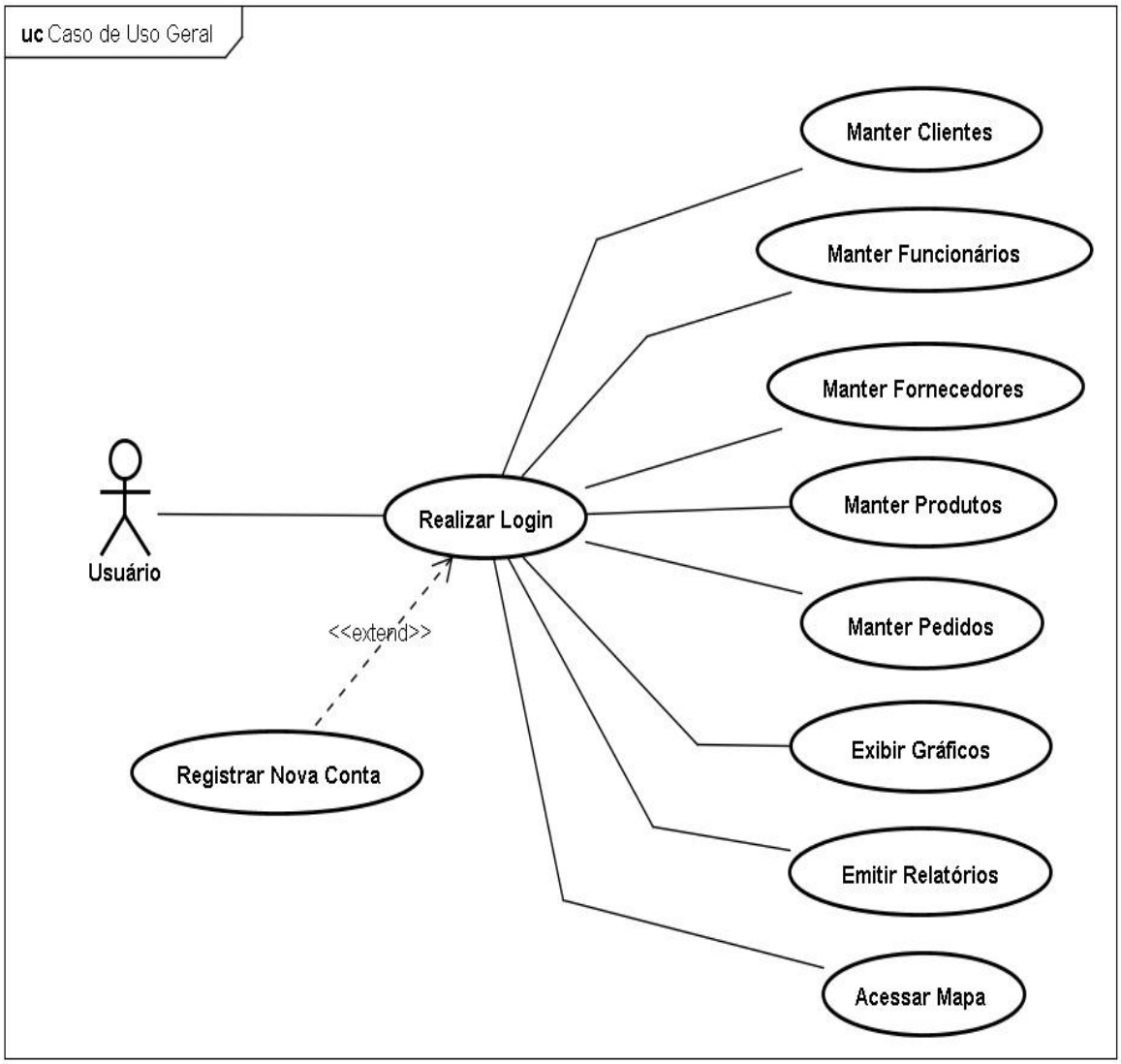

Figura 14 - Diagrama de Casos de Uso Geral do Projeto.
Fonte: Autoria Própria.

3.3 ESTRUTURA DO BANCO DE DADOS

O diagrama de classes apresentado na figura 15 exibe como foram mapeadas as tabelas dos bancos de dados, SQLite e PostgreSQL, presentes no projeto. É possível perceber que foram criadas seis tabelas, Participante, Rastreamento, Nivel_Acesso, Pedido, Item_Pedido e Produto.

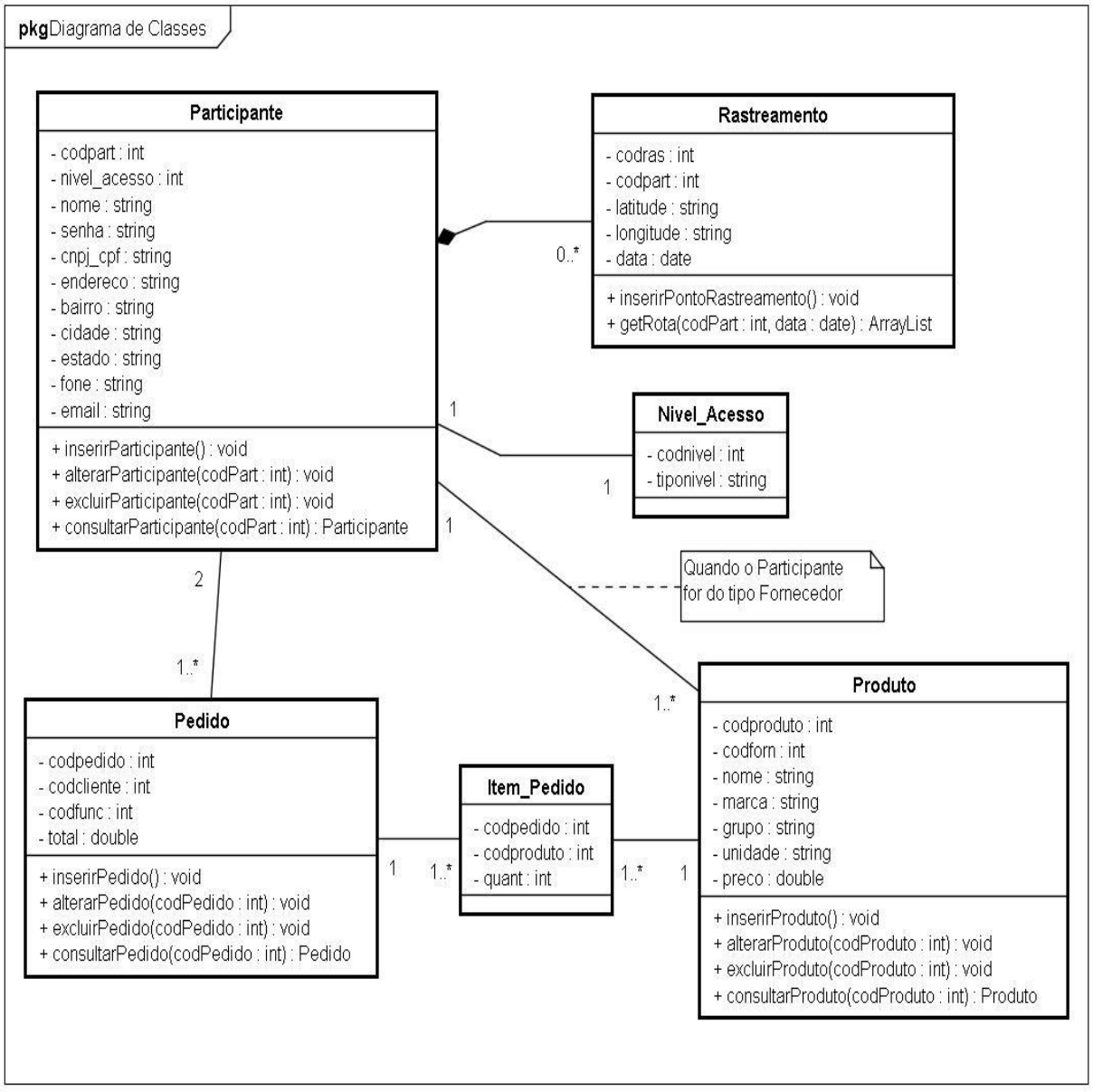

Figura 15 - Diagrama de Classes do projeto.

Fonte: Autoria Própria.

A tabela Participante engloba Gerentes, Funcionários, Fornecedores e Clientes, cada um com seu respectivo nível de acesso dentro do sistema, realizando

esta ligação com a tabela Nivel_Acesso. A mesma tabela é vinculada a tabela de Rastreamento devido ao fato dos funcionários enviarem constantemente informações sobre a sua localização, sendo rastreados por meio do GPS do dispositivo móvel. Outra ligação da tabela Participante é realizada com a tabela de Produto, pois, caso o participante seja um Fornecedor, este terá produtos vinculados ao seu registro.

A tabela de Pedidos se relaciona com a tabela Participante, visto que o Cliente pode ordenar algum pedido e Gerente ou Vendedor pode realizar algum pedido. Esta tabela também se relaciona com a tabela de Produto onde um pedido pode ter inúmeros produtos e um produto poderá pertencer a inúmeros pedidos, necessitando assim de uma tabela intermediária para administrar essas ligações chamada de Item_Pedido, além de registrar a quantidade solicitada de cada produto.

Existem dois bancos de dados no projeto, o SQLite, que armazena as informações localmente e o PostgreSQL, onde os dados são enviados por meio do *Web service* sendo posteriormente armazenados na base de dados no servidor.

O banco de dados SQLite é criado de maneira simples, por meio de um *script* SQL que cria o banco e as tabelas rapidamente, armazenando tudo dentro do dispositivo móvel. Para utilizar o *Web service* no aplicativo Android foi empregado o uso do *plug-in* Ksoap2, uma biblioteca SOAP leve e eficiente para clientes, que permite acessar *Web services*, requisitar e receber dados.

Ambos os bancos de dados tem a mesma estrutura e respeitam as mesmas regras, além de apresentar métodos de inserção, alteração, deleção e seleção idênticos, onde, por exemplo, os dois bancos contém o método de *login*, porém o SQLite é utilizado para *login off-line*, quando não há conexão com a internet, enquanto o *Web service* para o *login on-line*, quando existe conectividade.

3.4 ESTRUTURA DO APLICATIVO

Após representar a estrutura geral do aplicativo por meio do diagrama caso de uso, da figura 15, é interessante expandir este diagrama e apresentar mais a fundo as funcionalidades do sistema e como este se comporta em determinadas situações, além de exibir a interação de cada tipo de usuário e mostrar parte do código-fonte essencial para o funcionamento do aplicativo.

3.4.1 Registrar Nova Conta

A Tela de Registro é a tela que proporciona ao usuário a oportunidade de adquirir uma conta e assim ingressar no sistema. A Tela de Registro, apresenta um *layout* simples e objetivo, utilizando os *ViewGroups LinearLayout* e *ScrollView* em conjunto para acomodar as vistas, *TextView*, *EditText*, *RadioGroup*, *RadioButton* e Button, e fazer com que o aplicativo suporte diferentes resoluções. A figura 16 mostra o *layout* da Tela de Registro.

Figura 16 - Tela de Registro.
Fonte: Autoria Própria.

A principal funcionalidade da Tela de Registro é o cadastro de um novo usuário. Para efetuar o registro o usuário deve informar os dados necessários e pressionar o botão a fim de registrar uma nova conta. O método de cadastro verifica se os campos foram preenchidos corretamente e faz uma consulta por meio do *Web service* para constatar se o nome inserido já foi utilizado no sistema, seja por um funcionário ou fornecedor, retornando mensagens de erro apropriadas para cada evento se o resultado da verificação for incorreto. Após a verificação, e se estiver

tudo de maneira adequada, também através do *Web service* é realizado o cadastro do usuário no sistema, sendo possível efetuar o login.

3.4.2 Realizar Login

A primeira tela apresentada ao usuário quando este ingressa no sistema é a Tela de Login, sendo importante pelo fato de possibilitar a entrada do usuário no sistema, logicamente se o usuário já possuir uma conta. A Tela de Login tem o *layout* bastante parecido com o da Tela de Registro, apresentando, do mesmo modo, os *ViewGroups LinearLayout* e *ScrollView* unidos, contendo as vistas, *TextView*, *EditText* e *Button*. A figura 17 exibe o *layout* da Tela de Login.

Figura 17 - Tela de Login.
Fonte: Autoria Própria.

A Tela de Login apresenta algumas das funcionalidades mais importantes do sistema, como sincronização de dados, passagem de informação para outras telas, e, certamente, realizar login. Todos esses processos ocorrem dentro de uma *thread*, que é empregada para a utilização da vista *ProgressDialog*, mostrando ao

usuário informações sobre o estado da operação, desaparecendo logo quando for concluída.

O usuário tem a possibilidade de efetuar o acesso ao sistema de duas maneiras, o *login* online e o *login* off-line, entretanto antes é necessário saber como se encontra o estado de conectividade da rede. Através do método estaOnline(), mostrado a partir do pseudocódigo abaixo, é possível realizar esta tarefa.

```
1     public boolean estaOnline() {
2           ConnectivityManager cm = (ConnectivityManager)
3                       getSystemService(Context.CONNECTIVITY_SERVICE);
4           NetworkInfo netInfo = cm.getActiveNetworkInfo();
5           if (netInfo != null && netInfo.isConnectedOrConnecting()) {
6                 return true;
7           }
8           return false;
9     }
```

Pseudocódigo 4 – Método estaOnline().

O método booleano criado consiste em verificar se o dispositivo tem conexão com a internet ou não, utilizando a classe *ConnectivityManager*, linha 2, que responde a consultas sobre o estado da conexão, além de notificar o aplicativo sempre que ocorrem mudanças de conectividade de rede. Outra classe utilizada é a *NetworkInfo*, na linha 4, que descreve o estado da interface de rede. Ainda na linha 4, o método *getActiveNetworkInfo*() retorna a informação sobre o estado da rede para o objeto *netInfo*. O objeto *netInfo* é usado na verificação, assim como o método *isConnectedOrConnecting*() na linha 5, retornando verdadeiro se existe conectividade de rede ou está para ser estabelecida, ou falso caso não haja conexão. É importante destacar que o método estaOnline() precisa declarar permissões no *AndroidManifest* para ser utilizado, e também que este método é empregado durante todo o desenvolvimento do projeto.

Após o usuário informar os dados, o método estaOnline(), verifica o estado da conexão e decide qual processo de *login* será realizado, o *login on-line* ou *login off-line*. O *login off-line*, quando não for possível realizar conexão, utiliza o banco de dados SQLite para fazer a validação do usuário e também para entrar no sistema. O *login on-line*, quando há conectividade de rede, se assemelha ao *login off-line*, porém ao invés de usar SQLite é utilizado o *Web service*, tanto para a validação do usuário quanto para ingressar no sistema.

Todavia, o *login on-line* ainda traz outra utilidade, a sincronização de dados. Como o sistema utiliza duas bases de dados, SQLite e PostgreSQL por meio de *Web service*, é necessário que os bancos de dados não apresentem disparates, sendo essencial a sincronização entre ambos. Outro ponto importante é que não é sempre que o usuário dispõe de acesso à internet, por esse motivo o sistema apresenta a maioria das funções para ambos os meios, *on-line* e *off-line*. Neste sentido, a sincronização dos dados evita que tenham dados distintos nos bancos de dados, fazendo com que sempre tenham os mesmos dados.

O método de sincronização, primeiramente, seleciona todos os dados contidos no banco de dados SQLite e os envia e armazena em listas. Em seguida estas listas são utilizadas para fazer comparações e validações, atentando para os dados diferentes e iguais dos contidos no banco de dados PostgreSQL, onde serão, respectivamente, cadastrados e atualizados, através do *Web service*, no banco de dados PostgreSQL.

Após inserção e atualização dos dados no PostgreSQL, outra etapa do método de sincronização é feita, a deleção de todos os dados do banco SQLite, visto que já foram movidos para a base de dados online. Assim como feito antes, porém desta vez com o banco de dados PostgreSQL, todos os dados, agora sincronizados, são enviados e armazenados em listas. Como o banco SQLite está vazio, estas listas são empregadas para inserir os dados atualizados no banco de dados SQLite, terminando assim o processo de sincronização. As funcionalidades apresentadas anteriormente são exibidas para o usuário na forma de barra de progresso, como mostra a figura 18.

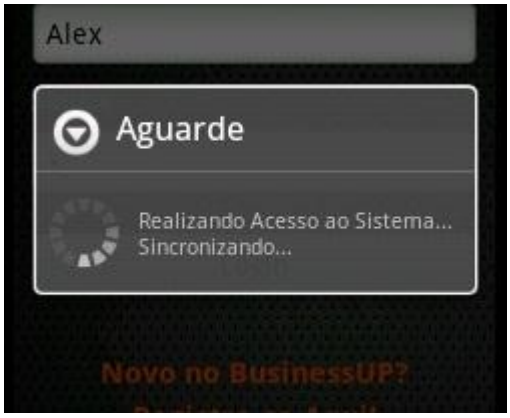

Figura 18 - Tela de Login mostrando o progresso dos processos.

Fonte: Autoria Própria.

Com a realização do *login* concluída, é necessário navegar até outra atividade, ou seja, ir para a Tela do Menu Principal. O pseudocódigo abaixo apresenta a ação de navegar até outra atividade, na linha 6, além da passagem de informações para a próxima tela como código, nome e nível de acesso do usuário, com a utilização do método *putExtra*(), presente a partir da linha 3.

```
1     Intent i = new Intent(getApplicationContext(),
2           TelaPrincipalActivity.class);
3     i.putExtra("codpart", participante.getCodpart());
4     i.putExtra("nomepart", participante.getNome());
5     i.putExtra("nivel_acesso", participante.getNivel_acesso());
6     startActivity(i);
```

Pseudocódigo 5 – Navegação e passagem de dados para outra Atividade.

A figura 19 exibe a Tela do Menu Principal, mostrando o nível de acesso do usuário e as demais funções do sistema. Esta tela utiliza parte do código do aplicativo Google I/O para modelagem de *layout*.

Figura 19 - Tela de Menu Principal.

Fonte: Autoria Própria.

A recuperação das informações passadas é feita através do método *getIntent*(), junto com outro método, *getExtras*(), recebendo a intenção que começou a atividade, atribuindo os dados ao objeto *Bundle*, que recupera esta informação através do nome passado anteriormente dentro do primeiro parâmetro no método *putExtra*(). O pseudocódigo a seguir mostra como a informação é recuperada, sendo importante informar o tipo de dado que será recuperado, como na linha 2, onde o tipo é inteiro.

```
1      Bundle extras = getIntent().getExtras();
2      int codpart = extras.getInt("codpart");
3      String nomepart = extras.getString("nomepart");
4      int nivel_acesso = extras.getInt("nivel_acesso");
```

Pseudocódigo 6 – Recuperação dos dados informados.

3.4.3 Manter Clientes

A Tela de Clientes é uma das opções que o usuário tem acesso no menu principal do aplicativo e assim como todo o aplicativo, possui um *layout* simples e de fácil manuseio. A Tela de Clientes emprega as *ViewGroups LinearLayout* e *ScrollView*, como nas telas anteriores, porém com a adição de uma nova *ViewGroup*, a *TableLayout*, afim de acomodar as vistas, *TextView*, *EditText*, *AutoCompleteTextView* e *Button*, em linhas e colunas gerando uma melhor adaptação e visualização, como apresenta abaixo a figura 20.

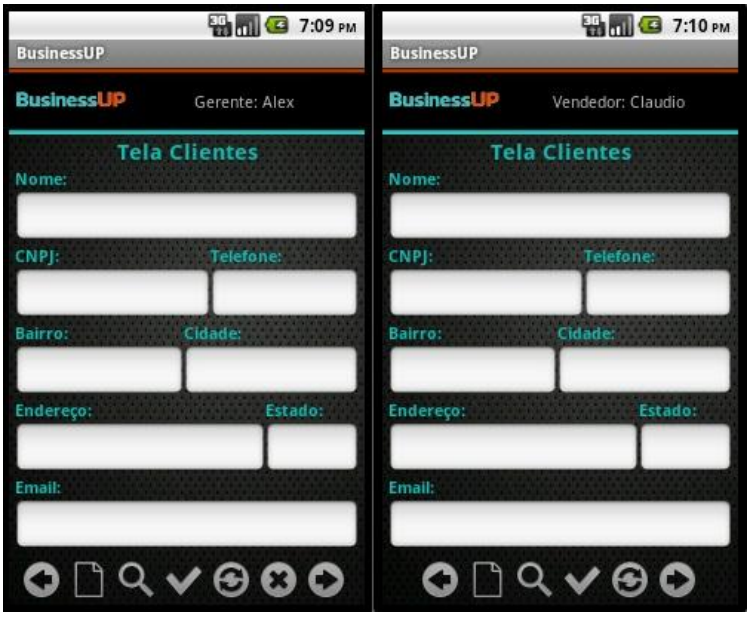

Figura 20 - Telas de Clientes mostrando níveis de acesso.

Fonte: Autoria Própria.

A partir da realização do login o sistema é completamente dividido em níveis de acesso, como mostra a figura 20. A Tela de Clientes pode ser acessada tanto pelo Gerente, quanto pelo Vendedor, sendo que este último sempre terá menos privilégios em relação ao Gerente, que se comporta como administrador do sistema. A maioria das funções da Tela de Clientes pode ser realizada por ambos os funcionários, a não ser a função de exclusão de clientes, exclusiva do Gerente.

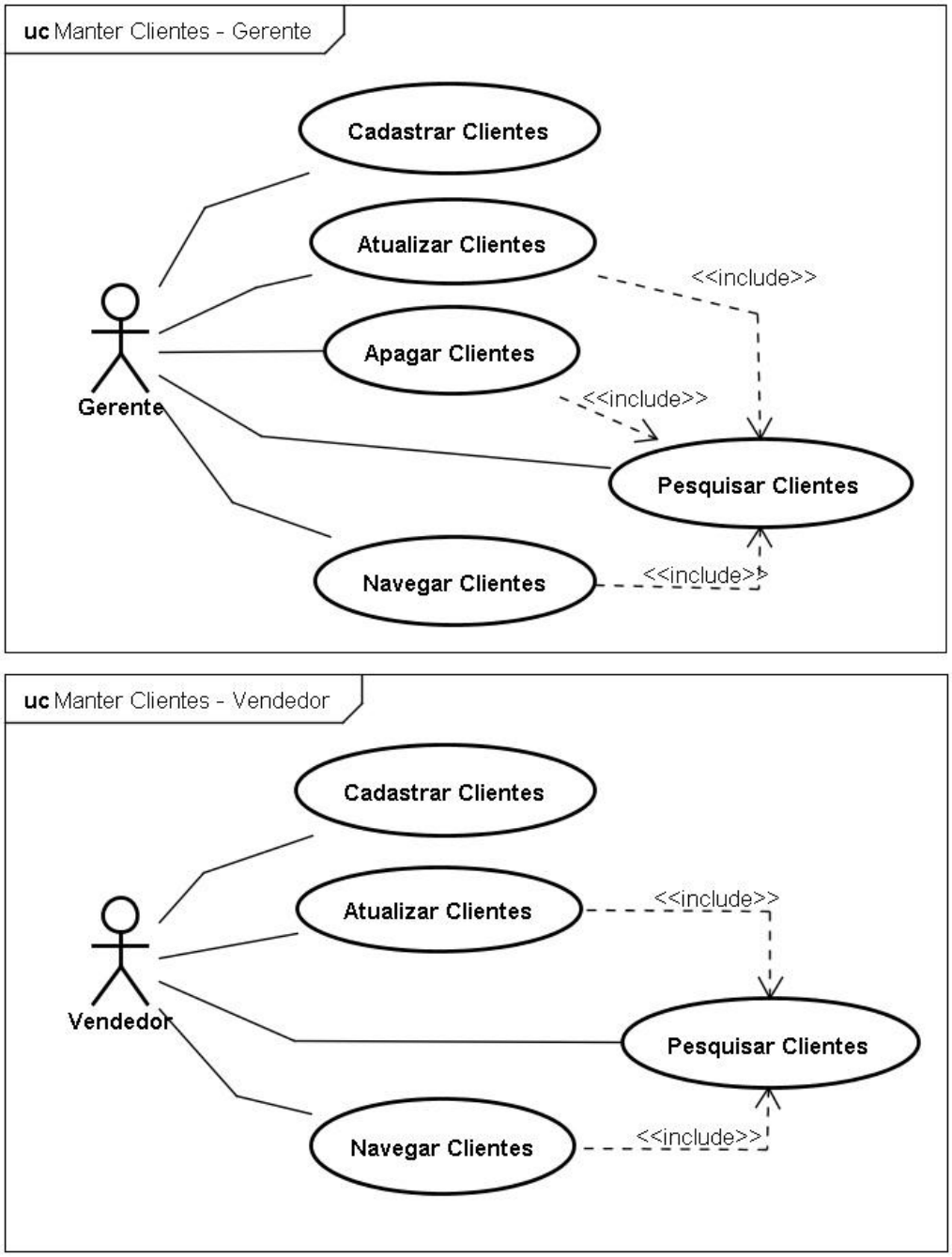

Figura 21 - Casos de uso Manter Cliente para Gerente e Vendedor.

Fonte: Autoria Própria.

A figura 21 apresenta as funcionalidades da Tela de Clientes, lembrando que o método estaOnline() é utilizado nesta tela para que não haja perda de informações em um eventual queda de conectividade de rede e que os dados podem vir de ambos bancos de dados, SQLite e PostgreSQL. As funções da Tela de Clientes são: navegar, cadastrar, alterar, apagar e pesquisar clientes.

A função navegar armazena todos os dados dos clientes em uma lista e depois os percorre enviando todos estes dados para as variáveis correspondentes, tornando possíveis as funções de alteração e exclusão, visto que é armazenada em uma variável o índice do cliente. A função de navegação possui dois botões, o anterior e o próximo, como apresentado na figura 20.

A função de cadastro, obviamente, insere os clientes na base de dados. Entretanto, uma série de validações são realizadas para assegurar que o banco de dados não guarde e forneça informações incorretas. Estas verificações consistem em averiguar se algum dado foi informado no cadastro, ao menos o nome de usuário deve ser fornecido, e também confere se o nome informado pelo usuário já está contido no banco, impossibilitando a inserção dos dados, por não tolerar nomes idênticos de clientes. Um processo interessante que ocorre junto à função de cadastro é a geocodificação inversa, que será usada na Tela de Mapa. Este processo obtém o endereço informado pelo usuário e realiza a geocodificação inversa para conseguir a latitude e longitude, empregando estas no mapa a fim de mostrar os clientes do sistema e ajudar a localização deste pelos funcionários.

A função de alteração compartilha dos mesmos princípios do cadastrar clientes, realizando as mesmas verificações de campos não preenchidos e a conferição de nomes já cadastrados no banco de dados, porém desta vez sendo diferente, já que, logicamente, o cliente terá o mesmo nome, pois está ocorrendo uma atualização. Então, a verificação realizada irá atualizar o registro sempre que o nome informado não existir ou o nome informado existir, mas com o mesmo índice dentro do banco de dados, ou seja, o registro utilizado no momento. As informações provenientes do processo de geocodificação inversa também serão atualizadas.

Presente somente para os Gerentes que se encontram online, a função de exclusão pega o índice do registro atual do cliente para fazer a exclusão dos dados, contudo, primeiramente, se certifica de não apagar um cliente que esteja “ligado” com outra tabela, por exemplo, nunca apagar um cliente que realizou um pedido, antes de desvinculá-lo do processo de pedido.

A última função, pesquisar, realiza a pesquisa de clientes através do nome. O campo usado para informar os caracteres da pesquisa é a vista *AutoCompleteTextView*, sendo que a medida que o usuário digita uma letra sugestões, que mais se assemelham com o caractere, são mostradas como um menu *drop-down*, possibilitando que o usuário escolha o cliente desejado. Após escolher o cliente os dados deste serão apresentados nos campos, sendo possível a realização das demais operações.

3.4.4 Manter Funcionários

A Tela de Funcionários, outra opção para o usuário estando presente no menu principal, apresenta um *layout* parecido com o da Tela de Clientes, exibindo as *ViewGroups LinearLayout*, *ScrollView* e *TableLayout*, e as vistas *TextView*, *EditText*, *AutoCompleteTextView*, *Button* e *Spinner*, sendo o única vista diferente. A figura 22 mostra o *layout* da Tela de Funcionários.

Figura 22 - Telas de Funcionários mostrando níveis de acesso.

Fonte: Autoria Própria.

A figura 22 mostra que a Tela de Funcionários pode ser utilizada pelo Gerente e pelo Vendedor, porém o Vendedor tem um papel muito menor nesta tela.

A maioria das funções da Tela de Funcionários é realizada pelo Gerente, restando para o Vendedor apenas a alteração da própria conta.

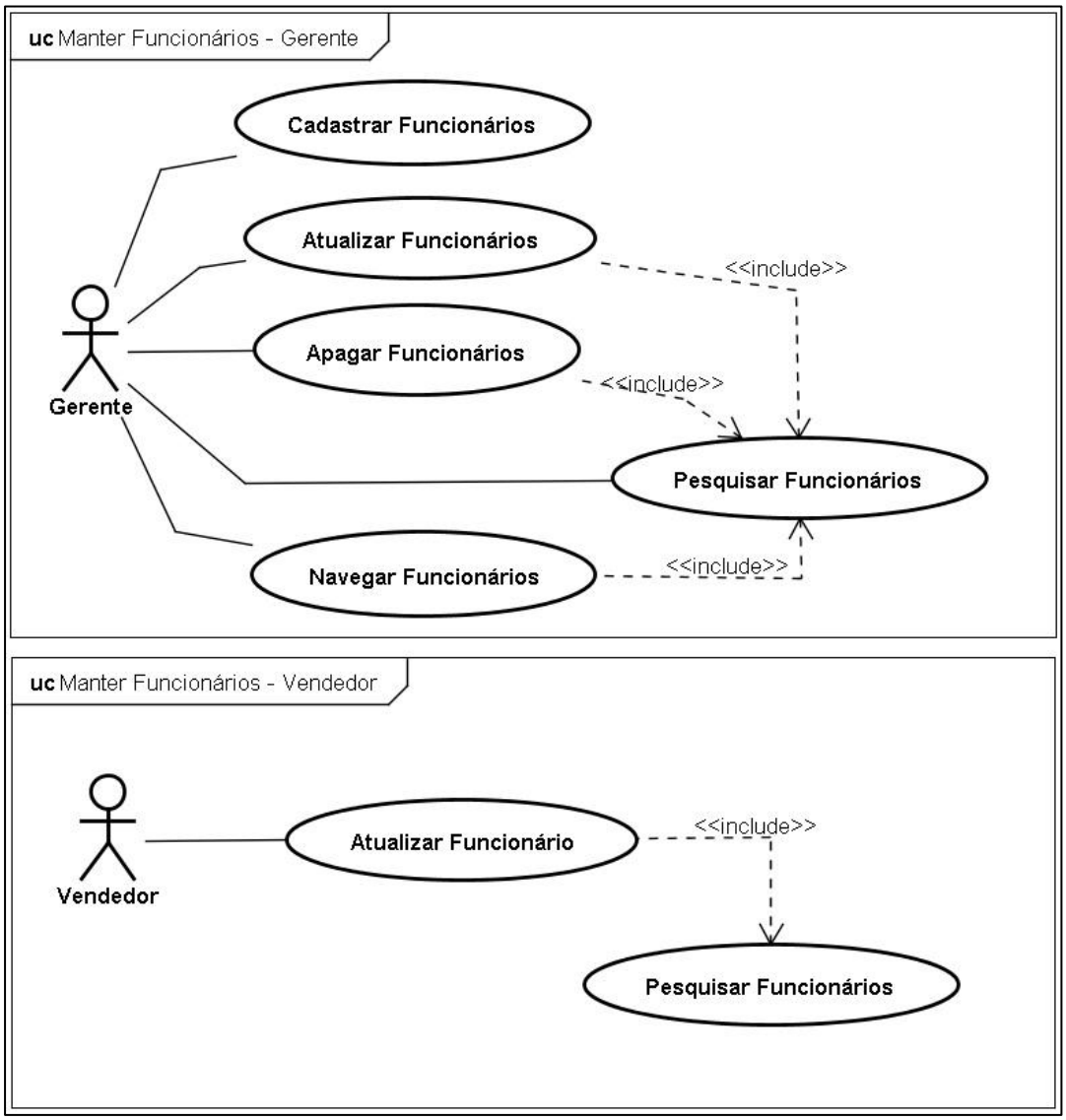

Figura 23 - Casos de uso Manter Funcionários para Gerente e Vendedor.
Fonte: Autoria Própria.

A Tela de Funcionários apresenta as funções contidas nos casos de uso da figura 23. Assim como na Tela de Clientes o usuário pode navegar, cadastrar, alterar, apagar e pesquisar, no entanto agora com dados do funcionário.

O processo de navegação funciona da mesma maneira, empregando o uso de botões de anterior e próximo. Os dados são pegos e atribuídos a uma lista, que posteriormente é percorrida para enviar seus dados para as variáveis correspondentes da tela, obtendo assim o índice do funcionário, que é armazenado em uma variável, para a realização das funções de alteração e exclusão. A

navegação não é utilizada pelo Vendedor nesta tela, visto que ele só pode editar seus próprios dados. Quando o sistema percebe que o usuário é um Vendedor suas informações são carregadas durante a inicialização da tela.

A função de cadastro insere os dados do funcionário, dependendo do estado de conectividade, na base de dados, mas antes algumas validações devem ser feitas a fim de manter a integridade do banco. A primeira verificação controla para que os campos informados não sejam nulos, sendo necessário pelo menos o usuário informar o nome do funcionário para realizar o cadastro. Outra verificação, e mais importante, valida o nome informado pelo usuário, procurando por todo o banco de dados nomes de funcionários iguais ao do recém informado, e ao constatar que os nomes são iguais o sistema apresenta uma mensagem não deixando o usuário inserir o dado no banco.

Ambos os usuários abrangem a função de alteração, entretanto o Gerente pode alterar qualquer usuário do sistema, enquanto o Vendedor possui apenas o poder para edição das informações pessoais. A alteração usa as mesmas validações presentes na função de cadastro, com uma ressalva, a validação de nomes iguais é ligeiramente diferente. O usuário ao editar seus dados pode perfeitamente deixar seu nome igual, sendo incorreta a afirmação anterior de não permitir nomes iguais, pois ao alterar não seria um novo registro, consistindo apenas em uma edição do mesmo dado, ou seja, na alteração é possível usar o mesmo nome desde que o registro seja o mesmo, com o mesmo índice.

A exclusão, como na maioria das vezes neste sistema, fica a cargo do Gerente. A função de exclusão utiliza o índice do registro atual do funcionário para realizar a exclusão, porém antes de apagar os dados, o sistema garante não apagar um funcionário que efetuou uma transação com outra tabela, além de também não apagar funcionário do tipo Gerente, para não gerar perda de funções e afetar a hierarquia do sistema.

A função de pesquisa utiliza a vista *AutoCompleteTextView* para facilitar a procura de funcionários pelo usuário do sistema, onde são informados os caracteres e a *View* apresenta uma lista de sugestões de funcionários, a partir do momento em que encontra a letra dentro a palavra, apenas sendo necessária a seleção do funcionário desejado.

3.4.5 Manter Fornecedores

Até o presente momento somente os usuários funcionários tiveram acesso ao sistema, contudo o fornecedor terá acesso e será essencial para a Tela de Fornecedores e para a Tela de Produtos, próxima tela a ser expandida. A Tela de Fornecedores tem o *layout* também parecido com as telas anteriores, contendo as *ViewGroups LinearLayout*, *ScrollView* e *TableLayout*, e as vistas *TextView*, *EditText*, *AutoCompleteTextView* e *Button*. A figura 24 expõe o *layout* da Tela de Fornecedores.

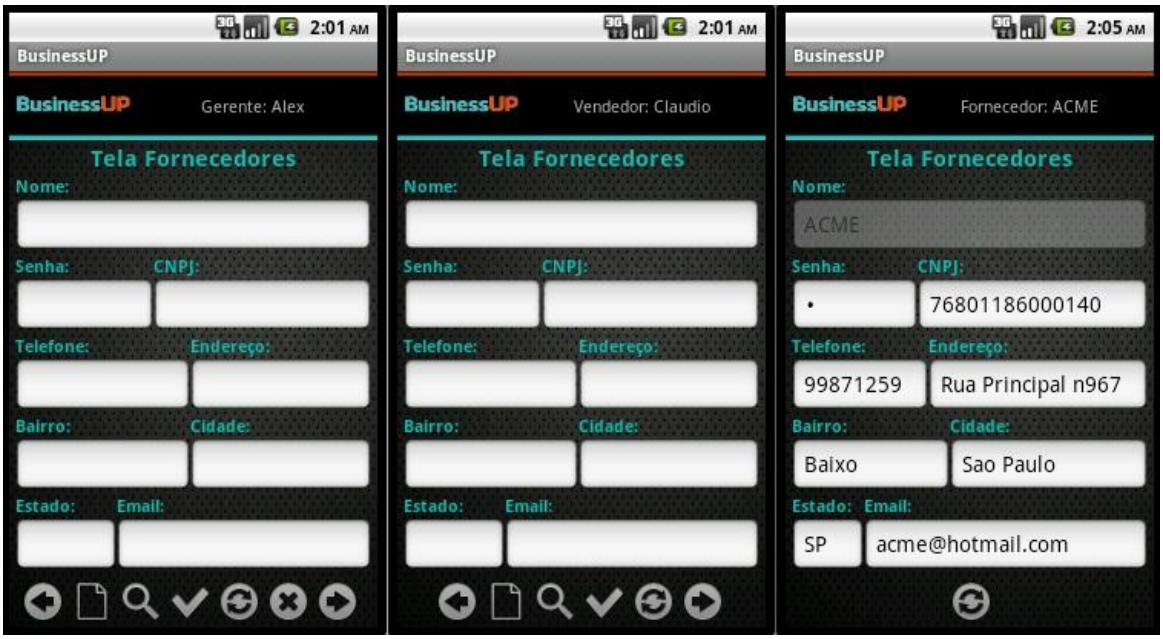

Figura 24 - Telas de Fornecedores mostrando níveis de acesso.
Fonte: Autoria Própria.

Como informado anteriormente a Tela de Fornecedores pode ser acessada pelo Gerente, Vendedor e Fornecedor. A poder do Fornecedor nesta tela é idêntico ao do Vendedor na Tela de Funcionários, em que a única função disponível é a de alteração dos próprios dados. A maioria das funções da Tela de Fornecedores pode ser realizada tanto pelo Gerente quanto pelo Vendedor, tirando a função de exclusão de fornecedores, que é específica do Gerente.

A figura 25 apresenta três casos de uso referentes às funcionalidades da Tela de Fornecedores, onde o usuário pode navegar, cadastrar, alterar, apagar e pesquisar os dados do fornecedor, dependendo do nível de acesso do usuário.

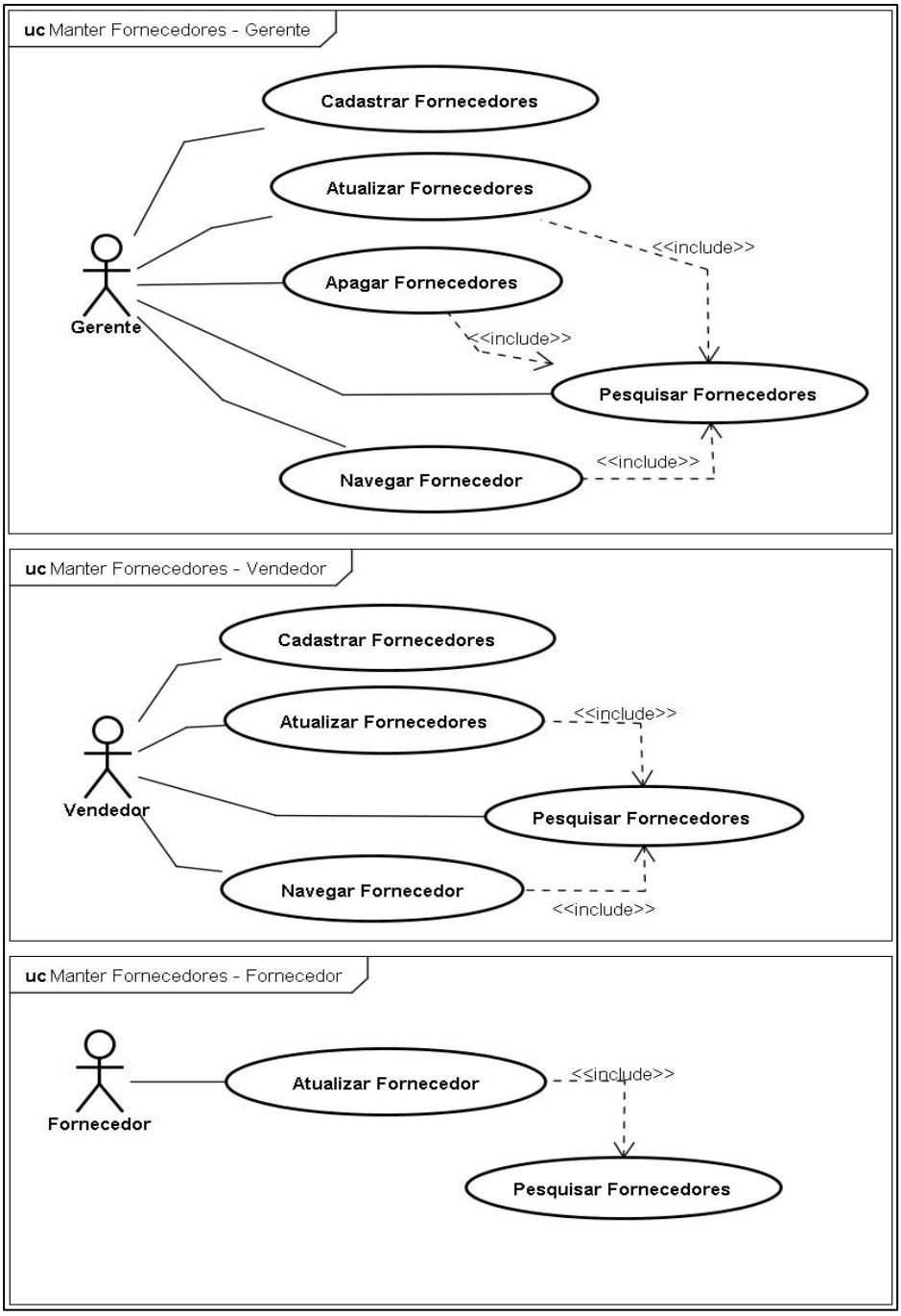

Figura 25 - Casos de uso Manter Fornecedores para Gerente, Vendedor e Fornecedor.

Fonte: Autoria Própria.

A função de navegação é igual em todo sistema e nesta tela possibilita o usuário percorrer todos os dados do fornecedor através dos botões de anterior e próximo, armazenando o índice do fornecedor em uma variável utilizada para executar as funções de alteração e exclusão. O Fornecedor não utiliza a navegação, já que não tem permissão de acesso aos dados de outros fornecedores.

Para inserir alguma informação na base de dados a função de cadastro é utilizada, porém para preservar a integridade do banco certas validações são necessárias. Verificar os dados informados pelo usuário é interessante, para evitar que campos vazios ocupem o lugar de informações valiosas. É obrigatório informar ao menos o nome para o cadastro do fornecedor, pois os outros campos podem ser editados mais tarde. Igualmente importante é a verificação do nome de fornecedor, que de nenhuma maneira deve ser igual à de outros fornecedores presentes no sistema.

Os três usuários, Gerente, Vendedor e Fornecedor, tem acesso à função de alteração de dados do fornecedor. O Fornecedor possui a capacidade para editar suas informações, enquanto os outros dois usuários podem efetuar as edições que acharem necessárias. Como esta função altera informações do banco de dados, é indispensável realizar validações para a manutenção dos dados. As validações são as mesmas utilizadas na função de cadastro de fornecedor, sendo levemente diferente com relação à afirmação de não permitir nomes iguais, onde o usuário pode usar nomes iguais, porém somente quando for editar o mesmo registro.

A função de exclusão, realizada pelo Gerente, emprega o índice do registro, armazenado anteriormente em uma variável, para apagar os dados do fornecedor, no entanto, antes de realizar esta tarefa o sistema se assegura de não apagar um fornecedor que tem um vínculo com outra tabela, que no caso dos fornecedores é possuir um produto.

O processo de pesquisa, semelhante com os mostrados anteriormente, tem o objetivo de facilitar a função de pesquisa de fornecedores para o usuário do sistema, por meio do uso da vista *AutoCompleteTextView*, que apresenta uma lista de sugestões de fornecedores quando o usuário informa um caractere, tornando simples a tarefa de procura.

3.4.6 Manter Produtos

A Tela de Produtos, uma das opções do menu principal, sofre grande influência do usuário do tipo Fornecedor, porem não é o único que tem acesso a esta tela. A Tela de Produtos, apresenta um *layout* com *ViewGroups LinearLayout*, *ScrollView* e *TableLayout*, e as vistas *TextView*, *EditText*, *AutoCompleteTextView* e *Button*. A figura 26 mostra o *layout* da Tela de Produtos.

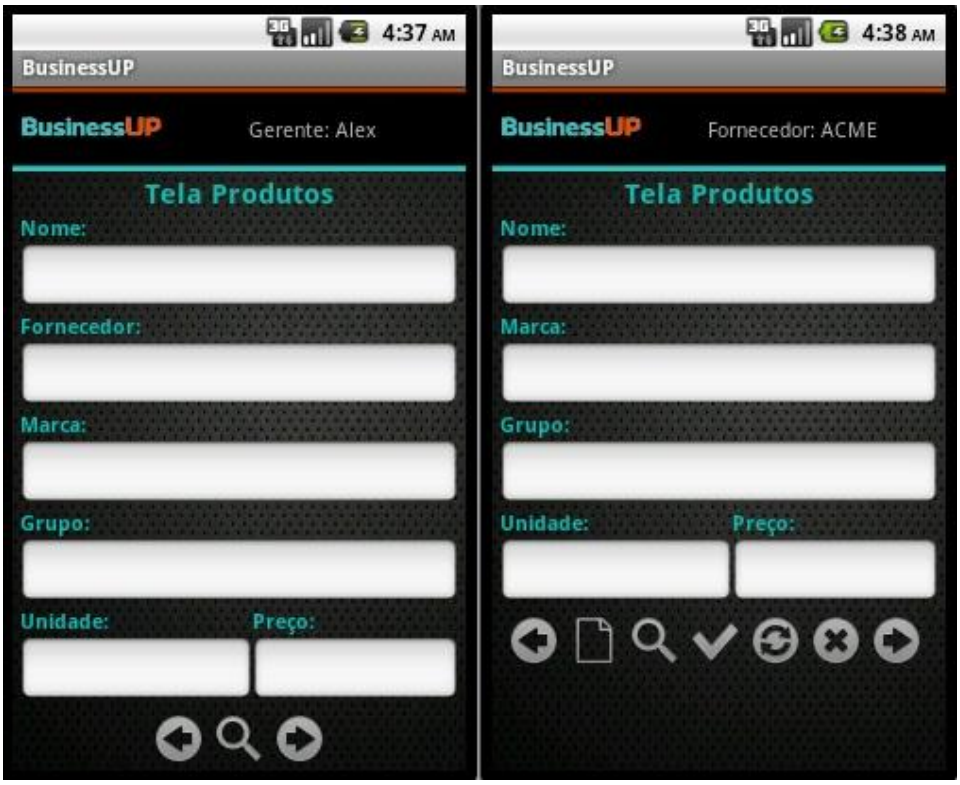

Figura 26 - Telas de Produtos mostrando níveis de acesso.
Fonte: Autoria Própria.

A figura 26 mostra que a Tela de Produtos é utilizada principalmente pelo Fornecedor, sendo ele a executar a maioria das funções em relação ao produto. É importante perceber na mesma figura que o Fornecedor recebe o privilégio de manipular as informações do produto, contudo somente as informações ligadas a ele. Por exemplo, o Fornecedor tem a permissão de inserir um produto no banco de dados, porém este produto pertence somente a ele, permanecendo ligado apenas com o fornecedor que o cadastrou, desta maneira o Fornecedor não pode observar ou manipular produtos que não pertençam a ele. Os outros dois funcionários, Gerente e Vendedor, podem apenas pesquisar e navegar, contudo são exibidos todos os produtos sem se importar com o fornecedor.

A Tela de Produtos contêm as funções que são mostradas nos casos de uso da figura 27. Dependendo do nível de acesso do usuário ele pode navegar, cadastrar, alterar, apagar e pesquisar, em relação aos dados do produto.

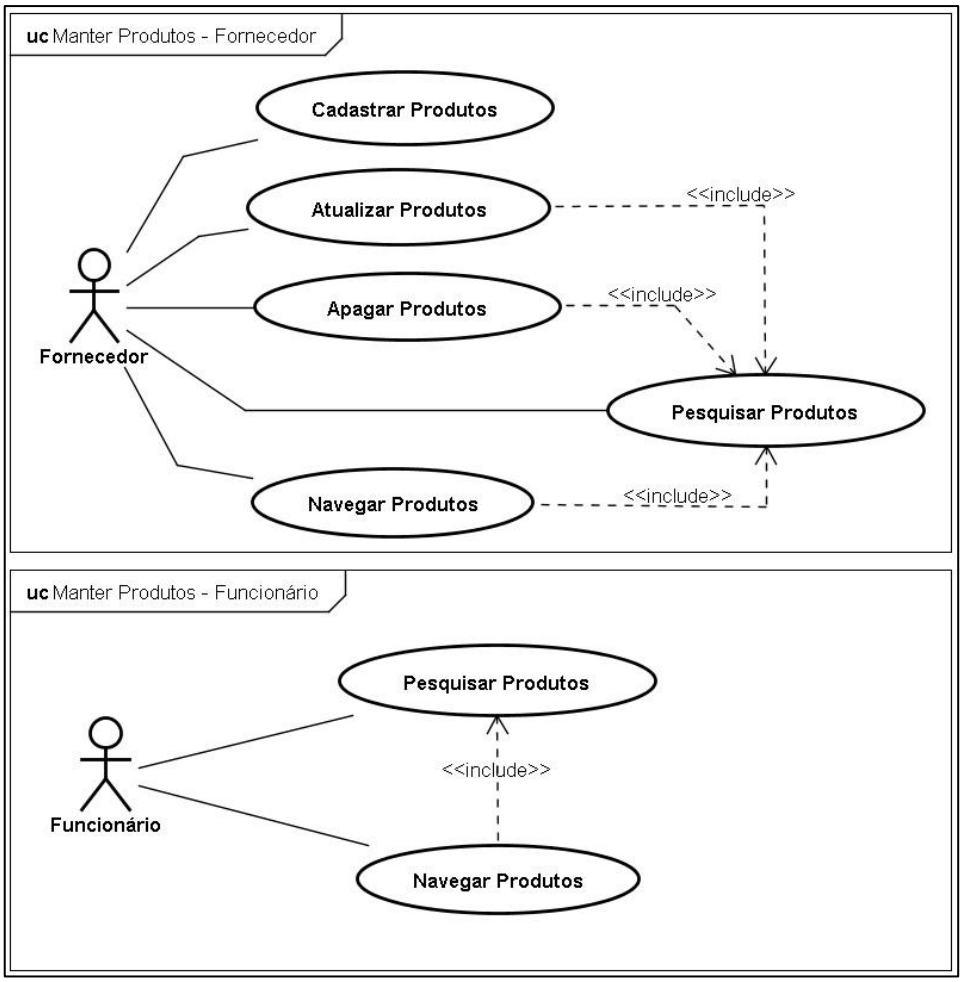

Figura 27 - Casos de uso Manter Produtos para Fornecedor e Funcionário (Gerente e Vendedor).

Fonte: Autoria Própria.

O processo de navegação tem alguns pontos diferentes em relação às demais funções de navegar. Ainda emprega o uso de botões de anterior e próximo para percorrer os dados do produto contidos em uma lista, enviando os dados para os campos correspondentes na tela, obtendo o índice do produto para poder alterar e apagar os dados do produto. O diferente é que existem dois métodos de navegação, um para o Fornecedor que não pode navegar entre todos os produtos, somente os com que têm um vinculo, e outro para os funcionários que podem navegar por qualquer produto.

A função de cadastro, presente para o Fornecedor, insere os informações do produto na base de dados após realizar algumas verificações. A primeira verificação confere se todos os campos da tela foram informados corretamente, sem espaços vazio ou caracteres no lugar de números. Outra verificação é em relação ao

produto já ter sido cadastrado por aquele Fornecedor, onde o nome do produto não pode ser igual quando o fornecedor for idêntico, ou seja, é impossível inserir um produto que já exista desde que seja o mesmo fornecedor.

Antes de realizar a edição dos dados do produto também são realizadas validações, semelhantes à função de cadastro. Verificar se todos os campos foram completados corretamente e se o usuário informou qualquer nome de produto que já exista. Obviamente, ao alterar os dados o nome existirá, devendo ser verificado se será usado o mesmo nome do banco durante a edição do próprio registro, com o próprio índice, podendo assim ocorrer à atualização.

A função de apagar utiliza o índice do registro atual do produto para realizar a exclusão dos dados, mas antes verifica se o produto não tem um vínculo com a realização do pedido, estando presente na tabela de itens de pedido.

A função de pesquisa também utiliza a vista *AutoCompleteTextView* para auxiliar o usuário durante a procura de produtos pelo sistema, onde são informados os caracteres e logo após é apresentada uma lista de sugestões de produtos. A pesquisa, assim como a navegação, apresenta dois métodos para a procura de produtos. Os funcionários podem pesquisar todos os produtos que desejarem, enquanto o Fornecedor só pode pesquisar os produtos que são manipulados por ele, os que apresentam uma conexão com o Fornecedor, como exibe a figura 28.

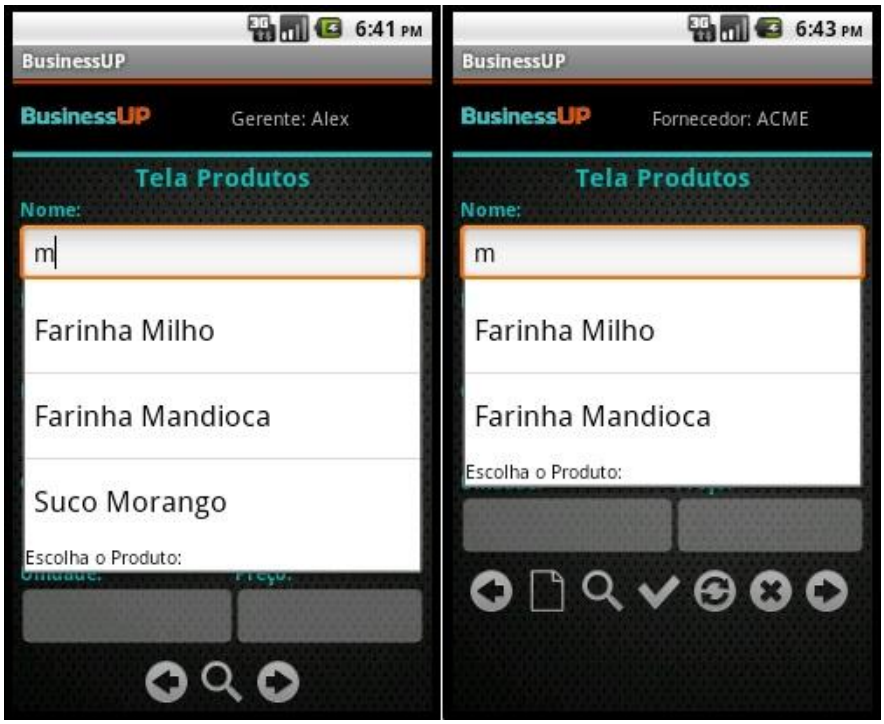

Figura 28 - Telas de Produtos função de pesquisa.

Fonte: Autoria Própria.

3.4.7 Manter Pedidos

A Tela de Pedidos, a principal da aplicação, fornece ao Gerente e Vendedor a função de realizar pedidos. Esta tela tem um *layout* simples com apenas a *ViewGroup LinearLayout* sendo utilizada, já que usa a vista *ListView*, não sendo aconselhável utilizar a *ScrollView*. As vistas *TextView*, *EditText*, *AutoCompleteTextView* e *Button* também são utilizadas, como mostra a figura 29.

Figura 29 - Tela de Pedidos.

Fonte: Autoria Própria.

É importante notar que a lista de produtos apresentada na imagem 29 é personalizada para acomodar o nome do produto e seu preço, além dos campos para informar a quantidade de tal produto e também selecioná-lo para a realização do pedido. Para criar um *ListView* customizado é necessário criar um modelo de objeto, uma classe, que irá conter todos os dados da lista, e ainda criar um novo *Adapter*, que herdará métodos do *ArrayAdapter*, especificando os objetos da classe modelo. Desta maneira, os dados mostrados na lista de produtos vêm de um novo *Adapter* que controla a matriz de dados.

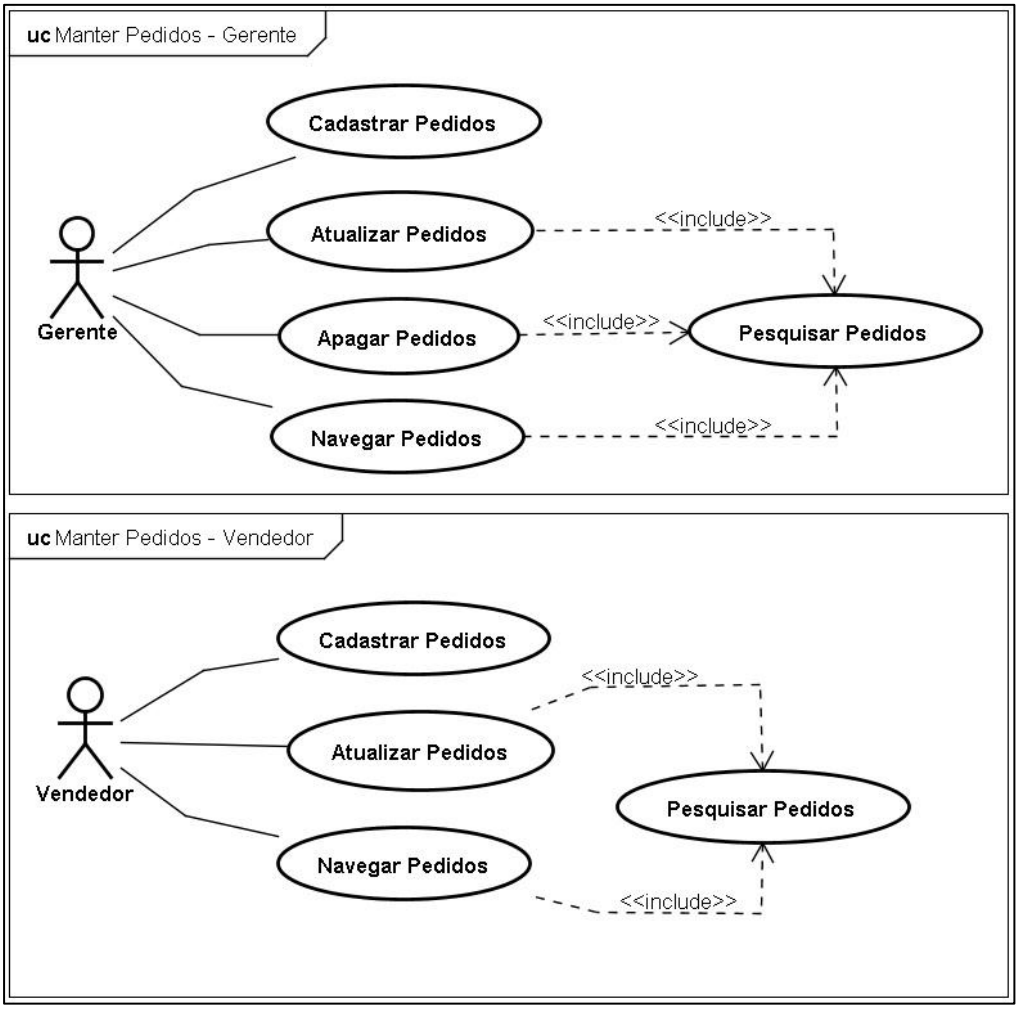

Figura 30 - Casos de uso Manter Pedidos para Gerente e Vendedor.
Fonte: Autoria Própria.

A Tela de Pedidos difere das anteriores por não apresentar função de pesquisa, já que clientes e funcionários podem realizar “infinitos” pedidos, impossibilitando o modelo de processo de filtragem adotado no restante das telas. As demais funções continuam presentes, como mostram os casos de uso da figura 30, o usuário, dependendo do nível de acesso, pode navegar, cadastrar, alterar, apagar os pedidos.

A função de navegar é semelhante com a adotada na Tela de Produtos, pois apresentam dois métodos de navegação, um para o Gerente, que pode visualizar as informações de todos os pedidos, e o Vendedor, que tem acesso apenas aos pedidos que ele mesmo realizou. As outras características da navegação continuam iguais, empregando o uso de botões de anterior e próximo

para navegar entre os pedidos contidos em uma lista e obtendo o índice do pedido para posterior alteração e exclusão os dados do pedido.

Para informar os nomes do cliente ou funcionário durante a realização do pedido é utilizada a vista *AutoCompleteTextView* para auxiliar o usuário durante a procura pelo cliente ou funcionário, além de agilizar o processo de cadastro. Com esta nova característica é necessário validar os dados antes de inserir na base de dados. A única verificação efetuada é para constatar se o cliente e/ou funcionário informado existe no banco de dados. Logicamente a vista *AutoCompleteTextView* traz a sugestão contendo o cliente ou funcionário, contudo o usuário pode não acatar essa proposta mostrada, informando qualquer outro dado ou deixando o campo vazio, mostrando a necessidade desta validação. É importante salientar que o pedido pode ser realizado mesmo sem nenhum produto informado, visto que o banco de dados sempre grava a data da realização do pedido, sem alterá-la quando é efetuada a atualização.

Da mesma maneira do cadastro, quando o usuário for realizar a função de alteração as informações também devem ser validadas. Após a validação todos os dados presentes na tela são atualizados, inclusive os itens de pedidos, ou produtos, que são primeiramente apagados e depois inseridos novamente para não gerar falhas no banco de dados.

A função de deleção, presente somente para o Gerente, possibilita apagar qualquer pedido utilizando o índice do registro atual do pedido. Outra função modesta, porém interessante é a de calcular o total do pedido. Esta função percorre toda a lista de produtos e observa qual está marcado, o qual estiver é adquirida a quantidade e então realizada a somatória para obter total do pedido.

3.4.8 Exibir Gráficos

A Tela de exibição de gráficos constrói gráficos em tempo real utilizando o *plug-in AChartEngine Release* 0.7.0. O *AChartEngine* é uma biblioteca de gráficos para aplicativos Android e suporta vários tipos de gráfico como gráfico de linha, gráfico de área, gráfico de dispersão, gráfico de tempo, gráfico de barras, gráfico de pizza, gráfico de bolhas, entre diversos outros tipos e até combinações.

Os gráficos podem ser construídos como uma *View* que pode ser adicionada a um *ViewGroup* ou como uma intenção, além da possibilidade de ser

usado para iniciar uma atividade. O modelo e o código são otimizados para manipular e apresentar um grande número de informações.

Os modelos de gráficos escolhidos para o aplicativo foram o gráfico de pizza e gráfico de linha, por apresenta as informações de maneira simples e objetiva. A Tela de Gráficos disponibiliza quatro tipos de gráficos com as seguintes informações: produtos mais pedidos, clientes com mais pedidos, funcionários com mais pedidos e número de pedidos por mês. A figura 31 mostra o gráfico de pizza representado os produtos mais pedidos do sistema.

Figura 31 - Telas de Gráficos - Produtos mais pedidos.

Fonte: Autoria Própria.

O gráfico apresentado na figura 31 exibe todos os produtos que estão presentes em um pedido. Ocorre uma seleção no banco de dados através do *Web service*, a informação selecionada é enviada para uma lista que é percorrida e adicionada no gráfico por meio do método *add*() da classe *CategorySeries*, como mostra o pseudocódigo abaixo. O método *add*() adiciona o nome do produto e quantidade de vezes que é pedido no gráfico.

```
1     for (int i = 0; i < produtoList.size(); i++) {
2           categorySeries.add(nomeProd, quantidadeProd));
3     }
```

Pseudocódigo 7 – Adição de informações ao gráfico.

Sempre que o gráfico é renderizado e mostrado na tela ele apresenta novas cores, porque não há maneira de saber quantas cores serão necessárias para a montagem do gráfico. Então é usado o número do tamanho da lista, para saber a quantidade de cores, e posteriormente a classe *Random* juntamente com a classe *Color*, para gerar cores aleatórias para o gráfico de pizza. O pseudocódigo a seguir mostra a realização desta função.

```
1      int colors[] = new int[tamanhoLista];
2      for (int i = 0; i < tamanhoLista; i++) {
3            Random r = new Random();
4            colors[i] = Color.rgb(r.nextInt(256),
5                        r.nextInt(256),
6                        r.nextInt(256));
7      }
```

Pseudocódigo 8 – Aplicação de cores aleatórias no gráfico.

Outra funcionalidade interessante do *AChartEngine* é que o gráfico não fica com informações ilegíveis na tela, com o método *setZoomEnabled(true)* é habilitado os comandos de *zoom* para aproximar ou afastar do gráfico, além de possuir a função de mover o gráfico para qualquer local, por ser sensível ao toque.

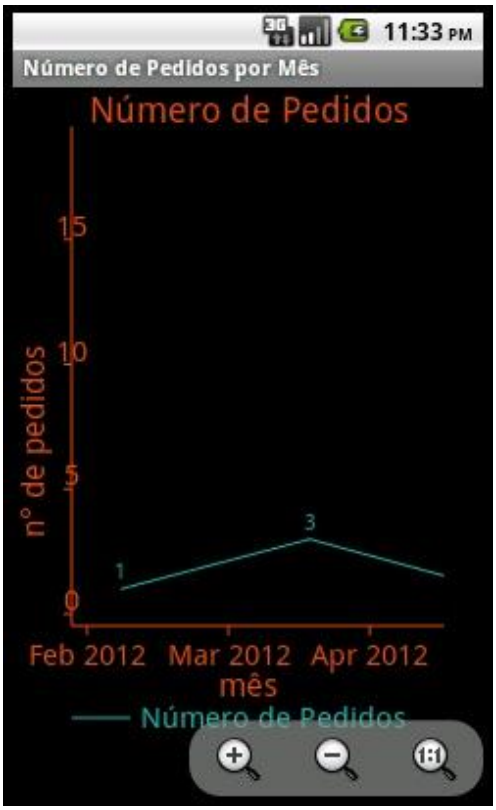

Figura 32 - Telas de Gráficos – Número de pedidos por mês.

Fonte: Autoria Própria.

O gráfico de linha da figura 32 apresenta o número de pedidos durante um período de tempo. Assim como no gráfico anterior os dados estão inclusos em uma lista, que contem o número de pedidos separado por mês e ano. O gráfico pode ser personalizado para definir o tamanho e cor da fonte do titulo do gráfico, legenda, titulo do eixo x e y, cor do eixo x e y, cor e espessura da linha, tipo do ponto de ligação da linha, entre outras opções.

No gráfico o eixo y representa o número de pedidos e o eixo x representa o mês e ano, apresentando uma linha do tempo “infinita”, pois enquanto houverem números de pedido, a o eixo x continuará a apresentar o mês e ano correspondentes. Desta vez as cores foram definidas no código-fonte da aplicação, por não ter necessidade de colorir a linha com cores diferentes, como acontecia no gráfico de pizza. As funções de *zoom* e movimentação do gráfico ainda estão presentes. É importante salientar que para exibir os gráficos é imperativo que o usuário tenha conexão com a internet, para não gerar gráfico com informações desatualizadas.

3.4.9 Emitir Relatórios

A Tela de Relatórios apresenta um design simples, com uma vista *Spinner* contendo todos os relatórios que o usuário pode fazer e um campo para informar o nome do arquivo que será criado, além do botão responsável por gerar o relatório. A figura 33 mostra a Tela de Relatórios.

Figura 33 - Tela de Relatórios.

Fonte: Autoria Própria.

As opções de emissão de relatórios são básicas, sendo feito apenas os relatórios principais em uma aplicação, que apresentam todos os clientes, funcionários, fornecedores, produtos e pedidos contidos no banco de dados.

A criação de relatórios é feita com o *plug-in iText* 5.2.0. O *iText* é uma biblioteca Java que permite a criação e manipulação de documentos PDF. O documento criado pelo *iText* apresenta uma estrutura hierárquica, com palavras, com fontes predefinidas, que formam frases permitindo definir o espaçamento entre linhas e ainda parágrafos que permitem diversos atributos de layout.

```
1     Font titulo = FontFactory.getFont("Arial", 16, Font.BOLD);
2     Font paragrafo = FontFactory.getFont("Arial", 12, Font.BOLD);
3     Font corpo = FontFactory.getFont("Arial", 10, Font.NORMAL);
4
5     Document documento = new Document();
6
7     File pastaTeste = new File("/sdcard/BusinessUP/");
8     pastaTeste.mkdirs();
9     File outputFile = new File(pastaTeste, "arquivo.pdf");
10    PdfWriter.getInstance(documento, new FileOutputStream(outputFile));
11    documento.open();
12    documento.addAuthor("Autor");
13
14    Paragraph texto = new Paragraph("bla bla bla...", corpo);
15    texto.setAlignment(Element.ALIGN_JUSTIFIED);
16    documento.add(texto);
17
18    documento.close();
```

Pseudocódigo 9 – Geração de documento PDF.

Para realizar um relatório primeiramente é necessário informar o tipo, número e espessura da fonte que será usada no novo documento, à classe *Font*, linha 1, é utilizada para este proposito. Para criar um novo documento é usada a classe *Document*, linha 5, que é instanciada e utilizada para “mostrar a cara” do documento. Entretanto antes é necessária a criação de dois objetos da classe *File*, utilizada na leitura e gravação de arquivos no sistema. O primeiro objeto, linha 7, é usado para representar o caminho em que o arquivo será gravado, neste caso no cartão de memória do dispositivo, utilizando o método *mkdirs*() para criar a estrutura do diretório, se necessário. Enquanto o segundo objeto, linha 9, é utilizado para produzir o arquivo com extensão PDF.

O *iText* então cria o documento PDF utilizando a instância da classe *Document* e o arquivo produzido anteriormente. O documento pode ser manipulado logo que é aberto, com método *open*(), tendo diversas funções como definir o

tamanho da pagina, inserir plano de fundo, adicionar novos parágrafos, adicionar paginas, adicionar autor, adicionar algum texto com uma fonte determinada anteriormente, alinhar o texto, adicionar imagem, e até mesmo utilizar encriptação de dados. É importante lembrar-se de fechar o documento após realizar todas as operações com o método *close*().

Depois da criação do relatório é necessário realizar a leitura deste. A classe File é novamente utilizada para procurar o arquivo a partir do caminho e nome do arquivo criado anteriormente. A leitura do arquivo PDF é feita por qualquer leitor de PDF presente no dispositivo, como por exemplo, o Adobe Reader. Para ler o relatório um objeto *Intent* é criado passando a constante de ação e dados, o *ACTION_VIEW* corresponde à ação de chamar um leitor de PDF, enquanto os dados vem do arquivo PDF, na forma de instancia da classe Uri. O pseudocódigo a seguir mostra como o processo de leitura é realizado.

```
1      File file = new File("/sdcard/BusinessUP/" + nomeArquivo + ".pdf");
2
3      Intent intent = new Intent();
4      intent.setAction(Intent.ACTION_VIEW);
5      Uri uri = Uri.fromFile(file);
6      intent.setDataAndType(uri, "application/pdf");
7      startActivity(intent);
```

Pseudocódigo 10 – Leitura de documento PDF.

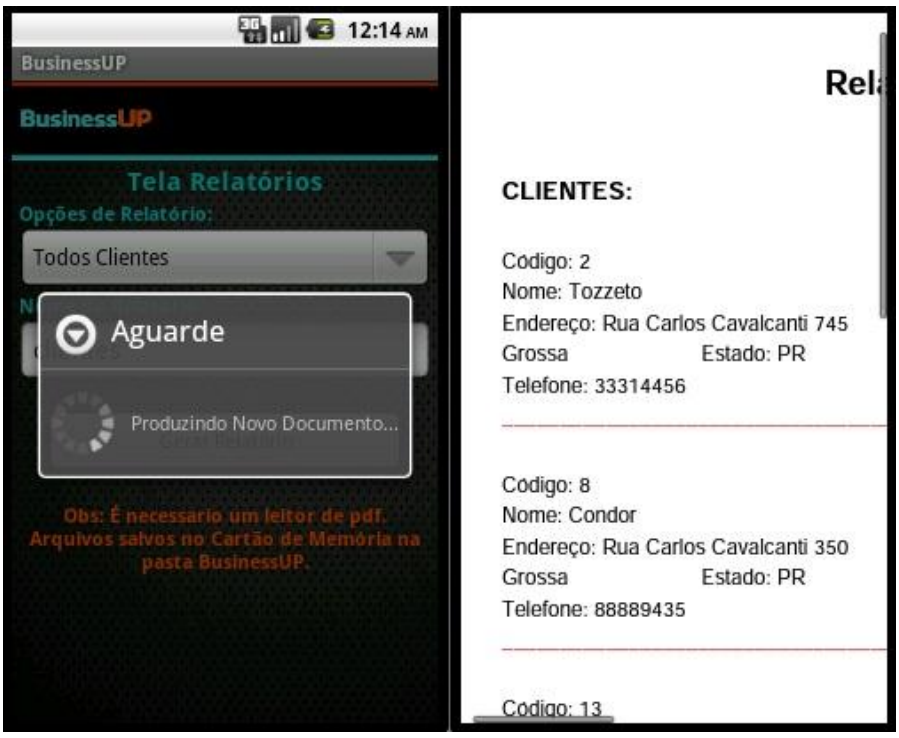

Figura 34 - Tela de Relatórios e o documento criado

Fonte: Autoria Própria.

A nova atividade é chamada e mostrada na tela para satisfazer a requisição feita no pseudocódigo anterior. A figura 34 apresenta o relatório, criado e lido, no leitor de PDF Adobe Reader.

3.4.10 Acessar Mapa

O último item da tela do menu principal é o acesso ao Mapa que usa a API do Google para que isto seja possível. No momento em que é acessada esta tela, o dispositivo captura as coordenadas de GPS do aparelho para exibir sua localização atual, também realiza a conversão dos endereços cadastrados na tabela de participantes em pontos de localização (*GeoPoint*) aplicando assim a geocodificação inversa, isto é, convertendo um endereço previamente cadastrado de um cliente ou fornecedor em latitude e longitude para que seja exibido um ponto no mapa a fim de localizar o indivíduo, confirmando que ali está um cliente ou fornecedor.

Se por ventura, o funcionário encontrar problemas para encontrar o cliente ou fornecedor ao longo do percurso, este poderá visualizar o mapa e saber onde ele se encontra no momento, evitando atrasos e outras confusões por não localizar o endereço de destino. Além dessas informações, o mapa possui um menu interno, permitindo que o usuário possa alternar a visualização do mapa em modo satélite, tráfico e *StreetView*, além de visualizar a rota percorrida por determinado funcionário durante aquele dia. A figura 35 exibe o mapa e o menu interno contendo as funções apresentadas anteriormente.

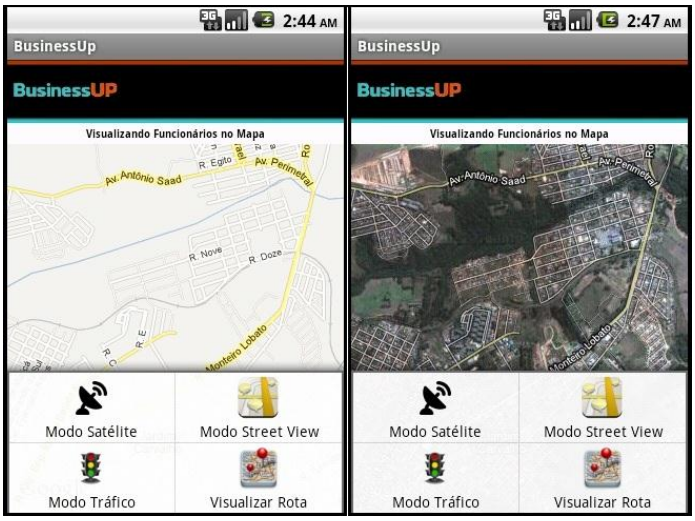

Figura 35 – Modos de visualização do mapa.

Fonte: Autoria Própria.

A visualização da rota percorrida pelo funcionário é interessante em casos onde o funcionário possui alguma dificuldade e necessita sanar o mais breve possível localizando outro funcionário mais próximo, conhecendo o caminho percorrido pelo mesmo, fica mais fácil descobrir de onde ele veio e para onde ele irá. Ou ainda um funcionário do tipo Gerente que gostaria de verificar se o funcionário está no lugar correto, desempenhando a função que era para ser realizada. Além de visualizar a quantidade de quilômetros percorridos até o momento e quais foram os pontos em que ele enviou a coordenada, bem como o horário de envio de cada coordenada.

É importante ter em vista que este serviço é somente oferecido para usuários que dispõem de acesso à internet, adquirindo informações atualizadas por meio do *Web service* em tempo real para o funcionário, bem como adquirir os pontos de latitude e longitude enviados por todos os funcionários para traçar a rota e desenhá-la no mapa. A figura 36 apresenta o mapa e a rota traçada por determinado funcionário.

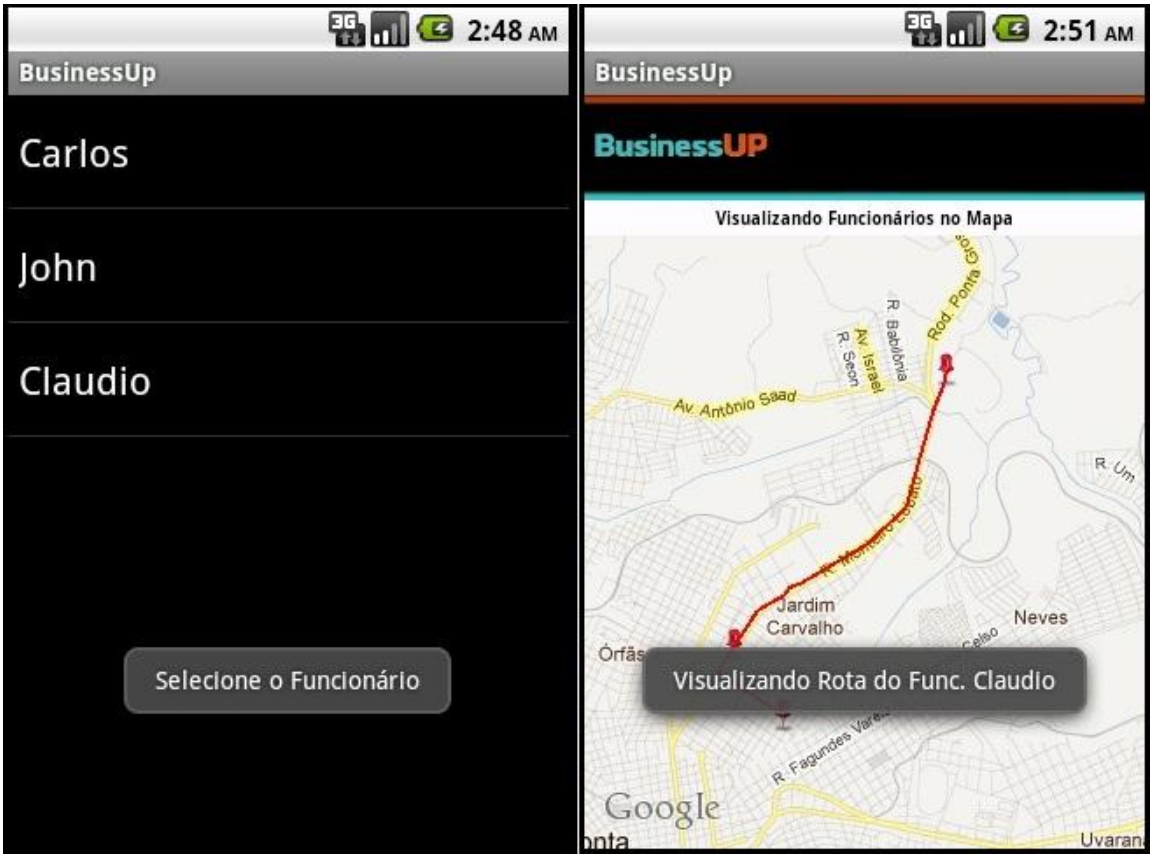

Figura 36 - Rota percorrida por determinado funcionário.

Fonte: Autoria Própria.

A cada ponto de latitude e longitude enviado pelo funcionário e presente no mapa, ao ser pressionado, apresenta algumas informações referentes à quilometragem percorrida até aquele ponto, horário do envio e nome do funcionário, exibidos na figura a seguir.

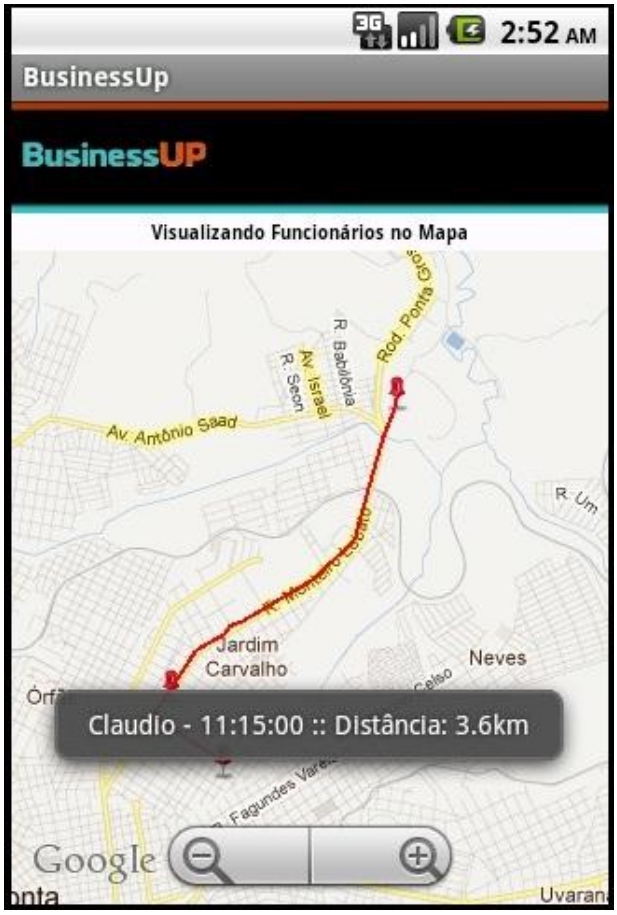

Figura 37 - Informações referentes ao ponto pressionado.

Fonte: Autoria Própria.

O pseudocódigo a seguir mostra como é feita a comunicação com o serviço da Google, *Google Maps*, passando por parâmetro quatro pontos, latitude e longitude de duas localizações (de onde e para onde), para que seja traçada a rota entre estes dois pontos, sendo possível obter todas as informações da figura anterior a partir disto.

```
1     public static String getUrlMapaGoogle(double fromLat, double fromLon,
2     double toLat, double toLon) {
3           StringBuffer url = new StringBuffer();
4           url.append("http://maps.google.com/maps?f=d&hl=en");
5           url.append("&saddr=");
6           url.append(Double.toString(fromLat));
```

```
7           url.append(",");
8           url.append(Double.toString(fromLon));
9           url.append("&daddr=");
10          url.append(Double.toString(toLat));
11          url.append(",");
12          url.append(Double.toString(toLon));
13          url.append("&ie=UTF8&0&om=0&output=kml");
14          return url.toString();
15      }
```

Pseudocódigo 11 – Traço de rotas entre dois pontos.

Por conseguinte, é filtrada a informação retornada para que finalmente seja desenhado para o usuário do sistema o caminho percorrido e assim pegar o próximo ponto continuando a rota até o último ponto enviado pelo usuário.

3.4.10.1 Serviço

Como já escrito anteriormente, serviços são executados em segundo plano e têm como objetivo realizar algum processamento de tempos em tempos sem interromper a atividade do usuário ou fazer com que o mesmo tenha que clicar em algo ou até mesmo ficar sabendo de tal ação.

O serviço presente no projeto inicia automaticamente quando o funcionário realiza login no sistema, tendo como principal objetivo enviar as coordenadas do dispositivo móvel do funcionário para o *Web service*, armazenando as informações no banco de dados, podendo assim, mais tarde traçar a rota deste funcionário no mapa. Com o serviço rodando, a cada cinco minutos é verificado se o usuário possui conexão com a internet, e se tiver, enviará as coordenadas de latitude e longitude do GPS para a base de dados. Caso o funcionário esteja sem conexão durante vinte e cinco minutos, ou seja, falha no envio durante cinco tentativas, o aparelho irá vibrar durante um segundo e logo após aparecerá na barra de notificação do SO Android uma mensagem informando sobre o ocorrido, requisitando ao usuário que o mesmo habilite a conexão com a internet para que em uma próxima oportunidade consiga rastrear a posição do dispositivo.

É importante frisar que em níveis gerenciais de hierarquia este tipo de serviço é extremamente importante devido ao fato de extrair informações mensais como, por exemplo, qual funcionário percorreu maior quilometragem em certo tempo, qual funcionário ficou mais tempo parado, quantas vezes o funcionário saiu da empresa, em que dias isto aconteceu etc..

A figura 38 apresenta a barra de notificação do Android, que ao ser pressionada, exibe a informação de que o aparelho não apresenta conexão com a internet. É essencial notar que para este exemplo, foram definidos apenas dez segundos para envio da localização.

Figura 38 - Barra de Notificação e mensagem de aparelho sem conexão.
Fonte: Autoria Própria.

4 CONCLUSÃO

Este projeto apresentou o estudo do Sistema Operacional Android, tecnologia recente, apresentando constantes atualizações tanto no SO como nas ferramentas de desenvolvedor, que vem procurando a estabilidade e seu crescimento constantemente no mercado, sendo utilizado por uma infinidade de pessoas, tornando-se vantajosa para desenvolvedores por inúmeros fatores.

O sistema de gerenciamento de pedidos foi um projeto que apresenta as funcionalidades do Android, procurando adicionar novas funcionalidades e características próprias para o sistema proposto.

As tecnologias, juntamente com as ferramentas de desenvolvimento e *plug-ins* estudados mostram como é possível à implementação de um aplicativo Android. Com o conhecimento anterior da linguagem Java e XML facilitam e agilizam consideravelmente a implementação do sistema.

O projeto de gerenciamento de pedidos foi desenvolvido para demonstrar algumas das capacidades do SO Android. A mobilidade uma de suas principais características, foi aplicada no uso do serviço de mapas da Google, (*Google Maps*).

Como resultado, o usuário do sistema tem acesso a funcionalidades como traçar rotas até determinado cliente presente na base de dados, onde são convertidos endereços em coordenadas de latitude e longitude, destacando o trajeto a ser realizado no mapa. Outra funcionalidade é o monitoramento do funcionário em tempo real por meio do GPS, possibilitando acompanhar o passo a passo do usuário que utiliza a aplicação.

Além do uso de mapas, a mobilidade possibilita o uso de sistemas remotos, como o *Web service*. No projeto estes serviços são responsáveis por diversas funções como inserção, alteração, deleção e vários tipos de seleção, para clientes, funcionários, fornecedores, produtos, pedidos, além de prover informações para geração e visualização de relatórios e gráficos.

Existe a possibilidade de ocorrer um problema nos casos anteriores, devido à necessidade constante de conexão com a internet. Sem esta, é provável que ocorra a perda de dados durante a realização dos procedimentos. A solução proposta faz uma cópia exata da base de dados do servidor, utilizando o banco de dados SQLite, que realiza as mesma funções do banco de dados tradicional, porém sem o uso da internet, armazenando os dados localmente dentro do dispositivo.

Entretanto surgiu outro problema, visto que existem duas bases de dados com informações possivelmente distintas. Este problema foi contornado realizando uma sincronização das bases de dados, onde são comparados e validados todos os dados de ambos os bancos, permitindo que sempre as duas bases tenham as mesmas informações. Por utilizar o *Web service* para comunicação com o servidor estes processos são geralmente lentos, dependendo do tipo de conexão utilizada pelo dispositivo.

Após a finalização do projeto pode-se concluir que os objetivos almejados foram alcançados, mostrando algumas das diversas funcionalidades da plataforma Android, assim como suas características e arquitetura. Durante o desenvolvimento do projeto, a linguagem Java foi constantemente utilizada, tanto na implementação do aplicativo Android como também na implementação do *Web service*, onde alguns pedaços de códigos-fonte foram disponibilizados para mostrar a sintaxe e ajudar futuros acadêmicos que por ventura utilizarão este trabalho em seus estudos.

4.1 TRABALHOS FUTUROS

O projeto realizado pode ser ampliado de modo a apresentar mais funcionalidades para auxiliar o usuário do sistema. Algumas sugestões como trabalhos futuros são exibidas a seguir:

- Otimização do aplicativo, já que o desenvolvimento e implementação não levou em consideração detalhes de desempenho em diferentes dispositivos móveis;
- Realizar teste e verificar as capacidades do banco de dados SQLite;
- Autorizar acesso dos clientes ao sistema, podendo em alguns casos poupar o deslocamento do funcionário até determinada localidade;
- Adicionar mais funcionalidades para o fornecedor;
- Disponibilizar o sistema em outras línguas;
- Dedicar mais trabalho ao *Web service*, deixando-o mais seguro e rápido.

REFERÊNCIAS

ACHARTENGINE. **Charting library for Android.** Disponível em <http://code.google.com/p/achartengine/>. Acesso em 30-04-2012, 2012.

ANDROID DEVELOPERS. Disponível em <http://developer.android.com/index.html>. Acesso em 20-01-2012, 2012.

BUSINESSWEEK. **Google Buys Android for Its Mobile Arsenal.** Disponível em <http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm>. Acesso em 23-08-2011.

GLASSFISHUSERFAQ. Disponível em < https://wikis.oracle.com/display/GlassFish/GlassFishUserFAQ>. Acesso em 22-04-2012, 2012.

GOMES, D. A. **Web Services SOAP em Java:** Guia Prático para o desenvolvimento de web services em Java. Novatec, 2010.

GONÇALVES, E. C. **SQLite, Muito Prazer!** Conheça o SQLite. Disponível em <http://www.devmedia.com.br/sqlite-muito-prazer/7100>. Acesso em 16-04-2012.

GOOGLE DEVELOPERS. **Obtaining a Maps API Key.** Disponível em <https://developers.google.com/maps/documentation/android/mapkey>. Acesso em 23-04-2012, 2012.

IBM *DEVELOPER WORKS*. **Introduction to Android development Using Eclipse and Android widgets.** Disponível em <http://www.ibm.com/developerworks/opensource/tutorials/os-eclipse-androidwidget/>. Acesso em 23-08-2011.

LEE, W. **Introdução Ao Desenvolvimento de Aplicativos para o Android.** Ciência Moderna, 2011.

LECHETA, R. R. **Google Android:** Aprenda a criar aplicações para dispositivos móveis com o Android SDK**.** Novatec, 2009.

MYERS, M. et al. **Global market segmentation for logistics services**. Industrial Marketing Management, 2003.

NETBEANS. **IDE Features**. Disponível em <http://netbeans.org/features/index.html>. Acesso em 20-04-2012, 2012.

ORACLE. **Object-Oriented Programming Concepts.** Disponível em <http://docs.oracle.com/javase/tutorial/java/concepts/index.html>. Acesso em 20-04-2012, 2012.

SQLITE. Disponível em <http://www.sqlite.org/>. Acesso em 05-03-2012, 2012.

W3C. **Web Services Glossary.** Disponível em <http://www.w3.org/TR/ws-gloss/>. Acesso em 16-04-2012, 2012.

W3SCHOOLS. **XML Tutorial**. Disponível em <http://www.w3schools.com/xml/default.asp>. Acesso em 12-04-2012, 2012.