UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS CURITIBA

ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

JULIO CESAR GONÇALVES

USO DA PLATAFORMA ANDROID EM UM PROTÓTIPO DE APLICATIVO COLETOR DE CONSUMO DE GÁS NATURAL

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA

2011

JULIO CESAR GONÇALVES

USO DA PLATAFORMA ANDROID EM UM PROTÓTIPO DE APLICATIVO COLETOR DE CONSUMO DE GÁS NATURAL

Monografia de especialização apresentada ao curso de Especialização em Tecnologia Java da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de especialista.

Orientador: Prof. Nelson Hideo Kashima
Co-orientador: Prof. Dr. João Alberto Fabro

CURITIBA
2011

AGRADECIMENTOS

A todos que direta ou indiretamente auxiliaram ou prestaram o seu apoio na realização deste trabalho.

RESUMO

O mercado de gás natural no Brasil acena com uma tendência de crescimento para os próximos anos e o segmento residencial, até então pouco explorado se comparado aos segmentos tradicionais de distribuição de gás natural como veicular e industrial, promete acompanhar esta tendência conforme apontam as expectativas de investimento das concessionárias distribuidoras de gás natural localizadas no sul do país. O crescimento do consumo de gás natural nas residências, por sua vez, deverá levar as concessionárias a terem uma forma de coletar estes valores de consumo de uma maneira prática e precisa. Este projeto pretende apresentar um protótipo de um aplicativo para auxílio no trabalho, de coleta de consumo de gás natural nas residências. Tal aplicativo foi desenvolvido para a plataforma Android que está disponível numa variedade de dispositivos móveis atuais como *smartphones* e *tablets*, e tendo como uma de suas principais características ser de código aberto e gratuito.

Palavras-chave: Android. Mobilidade. Coletor de dados. Dispositivos Móveis.

ABSTRACT

The natural gas market in Brazil has a tendency to grow in the next years and the residential segment, until then very little explored compared to the traditional segments of natural gas distributing as vehicular and industrial. The growth of the natural gas consumption in residences, by the way, will make the distribution companies look for new ways to collect these consumption values in a practical and precise manner. This project intends to introduce a prototype of an application to aid the work of collecting the consumption of natural gas in residences. The application was developed for the platform Android, that is available in a variety of modern mobile devices such as smartphones and tablets, and having as one of its main features to be a free and open source application.

Keywords: Android. Mobility. Data collector. Mobile devices.

LISTA DE ILUSTRAÇÕES

Figura 1: Fluxograma de procedimento de natureza contínua. 14
Figura 2: Ordem de leitura em um quadro de medidores. 15
Figura 3: Arquitetura do Android. 21
Figura 4: Os componentes de uma aplicação do Android. 23
Figura 5: Ciclo de vida de uma aplicação no Android. 25
Figura 6: Estudo inicial das principais telas do sistema. 31
Figura 7: Tela de criação de um projeto no Eclipse. 34
Figura 8: Estrutura de um projeto Android no Eclipse. 35
Figura 9: Estrutura da tabela ROTEIROS. 36
Figura 10: Caso de uso do aplicativo de coleta de consumo. 38
Figura 11: Diagrama de seqüência do aplicativo de coleta de consumo. 40
Figura 12: Tela com o *menu* inicial da aplicação. 41
Figura 13: Confirmação de importação e exportação: (A) e (B). 42
Figura 14: Telas de listagens e coleta de dados: (A), (B) e (C). 43
Figura 15: Diagrama de classes. 44
Figura 16: Classes do projeto na IDE Eclipse. 45
Figura 17: Arquivos XML do projeto na IDE Eclipse. 45
Figura 18: Extrato de código de classe que estende *Activity*. 46
Figura 19: Extrato de código de layout descrito em XML. 46
Figura 20: Trecho de código utilizando *ListView* adaptada. 47
Figura 21: Trecho de código para *POST* que retorna XML. 49
Figura 22: Trecho de código para *parser* do XML retornado. 49
Figura 23: Trecho de código de importação de roteiros. 50
Figura 24: Trecho de código para envio de XML via *POST*. 50
Figura 25: Trecho do conteúdo de *AndroidManifest.xml*. 51
Figura 26: Tabela ROTEIROS na base de dados remota. 52
Figura 27: Emulador Android SDK utilizado para validação. 53
Figura 28: Dispositivo Galaxy 5 utilizado para validação. 53
Figura 29: Tela informando o processo de importação. 54
Figura 30: Seqüência para validação da funcionalidade de coleta: (A), (B) e (C). . 55
Figura 31: Validação dos alertas de inconsistência. (A), (B) e (C). 55
Figura 32: Validação da funcionalidade de coleta do consumo: (A), (B) e (C) 56
Figura 33: Validação da funcionalidade de exportação. 57
Figura 34: Base de dados remota com consumo coletado. 57

LISTA DE QUADROS

Quadro 1: Classificação dos tipos de leitura. 13
Quadro 2: Detalhamento do procedimento de natureza contínua. 15
Quadro 3: Principais plataformas para dispositivos móveis. 18
Quadro 4: Métodos do ciclo de vida de uma aplicação. 25
Quadro 5: Tipos de layout. 26
Quadro 6: Opções para persistência de dados. 28
Quadro 7: Dicionário de dados da tabela roteiros. 37

LISTA DE ABREVIATURAS E SIGLAS

ADT: Android Development Tools
API: Application Programming Interface
AVD: Android Virtual Device
FK: Foreign Key
HTTP: Hypertext Transfer Protocol
IDE: Integrated Development Environment
JDK: Java Development Kit
MP: Medida Provisória
OHA: Open Handset Alliance
PDA: Personal Digital Assistants
PHP: Hypertext Preprocessor
PK: Primary Key
PPB: Processo Produtivo Básico
RDBMS: Relational Database Management System
SD: Secure Digital
SDK: Software Development Kit
USB: Universal Serial Bus
VM: Virtual Machine
XML: eXtensible Markup Language

Sumário

1 INTRODUÇÃO 10
1.1 Contextualização 10
1.2 Objetivos 11
1.2.1 Geral 11
1.2.2 Específicos 11
1.3 Justificativa 11
1.4 Escopo e Delimitação do Trabalho 12
2 REVISÃO BIBLIOGRÁFICA 13
2.1 O Processo de leitura de medidores de gás 13
2.2 Dispositivos Móveis 16
2.3 Plataformas de Desenvolvimento 18
2.4 O Android 19
2.4.1 A Plataforma Android 20
2.4.2 Visão Geral da Arquitetura 20
2.4.3 Visão Geral do SDK 22
2.4.4 Android Runtime 22
2.4.5 Estrutura das Aplicações Android 23
2.4.6 Ciclo de Vida 24
2.4.7 Interface com o Usuário 26
2.4.7.1 Gerenciadores de *Layout* 26
2.4.7.2 Componentes de Interface 27
2.4.8 Persistência de Dados 28
3 DESENVOLVIMENTO DO PROTÓTIPO 30
3.1 Requisitos 30
3.2 Ambiente de Desenvolvimento 32
3.2.1 AVD – Android *Virtual Devices* 32
3.2.2 Criação e Estrutura do Projeto 33
3.3 Especificação da Base de Dados 36
3.4 Caso de Uso 38
3.5 Diagrama de Sequência 40
3.6 Interface 41
3.7 Implementação 44
4 VALIDAÇÃO DO PROTÓTIPO 52
5 CONCLUSÃO 58
6 CONTRIBUIÇÕES E TRABALHOS FUTUROS 60
REFERÊNCIAS BIBLIOGRÁFICAS 61

1 INTRODUÇÃO

1.1 Contextualização

O mercado de gás natural no Brasil, com as descobertas do pré-sal e a expansão da malha de gasodutos, promete ser um mercado em forte expansão nos próximos anos [1]. Empresas distribuidoras de gás natural tiveram em sua maioria crescimento em 2010 tanto em volume de consumo quanto em captação de novos clientes, e a expectativa é de altos investimentos para os próximos anos conforme demonstram as distribuidoras de gás dos estados do Paraná [2], Santa Catarina [3] e Rio Grande do Sul [4], só para ficarmos na região sul do país. Estas concessionárias prometem investir em um mercado pouco explorado, se comparado com os mercados de distribuição para indústrias e veículos, que é o mercado de distribuição de gás natural para residências.

Porém, a expansão do consumo do mercado residencial poderá trazer certa dificuldade para as distribuidoras no que diz respeito ao levantamento do gás utilizado por cada ponto de consumo residencial, já que hoje este levantamento é em sua maioria executado através da anotação manual em planilhas impressas, para posteriormente serem digitados em sistemas apropriados.

Esta sistemática torna-se inapropriada, pois tende a ocasionar problemas como perda ou inconsistência dos dados devido a erros humanos ou de transcrição de leituras, tudo isto podendo levar ao atraso ou até mesmo erro na emissão de faturas, e por consequência atraso e dificuldade no faturamento.

Uma forma de solucionar este problema seria lançar mão da utilização de dispositivos móveis para armazenar as informações de consumo registradas nos medidores das unidades consumidoras.

A computação móvel permite a quem a utiliza estabelecer comunicação com outros usuários e sistemas, bem como possibilita o gerenciamento do trabalho enquanto se movimenta, e estas são características importantes para situações na qual as atividades se apresentam geograficamente dispersas como é o caso do trabalho de coleta de consumo em campo.

1.2 Objetivos

1.2.1 Geral

Apresentar um protótipo de aplicação para dispositivos móveis, que vise auxiliar o trabalho de campo relacionado ao processo de anotação da leitura do consumo de gás natural em cada unidade consumidora.

1.2.2 Específicos

Criar um mecanismo que carregue informações de um repositório de dados armazenado remotamente, para o dispositivo móvel, contendo o roteiro de medições a ser efetuado no trabalho de campo.

Criar um mecanismo para armazenamento no dispositivo móvel, das informações referentes ao consumo de gás identificado no trabalho de campo.

Criar um mecanismo para sincronizar as informações de medição armazenadas no dispositivo móvel com o repositório de dados remoto.

1.3 Justificativa

Boa parte das distribuidoras de gás natural tem como modelo atual de coleta de informação de medidores, a utilização de planilhas impressas que recebem anotações manuais da pessoa que efetua a leitura. Posteriormente estas informações são transferidas através de digitação em um sistema específico.

Tendo em vista que o mercado de consumo residencial tende a crescer, este tipo de solução acabará se tornando inviável, pois além da morosidade e re-trabalho, é um processo no qual podem surgir falhas e inconsistências entre os dados anotados e os dados digitados.

Com o processo de automação da medição através da coleta dos dados via dispositivo móvel, estas falhas tendem a ser minimizadas, além do que proporciona uma maior agilidade na forma de levantamento e disponibilidade dos dados de medição de consumo para o processo de faturamento.

1.4 Escopo e Delimitação do Trabalho

No universo do desenvolvimento para dispositivos móveis existem diferentes categorias de dispositivos como *Smartphones*, *PDAs* e *Tablets*.

Existe também uma variada disponibilidade de plataformas de desenvolvimento para estes dispositivos, dentre as quais podemos citar Android, iOS, Symbiam, webOS entre outras.

O fato do escopo deste projeto focar no desenvolvimento de um protótipo para funcionar em um dispositivo específico (*Smartphone*) baseado em uma plataforma específica (Android) constitui-se em uma das restrições deste trabalho.

Outra restrição é relacionada ao fato de que o protótipo resultante deste trabalho não pretende ser um aplicativo de software completo, a pretensão é que este seja utilizado como um estudo de caso para projeto e melhorias subsequentes que serão descritas como trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

2.1 O Processo de leitura de medidores de gás

O processo de medição de consumo de gás natural corresponde a coletar e disponibilizar os dados referentes ao registro de consumo e inspecionar visualmente o local de instalação dos medidores procedendo com as devidas anotações de anormalidades.

O quadro 1 demonstra quais são os tipos de leitura existentes bem como o que caracteriza cada tipo de leitura.

Tipo	Descrição
Leitura Real	Efetuada através da obtenção dos números inteiros indicados no visor do medidor, na primeira coleta de dados, conforme calendário de leitura.
Leitura pela Média	Definida quando não existe possibilidade de coleta da leitura real no medidor. É indicada pelos códigos de impossibilidade de leitura.
Leitura em aberto	Leitura não realizada no dia estipulado. Deverá ser realizada na nova data determinada pela empresa.
Releitura	Segunda leitura efetuada após a leitura real ou impossibilidade de leitura devendo ser anterior ao próximo dia de medição previsto no calendário.
Validação da leitura	É a análise crítica das leituras coletadas em campo que se enquadram fora do perfil de consumo do usuário. Na validação são analisados, por usuário: dados cadastrais, serviços emitidos, perfil de consumo, condições de acesso, de instalação do medidor, cumprimento dos procedimentos de medição, possíveis erros de leitura, anormalidades e ocorrências.

Quadro 1 – Classificação dos Tipos de Leitura

O processo de leitura é dividido em procedimentos de natureza contínua e pré-determinada, conforme suas definições a seguir:

a) Procedimentos de natureza contínua são executados a partir de entrada ou exclusão de um cliente ou mediante ajustes dos recursos necessários para a execução da medição a qualquer momento. Objetiva a criação do calendário de leitura juntamente com os ciclos de consumo, grupos, rota, roteiros e medidores, disponibilizando os dados necessários para a execução da medição;

b) Procedimentos de natureza pré-determinada são aquelas realizadas em uma data determinada com início e término estipulados, visando obter os dados da medição.

A figura 1 demonstra o fluxograma dos procedimentos de natureza contínua que tem o propósito de delinear o processo de estruturação do roteiro.

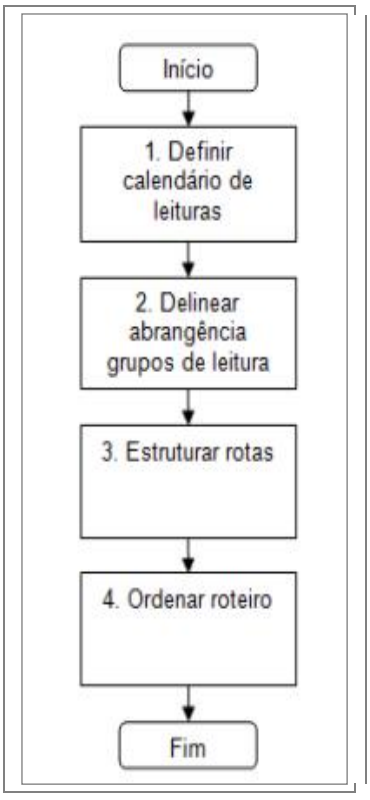


Figura 1: Fluxograma de Procedimento de Natureza Contínua

O quadro 2 exibe o detalhamento de cada etapa identificada no fluxo de procedimento de natureza contínua.

Etapa	Detalhamento do procedimento	Etapa anterior
1	O calendário de leituras deve ser estruturado em uma planilha eletrônica ou sistema informatizado devido à complexidade de informações de datas de leitura (dias úteis) e dias de consumo (entre 27 e 33)	Início
2	Os grupos de leitura em ordem crescente devem ficar próximos uns dos outros, no mapa eletrônico.	1
3	A rota é definida em função da quantidade e proximidade ou da quantidade, tempo de execução e percurso total do roteiro.	2
4	O roteiro é a ordem de leitura. Os medidores devem ser lidos de cima para baixo, da esquerda para a direita. Em um mesmo nível as leituras devem ser efetuadas no sentido horário (da esquerda para a direita), conforme demonstra a figura 2.	3

Quadro 2 – Detalhamento do Procedimento de Natureza Contínua.

Figura 2: Ordem de Leitura em um Quadro de Medidores.

2.2 Dispositivos Móveis

A constante e cada vez maior necessidade das pessoas contarem com o acesso a informações pessoais e corporativas, independente do momento ou local no qual se encontra, fez com que a indústria de tecnologia computacional trabalhasse cada vez mais no sentido de prover equipamentos que viessem de encontro a esta necessidade. Estes equipamentos, classificados como dispositivos de computação móvel, já foram encarados como simples agendas eletrônicas ou assistentes pessoais (PDA, *Personal Digital Assistants*), porém, hoje já fazem parte do cotidiano da maioria das pessoas permitindo a estas a possibilidade de deslocamento em conjunto com o seu ambiente computacional, proporcionando assim uma forma rápida e eficiente de permanecer em contato constante com suas fontes de informação.

Temos diversas categorias de dispositivos que podemos considerar de computação móvel [5], os quais podem ser divididos nos seguintes grupos:

O primeiro grupo é o dos *laptops* (*notebooks* e *netbooks*), que são dispositivos portáteis com capacidade semelhante aos computadores pessoais (*desktops*).

No segundo grupo encontram-se os aparelhos PDA, que podem possuir aplicativos desenvolvidos por linguagens de alto nível, capacidade de reproduzir recursos multimídia, fornecer acesso a redes, etc. Apresentam um poder de processamento geralmente menor do que os *laptops*, porem superior ao desempenho dos celulares.

Os celulares pertencem a um terceiro grupo de dispositivos que possuem poder de processamento reduzido, porém com recursos que vão desde o acesso à rede *Bluetooh*, quanto ao suporte a executar aplicativos desenvolvidos em linguagem Java.

Num quarto grupo podemos classificar dois tipos que começam a despontar no mercado de dispositivos móveis, os *smartphones* que possuem os recursos do celular incorporando muitos recursos dos aparelhos PDA, e os t*ablets* que são dispositivos com telas sensíveis ao toque e com recursos e sistemas operacionais semelhantes aos *laptops*.

A popularidade em conjunto com a disponibilidade de ofertas e planos de acesso proporcionados pelas empresas de telefonia torna o celular o dispositivo móvel com acesso a recursos de Internet mais utilizados hoje. Porém, com a entrada do mundo corporativo cada vez mais demandando soluções para computação móvel, os *smartphones* aparecem também como grande opção.

Os *tablets* ainda se encontram em uma fase inicial de utilização tanto pelo mercado

corporativo, quanto pela população em geral, em muitos casos por conta do custo do dispositivo que ainda é bastante alto comparado a um celular ou até mesmo a um *smartphone* de configuração popular.

Porém, a intenção do Governo Federal [20] de incluir neste ano os *tablets* no Processo Produtivo Básico (PPB), que possibilita a desoneração do equipamento através da redução de impostos sobre o produto, bem como a proposta de inclusão dos mesmos na MP do Bem [21] que dá incentivos tributários para fabricação e venda de equipamentos eletrônicos visando inclusão digital, poderá fazer com que o custo destes dispositivos caia consideravelmente, o tornando também um dispositivo móvel popular.

Diversas plataformas e linguagens de programação podem ser utilizadas para criação e execução de aplicativos para dispositivos móveis, dependendo do fabricante pode haver até mais de uma linguagem disponível para desenvolvimento no mesmo dispositivo, de acordo com o suporte disponível no hardware.

Além disto, o desenvolvimento para dispositivos móveis ainda requer certa atenção a aspectos relativos a limitações destes dispositivos como, por exemplo, o tamanho reduzido da tela, e recursos de memória e processamento reduzidos.

2.3 Plataformas de Desenvolvimento

São diversas as plataformas e linguagens disponíveis para a criação de aplicativos móveis cada qual com a sua peculiaridade. O quadro 3 exibe as principais plataformas na atualidade.

	Android	Blackbarry	iOS	LiMo
Mantenedora	Google / OHA	RIM	Apple	LiMo Foudation
Kernel	Linux	BlackBerry OS	Mac OSX	Linux
Desenvolvimento	Java, C, C++, JavaScript	JavaME, BB API	Objective C	C, C++
SDK	Sim	Sim	Sim	Não
Código Aberto	Sim	Não	Não	Para membros
Loja de Aplicativos Oficial	Android Market	App World	App Store	Não disponível
N° de apps (aproximado)	206 mil	27 mil	333 mil	---
Fabricante de Aparelhos	Diversos	RIM	Apple	Diversos

	MeeGo	Symbian	WebOS	WinPhone
Mantenedora	Nokia	Nokia	HP	Microsoft
Kernel	Linux	EPOC	Linux	Windows
Desenvolvimento	Qt, C++	C++, Qt, Python Ruby, .NET	C++, JavaScript	C#
SDK	Sim	Sim	Sim	Sim
Código Aberto	Sim	Sim	Parcial	Não
Loja de Aplicativos Oficial	OVI	OVI	Palm App	Market Place
N° de apps (aproximado)	Não disponível	30 mil	6 mil	7 mil
Fabricante de Aparelhos	Intel, Nokia	Diversos, Nokia	Palm/HP	Diversos

Quadro 3 - Principais plataformas para dispositivos móveis (adaptado de [18])

Os sistemas operacionais móveis estão presentes na maioria dos dispositivos que utilizamos no dia a dia, sejam eles aparelhos celulares, *smartphones*, *tablets*, GPS e até mesmo nosso carro.

Atualmente as atenções estão muito voltadas para o Android, que segundo pesquisas realizadas pelo instituto Nielsen [6], pela primeira vez superou o iPhone no mercado americano na preferência dos consumidores. O estudo mostra que 31% dos americanos demonstraram interesse em dispositivos baseados em Android contra 30% que sejam baseados no iOs, sendo que o BlackBerry ocupa a terceira posição com 22% das escolhas. O estudo mostra ainda aspectos interessantes, como o WindowsPhone que permanece estagnado desde o seu lançamento com 6% das intenções de utilização e também o Symbian, que outrora já foi líder de mercado, hoje não aparece nem com 1% do interesse.

Esta tendência de preferência ao Android também já foi apontada pelo Instituto Gaertner [7] no início deste ano de 2011. Para o Instituo o Android - até o final de 2012 - dominaria quase metade do mercado representando 49% dos dispositivos vendidos ao redor do mundo.

Baseado nestas pesquisas é possível identificar o quanto é promissor o mercado de desenvolvimento de aplicativos para a plataforma Android, e este foi um dos motivos da escolha desta plataforma para o desenvolvimento do protótipo que será demonstrado neste trabalho. Outros motivos não menos importantes foram:

- É uma plataforma *open source*, baseada em um núcleo robusto como é o Linux.
- O desenvolvimento é baseado em Java que conta com uma grande comunidade de desenvolvedores, e possui documentação de acesso livre.
- Está disponível em uma vasta quantidade de marcas e modelos de dispositivos móveis, com várias opções de preços, podendo com um custo relativamente baixo ter acesso a um dispositivo para teste real da aplicação, sem ficar dependente apenas de emuladores.

2.4 O Android

O Android foi um projeto inicialmente desenvolvido por uma *startup* americana do Vale do Silício chamada Android Inc. Esta pequena empresa foi adquirida pelo Google no ano de 2005, que por sua vez tratou de amadurecer o projeto e o tornou público em meados de 2007 com o objetivo de apresentar a primeira plataforma *open source* de desenvolvimento para dispositivos móveis. Atualmente o Android é mantido por um grupo denominado *Open Handset Alliance* (OHA), que é formado por mais de 40 empresas das

quais figuram o próprio Google e outras de importância nos ramos de telefonia (Telefónica), fabricação de semicondutores (Intel) e fabricação de celulares (Motorola), dentre outras.

2.4.1 A Plataforma Android

O Android é uma plataforma para desenvolvimento de aplicativos voltados para funcionar em dispositivos móveis baseados em um núcleo de Linux, sendo que as aplicações a serem geradas são escritas em linguagem Java. Estas aplicações são compiladas em *bytecodes Dalvik* e executadas em uma máquina virtual desenvolvida especialmente para utilização em dispositivos móveis denominada Máquina Virtual *Dalvik*.

Disponibiliza um kit de desenvolvimento denominado Android SDK que proporciona as APIs e ferramentas necessárias para o desenvolvimento de aplicações, tendo como principais recursos:

- *Application framework* que proporciona a reutilização de componentes;
- *Dalvik virtual machine* que é otimizada para dispositivos móveis;
- Um *browser* integrado baseado no *webkit engine*;
- Gráficos otimizados através de utilização de bibliotecas 2D; e 3D baseada na especificação OpenGL ES 1.0 (aceleração de hardware opcional).
- SQLite para armazenamento de banco de dados estruturados;
- Suporte multimídia para áudio, vídeo e formatos de imagem (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF);
- Ambiente para de desenvolvimento rico, apresentando emulador de dispositivo, ferramentas de depuração, memória, desempenho e um *plugin* para o Eclipse (ADT).

Os seguintes recursos também estão presentes, porém dependentes de hardware: telefonia GSM, *Bluetooth*, EDGE, 3G, WiFi, câmera, GPS, compasso, e acelerômetro.

2.4.2 Visão Geral da Arquitetura

Android é uma plataforma que apresenta desde sistema operacional, até *middleware* e aplicativos, conforme pode ser observado na figura 3. Sua arquitetura é

dividida em diversas camadas as quais são: núcleo do sistema operacional, bibliotecas, *runtime*, *framework* e aplicativos.

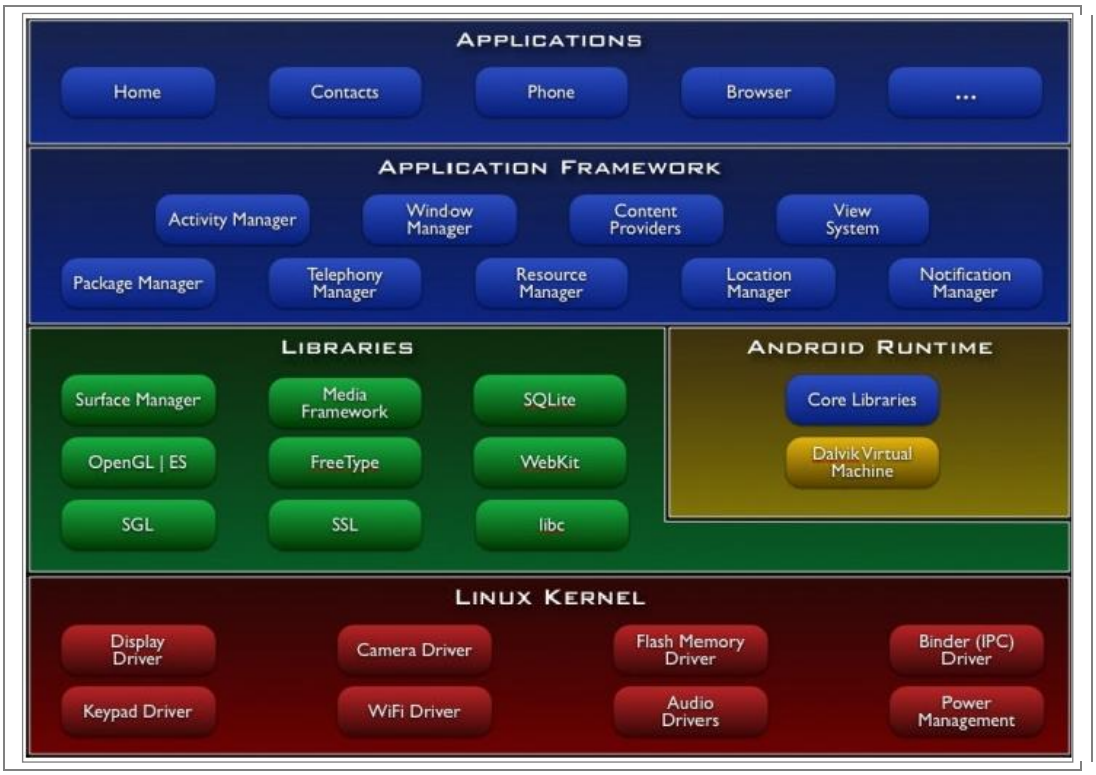


Figura 3 - Arquitetura do Android [15]

Na camada do núcleo (Linux *Kernel*), baseada em Linux, localiza-se o sistema operacional da plataforma, responsável por serviços denominados de baixo nível como gerenciamento de processos, gerenciamento de memória, segurança, etc.

Na camada de bibliotecas (*Libraries*), ficam as APIs desenvolvidas em C/C++ e que dão suporte dentre outros recursos à renderização 3D (OpenGL ES), gerenciamento de base de dados (SQLite) e suporte aos diversos formatos de vídeo e áudio.

Na camada de *runtime* (Android *Runtime*), encontram-se componentes como as *core libraries*, que disponibilizam a API Java necessária para a escrita do código de programação das aplicações, bem como a *Dalvik Virtual Machine*, que é a máquina virtual que dará condições para que a aplicação Java desenvolvida possa ser executada.

Na camada de *framework* (*Application Framework*), estão localizadas as APIs que serão utilizadas pelas aplicações que executam sobre a plataforma do Android, como por exemplo, os gerenciadores de telefonia, localização e notificação.

Na camada restante, as de aplicativos (*Applications*), estarão representadas as aplicações que são executadas sobre a plataforma, sejam elas nativas como o caso da

calculadora, do gerenciador de contatos, do calendário, etc., ou aplicações desenvolvidas por terceiros como é o caso do protótipo que será desenvolvido neste trabalho. Para a camada de aplicativos, não existe diferença entre aplicações nativas e aplicações de terceiros, todas são escritas com as mesmas APIs e executadas no mesmo *runtime*, inclusive tendo a possibilidade da troca de uma aplicação nativa por outra que tenha a mesma finalidade e seja desenvolvida por um terceiro ou pelo próprio usuário.

2.4.3 Visão Geral do SDK

O kit de desenvolvimento para Android (Android SDK) está disponível para Windows, Linux e MacOS, provendo ao desenvolvedor um conjunto rico de ferramentas que inclui um depurador, bibliotecas, emulador de *smartphone*, documentação, código de exemplo e tutoriais.

2.4.4 Android Runtime

O mecanismo de *runtime* do Android é baseado em dois grupos fundamentais que são as suas bibliotecas centrais e sua máquina virtual criada para que cada dispositivo móvel possa executar múltiplas VM.

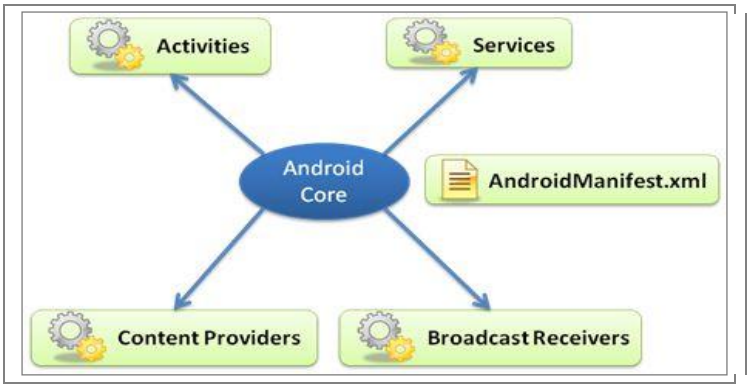
Cada aplicação desenvolvida em Android executa em um único processo o qual é uma instância desta máquina virtual, executando arquivos no formato ‘.dex’ (abreviação de *Dalvik Executable*), formato otimizado para carregamento rápido e com consumo mínimo de memória.

A VM *Dalvik* executa classes compiladas em linguagem Java, porém transformadas no formato ‘.dex’, usando ainda o *kernel* do Linux para aumentar as suas funcionalidades como o uso de *threads* e gestão de memória de baixo nível.

2.4.5 Estrutura das Aplicações Android

Aplicações desenvolvidas em Android são baseadas em uma arquitetura de componentes chave demonstrada pela figura 4, porém, não necessariamente uma aplicação deve obrigatoriamente utilizar-se de todos estes componentes, no geral as aplicações são compostas por uma combinação destes.

Em conjunto com estes componentes existe um arquivo XML denominado *AndroidManifest.xml* de existência obrigatória, e no qual são feitas configurações gerais da aplicação e dos componentes utilizados por ela. Juntam-se a esta estrutura dois outros itens importantes que fazem estes quatro componentes chave funcionarem que são as *Intents* e as *Views.*



Figuras 4 – Os componentes de uma aplicação Android [17]

Activities formam a base para o desenvolvimento visual de uma aplicação, sendo o componente mais comum em uma aplicação. Uma *activity* é quem realiza os tratamentos dos eventos da tela e que define qual *view* será desenhada na tela [9]. Para uma aplicação que possua múltiplas telas, cada tela deverá ser representada por uma *activity* que é implementada como uma subclasse de *Activity*.

Services são códigos executados em segundo plano e que não apresentam uma interface visual. Cada *service* é executado na *thread* principal do processo que o criou, não causando bloqueio ou interferência em outros componentes. Cada serviço é uma classe que herda de *Service*.

Broadcast Receivers são componentes receptores de ocorrências de eventos do sistema e que reagem a estes eventos. Cada *receiver* é uma classe que herda de *BroadCastReceiver*.

Content Providers são componentes que tornam um conjunto específico de dados

da aplicação disponível para outras aplicações. Cada *provider* é uma classe que herda de *ContentProvider* e disponibiliza um conjunto padrão de métodos para que outras aplicações possam recuperar e armazenar dados do tipo que o provedor controla.

Intents são mensagens responsáveis por ativar os componentes *Service*, *Activity* e *BroadcastReceiver* de uma aplicação. Essas mensagens são utilizadas para facilitar a ligação entre os componentes da aplicação ou de aplicações diferentes, em tempo de execução.

Views são elementos utilizados para definir objetos gráficos exibidos na tela, com o objetivo de prover interação com o usuário. Exemplo destes elementos são botões, caixas de diálogo, mapas entre outros.

AndroidManifest.xml é o arquivo de manifesto escrito em XML, obrigatório e único para a aplicação. Nele são descritos os componentes que fazem parte da aplicação, definidos nomes para as *activities*, o modo de orientação da tela, bem como declaradas permissões para acesso a recursos como o GPS ou Internet. Este arquivo lista também as bibliotecas que a aplicação vai usar e qual *activity* principal irá iniciar a aplicação.

2.4.6 Ciclo de Vida

O ciclo de vida de uma aplicação no Android é representado pelos seguintes estados [9]: em execução, temporariamente interrompida em segundo plano ou completamente destruída.

Deve-se ter como entendimento o termo ciclo de vida como algo que tem início, meio e fim, sendo que a documentação do Android indica a existência de três subníveis do ciclo de vida, que por sua vez ficam se repetindo durante a execução de uma aplicação, são eles:

Entire lifetime, que representa um ciclo de vida completo entre o início e a destruição da *activity* da aplicação.

Visible lifetime*,* no qual a *activity* da aplicação está iniciada podendo estar no topo da pilha interagindo com o usuário, ou temporariamente parada em segundo plano.

Foreground lifetime*,* no qual a *activity* da aplicação está no topo da pilha e interagindo com o usuário.

A figura 5 demonstra o ciclo de vida completo de uma *activity* da aplicação, exibindo os estados possíveis e a chamada de cada método.

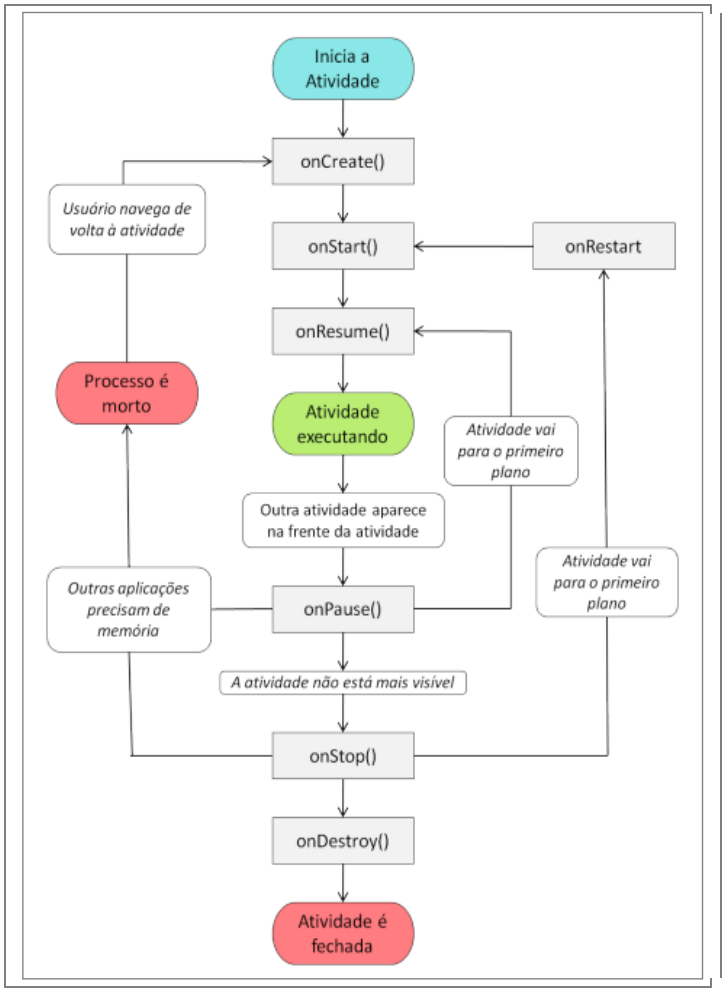


Figura 5 – Ciclo de Vida de uma Aplicação no Android [16]

Cada um destes ciclos se inicia durante a chamada de um dos métodos controladores e termina quando algum outro método é chamado. O quadro 4 exibe as descrições de cada um destes métodos.

Método	Descrição
onCreate(bundle)	Método obrigatório e invocado uma única vez, neste momento cria-se a *view* e invoca-se o método *setContentView(view)* para que seja exibida a tela representada pela *view*. Depois de finalizado, o método *onStart( )* é chamado para que se dê início ao ciclo de vida visível da *activity*.
onRestart( )	Método chamado quando a *activity* está ficando visível ao usuário, dependendo do estado da aplicação pode ser chamado depois do método *onCreate( )* ou *onRestart( )*.
onStart( )	Método chamado quando uma *activity* está parada temporariamente e em processo de reinício, de forma automática chama o método *onStart( )*.

onResume( )	Método chamado quando a *activity* começa a interagir com o usuário, representa o estado de execução da aplicação. É chamado sempre depois do método *onStart( )*.
onPause( )	Método chamado quando algum evento ocorrer com o dispositivo e que faça com que a *activity* encontrada no topo da pilha tenha de ser temporariamente interrompida. É normalmente utilizado para gravar dados não salvos até o momento, para que tudo possa ser recuperado, se necessário, no método *onResume( )*.
onStop( )	Método chamado quando a *activity* está sendo encerrada, e não está mais visível ao usuário. Se a *activity* for reiniciada, o método *onRestart( )* é chamado ou no caso de ficar muito tempo parada o Android pode chamar automaticamente o método *onDestroy( )* para remover completamente a *activity* da pilha.
onDestroy( )	Método que literalmente encerra a execução de uma *activity*, sendo feita após sua execução, remoção completa da aplicação da pilha e o encerramento completo do processo.

Quadro 4 – Métodos do ciclo de vida de uma aplicação

2.4.7 Interface com o Usuário

A classe *android.view.View* é a classe mãe de todos os componentes visuais e suas subclasses irão compor uma interface com o usuário no Android. Estas subclasses implementam o método *onDraw(Canvas)*, responsável por desenhar os componentes na tela. Estes componentes dividem-se em dois grupos, os *widgets* e os gerenciadores de layout. O primeiro é um componente simples que herda diretamente da classe *View*, como exemplos temos as classes *Button*, *ImageView* e *TextView*, já os gerenciadores de layout são subclasses de *android.view.ViewGroup*.

2.4.7.1 Gerenciadores de *Layout*

Um gerenciador de *layout* organiza a disposição dos componentes na tela, as possibilidades de *layout* são classificadas nos seguintes tipos:

Tipo	Descrição
AbsoluteLayout	Permite posicionar os componentes, fornecendo as coordenadas x e y.

FrameLayout	Tipo mais comum e simples de *layout*, utilizado por um componente que precisa preencher a tela inteira.
LinearLayout	Utilizado para organizar os componentes na vertical ou horizontal.
TableLayout	É uma herança de *LinearLayout* e pode ser utilizado para organizar os componentes em uma tabela, com linhas e colunas.
RelativeLayout	Permite posicionar um componente relativo a outro, por exemplo, abaixo, acima ou ao lado de um componente já existente.

Quadro 5 – Tipos de *layout*

2.4.7.2 Componentes de Interface

Como em qualquer outra aplicação, no Android também é necessário trabalhar com componentes para compor uma interface com o usuário. As interfaces de aplicações Android são constituídas de componentes gráficos denominados *widgets* e que são subclasses da classe *View.* Existe uma ampla variedade destes componentes disponíveis na plataforma Android, sendo que os básicos são:

TextView - representa a primeira e mais simples das subclasses de *View* e representa um texto ou rótulo na tela.

EditText - subclasse de *TextView,* utilizada para que o usuário possa digitar informações em um campo de texto.

Button e ***ImageButton*** - componentes utilizados para criar um botão na tela, sendo que no caso de *ImageButton* pode-se usar uma imagem para desenhar o botão.

RadioButton – componente que permite ao usuário selecionar apenas uma única opção de uma determinada lista.

CheckBox – componente que permite ao usuário selecionar uma ou mais opções de uma determinada lista.

ListView – componente que permite exibir uma lista de conteúdos.

2.4.8 Persistência de Dados

O Android fornece diversas opções para persistir dados [8] em uma aplicação, e a escolha de qual opção seguir depende da característica de cada aplicação com relação aos dados. Deve-se levar em consideração se estes devem ser de acesso privado à aplicação, ou se podem ser acessados por outras aplicações, ou por consequência por outros usuários, e também de quanto espaço será necessário para manter os dados.

O quadro 6 mostra as opções para persistência de dados:

Tipo	Característica
Internal Storage	Armazenamento de dados privados na memória do dispositivo
External Storage	Armazenamento de dados públicos no cartão de memória
Network Connection	Armazena dados na web no seu servidor de rede.
Shared Preferences	Armazenamento de dados primitivos provados em pares chave-valor
SQLite Databases	Armazenamento de dados estruturados num banco de dados privado

Quadro 6 – Opções para persistência de dados.

Internal Storage representa o armazenamento diretamente na memória interna do dispositivo, sendo que os dados armazenados desta forma por padrão são privados apenas à aplicação que os gravou, portanto outras aplicações não poderão ter acesso a eles. A remoção dos dados é feita quando o usuário desinstala a aplicação.

External Storage pode ser utilizada nas ocasiões em que o dispositivo suporta o uso de memórias externas como um cartão do tipo SD ou uma memória interna não removível. Os arquivos armazenados desta forma poderão ser lidos por qualquer outra aplicação ou dispositivo, podendo também ser modificados pelo usuário quando da transferência USB do dispositivo para outro computador.

Um cuidado que deve ser observado neste caso é que arquivos armazenados nesta forma podem ser perdidos se o usuário montar a mídia em um computador ou remover o cartão, não existindo medidas de segurança para salvaguardar este tipo de perda, pois todas as aplicações podem ler e escrever arquivos em memória externa e o usuário pode removê-los.

Network Connection pode ser utilizada quando há uma rede disponível para armazenar os dados dos serviços baseados em *web*, sendo que para usar este tipo de operação deve-se utilizar classes localizadas nos pacotes *java.net* e *android.net*.

Utilizando ***Shared Preferences*** pode-se persistir dados através da utilização de um *framework* geral que permite salvar e recuperar pares de chaves-valor de tipos de dados primitivos como *booleans, floats, ints, longs,* e *strings*, sendo que estes dados serão persistentes entre sessões, mesmo que a aplicação seja encerrada.

SQLite Databases possibilita a persistência de dados estruturados através de um motor de banco de dados leve e simples. Cada aplicação pode criar um ou mais banco de dados, sendo visíveis apenas à aplicação que os criou.

3 DESENVOLVIMENTO DO PROTÓTIPO

3.1 Requisitos

O protótipo da aplicação foi desenvolvido para funcionar em *smartphones* com plataforma Android, sendo capaz de listar roteiro pré-cadastrado de imóveis para medição do consumo de gás. Ao ser selecionado um imóvel deverão ser listados todos os pontos de consumo existentes no imóvel.

Ao selecionar um ponto de consumo no imóvel, deverá ser apresentada a tela para execução da anotação do consumo, bem como informações sobre o cliente e aparelho medidor, os campos para inserção de informações deverão ser o de valor atual de consumo e de anomalia identificada durante a anotação (p.ex. cachorro solto impedindo a medição, medidor avariado, etc.).

Após efetuar a gravação do valor de consumo, a aplicação deve voltar à tela que lista os pontos de consumo, até que todos os pontos de consumo daquele imóvel tenham o valor de consumo informado, não tendo mais anotações em aberto para o imóvel, o sistema deve voltar à tela que lista o roteiro de imóveis, para que se possa selecionar novo imóvel e iniciar a anotação de consumo dos pontos vinculados ao imóvel.

A figura 6 representa um estudo inicial que demonstra a estrutura para as principais telas do sistema que deverão respectivamente:

1) exibir a relação de imóveis de acordo com o roteiro;

2) exibir a relação de pontos de consumo baseados no imóvel;

3) exibir os campos para anotação do consumo referente ao ponto.

Figura 6: Estudo inicial das principais telas do sistema.

Os roteiros pré-cadastrados devem ser carregados pela aplicação de um repositório de dados remoto já existente, da mesma forma, após serem efetuadas as gravações de todos os valores de consumo das unidades consumidoras, o sistema deve ser capaz de enviar estas informações para o mesmo repositório de dados remoto.

O sistema deve estar preparado para funcionar em dispositivos móveis (*smarthphones*) com plataforma Android (versão 2.1 ou superior), permitindo que uma equipe de posse destes dispositivos em trabalho de coleta de consumo em campo, possa interagir com uma aplicação *web* remota que acessará uma base de dados ligada a um sistema departamental, possibilitando efetuar tanto a carga dos roteiros quanto o envio dos dados para este repositório remoto.

3.2 Ambiente de Desenvolvimento

Para o desenvolvimento de aplicativos em Android é necessário que esteja configurado um ambiente contendo a última versão do *Java Development Kit* (JDK) juntamente com o Android SDK, também é altamente recomendável que se utilize uma IDE para a codificação.

Para o protótipo desenvolvido neste trabalho, optou-se pela utilização da IDE Eclipse na sua versão 3.6 (codinome Helios), pois além de ser a IDE mais indicada pelas referências oficiais [11], é uma IDE de código aberto e com a opção de extensão de suas funcionalidades através da instalação de vários *plugins* disponíveis, dentre eles o ADT que é um *plugin* desenvolvido para estender as funcionalidades do Eclipse possibilitando a criação de projetos voltados para o Android. Este *plugin* utiliza as bibliotecas disponibilizadas pelo SDK, funcionando como uma ponte que une o Android SDK ao Eclipse.

Detalhes de como fazer a instalação e configuração de um ambiente para desenvolvimento em Android (IDE + ADT + SDK), podem ser obtidos na referência oficial do projeto Android [12].

3.2.1 AVD – Android *Virtual Devices*

Estando instalado e configurado o ambiente (IDE + ADT + SDK), é o momento de adicionar um dispositivo virtual (AVD), para que se possa executar e testar a aplicação desenvolvida. O AVD é um dispositivo virtual do sistema operacional do Android que simula um aparelho no qual você poderá executar e testar suas aplicações, você pode criar vários AVDs de acordo com o tipo de aparelho que você deseja emular e versão da plataforma Android na qual você está desenvolvendo.

Um AVD é basicamente composto de:

- Um perfil de hardware (possui câmera, utiliza teclado, etc.).
- Qual versão da plataforma Android a executar.
- Emulador de cartão SD.
- Área de armazenamento na máquina de desenvolvimento.

Detalhes de como criar um AVD podem ser obtidos na referência oficial [13].

3.2.2 Criação e Estrutura do Projeto

Com o ambiente de desenvolvimento configurado e a AVD criada, já é possível criar o projeto da aplicação, e para tanto na IDE Eclipse deve-se seguir o caminho *File -> New -> Other,* selecionar a opção *Android -> Android Project* e clicar no botão *Next.* A figura 7 demonstra a tela de configuração do projeto no Eclipse, sendo que os principais itens que devem ser observados são:

Build Target: Selecionar a versão da plataforma na qual será desenvolvido o aplicativo, desta forma a aplicação funcionará nesta versão ou em versões superiores.

Application Name: Especificar o nome do aplicativo, que será exibido na lista de aplicativos do dispositivo. O nome definido aqui também será o que estará disponível na loja de aplicativos do Android caso o aplicativo venha ser publicado.

Package Name: Especificar o nome do pacote que conterá as classes que estendem de *Activity*. Caso a aplicação venha ser publicada na loja de aplicativos do Android, será necessário você ter certeza de que não haverá outro denominado de forma igual, para que esta situação seja evitada sugere-se que o pacote seja um nome de domínio ou que seja composto pelo nome e sobrenome do desenvolvedor.

Min. SDK Version: Especificar qual o número da versão mínima compatível com a aplicação a ser desenvolvida, normalmente será o número que acompanha a versão do Android escolhida em *Build Target*.

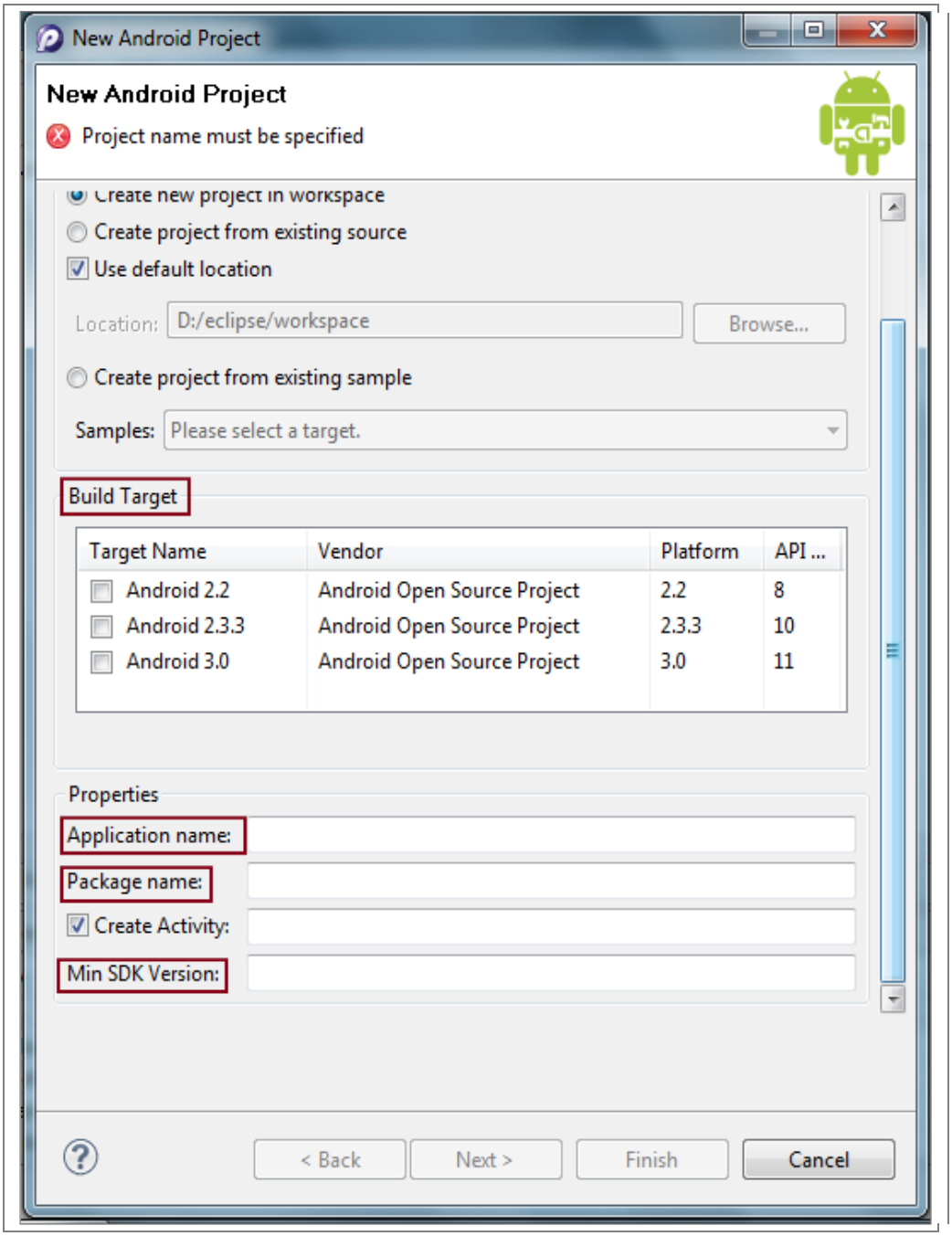


Figura 7: Tela de configuração de um projeto no Eclipse.

Depois de confirmadas as configurações o Eclipse cria o projeto com estrutura semelhante à exibida na figura 8, cada item desta estrutura tem um significado especial, os principais itens são descritos a seguir:

src: pasta na qual fica a sua *Activity* principal definida durante a criação do projeto, bem como todas as outras classes que deverão compor o código fonte da aplicação.

gen: pasta na qual é criada a classe *R.java*, que é gerada automaticamente pelo ADT e não deve ser alterada manualmente. Contém todas as referências aos recursos que representam a aplicação, tais como arquivos de XML, imagens, etc.

assets: pasta auxiliar que pode conter arquivos diversos como fontes *TrueType*, *javascripts*, etc.

res: localização dos recursos da aplicação como arquivos de imagem, internacionalização e *layout*.

drawable: pasta na qual ficam localizadas as imagens da aplicação, sendo possível categorizá-las em três resoluções distintas utilizando as sub-pastas: *drawable-ldpi, drawable-mdpi e drawable-hdpi*.

layout: pasta na qual estarão localizados os arquivos XML que representam as telas da aplicação.

values: pasta na qual ficam arquivos XML que guardam *Strings* que podem ser usadas na aplicação, os valores que constam neste XML são guardados através de *tags* com a seguinte estrutura *"<string name="nomeString">Conteúdo da String </string>*".

AndroidManifest.xml: é o arquivo de manifesto, nele constarão todas as configurações da aplicação e de cada *Activity* existente.

default.propierties: arquivo que contém configurações do projeto e é utilizado para controle de revisão de código. Assim como o *R.java*, não deve ser editado manualmente.

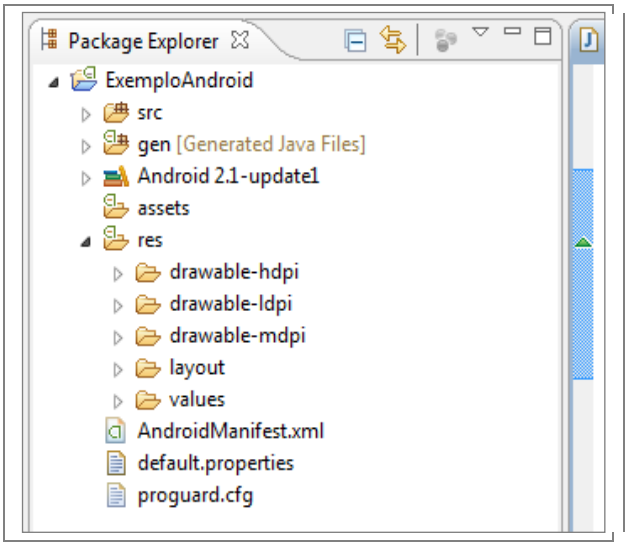


Figura 8: Estrutura de um projeto Android no Eclipse.

3.3 Especificação da Base de Dados

O protótipo necessita de um banco de dados para armazenar as informações coletadas sobre o valor consumido de gás em cada ponto de consumo.

A figura 9 exibe a estrutura da tabela na qual serão armazenadas as informações referentes ao consumo, a chave primária está representada pela sigla *PK (primary key)* e a chave estrangeira está identificada como a sigla *FK (foreign key)*. Esta tabela representa o armazenamento local dos dados de consumo coletados, uma estrutura com no mínimo estes campos deverá estar disponível em uma fonte de dados remota para que seja possível fazer a importação dos dados referentes aos roteiros de coleta, bem como a exportação dos dados de consumo coletados durante o trabalho de campo.

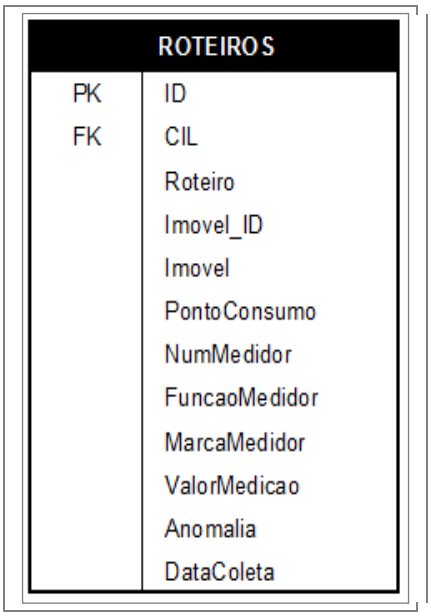


Figura 9: Estrutura da tabela ROTEIROS.

Esta base de dados será criada no dispositivo móvel usando o motor de banco de dados denominado SQLite, que é um recurso nativo encontrado na plataforma Android para persistência de dados estruturados.

SQLite [22] é uma biblioteca C que implementa um banco de dados SQL embutido, permitindo aos programas que a utilizam ter acesso a banco de dados SQL sem executar um processo RDBMS separado.

SQLite não é uma ferramenta de cliente usada para conectar com um grande servidor de banco de dados, mas sim o próprio servidor, lendo e escrevendo diretamente para e do arquivo do banco de dados no disco.

Algumas Características do SQLite:

- Software Livre e Multiplataforma.
- Mecanismo de armazenamento seguro com transações ACID.
- Não necessita de instalação, configuração ou administração.
- Implementa a maioria das funcionalidades do SQL92.
- Cada base de dados criada é armazenada em um único arquivo.
- Suporta bases de dados acima de 2 *terabytes*.
- Sem dependências externas.

O dicionário de dados da tabela que armazenará os roteiros é exibido no quadro 7, ele informa a descrição de cada campo bem como o seu tipo de dado.

Atributo	Descrição	Tipo	Tamanho
ID (PK)	Código da coleta	Numérico	5
CIL (FK)	Código do cliente	Numérico	5
Roteiro	Descrição do roteiro	Caractere	50
Imovel_ID	Código do imóvel	Numérico	5
Imovel	Identificação do imóvel	Caractere	50
PontoConsumo	Identificação do ponto de consumo (Cliente)	Caractere	50
NumMedidor	Número do medidor	Numérico	4
FuncaoMedidor	Função do Medidor	Caractere	50
MarcaMedidor	Marca do medidor	Caractere	50
ValorMedicao	Valor de consumo coletado	Numérico	5
Anomalia	Desc. de anomalia ocorrida durante a coleta	Caractere	50
DataColeta	Data de registro da coleta	Data	-x-

Quadro 7: Dicionário de dados da tabela ROTEIROS.

3.4 Caso de Uso

Um caso de uso representa a interação do atores com o sistema e pode ser representado por uma descrição simples e em linguagem natural. Isso ajuda a identificar os objetos e a compreender o que o sistema deve realizar [14].

A figura 10 demonstra o diagrama de caso de uso referente à interação dos atores com a aplicação executada no dispositivo móvel.

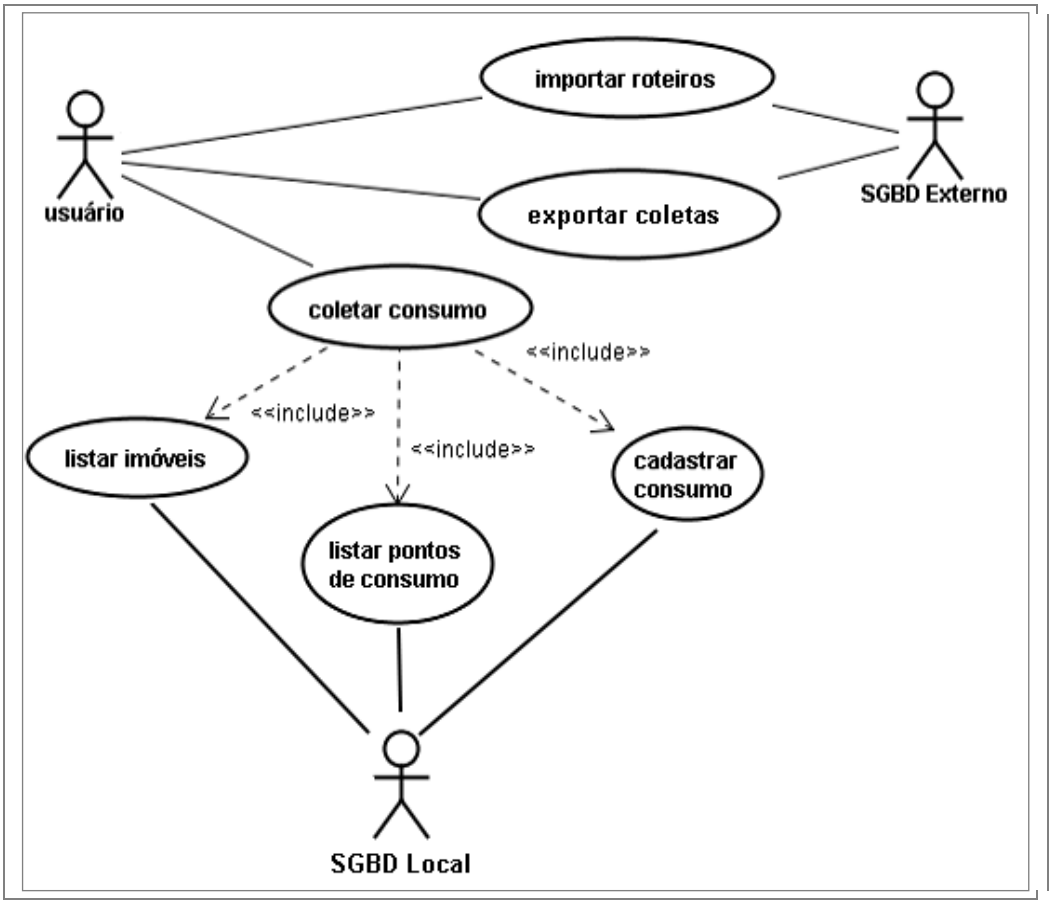


Figura 10: Caso de uso do aplicativo de coleta de consumo.

Na arquitetura são propostos seis casos de uso que representam funcionalidades disponíveis e interações dos atores envolvidos com o sistema.

No primeiro caso denominado “importar roteiros”, o usuário irá interagir através do sistema com um repositório de dados externo ao dispositivo que conterá os roteiros de coleta de consumo, neste caso de uso os roteiros importados são armazenados no repositório de dados local do dispositivo para utilização no trabalho da coleta de dados.

No segundo caso de uso denominado “exportar roteiros”, o usuário irá interagir através do sistema com um repositório de dados externo ao dispositivo com o objetivo de

enviar os dados coletados no trabalho de campo, as informações de consumo que se encontram armazenadas no dispositivo serão armazenadas neste repositório de dados remoto para utilização de algum sistema departamental.

No terceiro caso de uso denominado “coletar consumo”, o usuário irá interagir através do sistema com o repositório de dados local do dispositivo, que possuirá a lista de roteiros importada previamente da base de dados remota, este caso de uso inclui outros três casos que são respectivamente:

“Listar imóveis”, que irá interagir com o armazenamento local para exibir a lista de imóveis disponíveis no roteiro, bem como o número de pontos de consumo disponíveis no imóvel e que ainda não tiveram o seu consumo coletado.

“Listar pontos de consumo”, que irá interagir com o armazenamento local para exibir a lista de pontos de consumo com informações específicas do cliente, e que ainda não tiveram a coleta de consumo efetuada.

“Cadastrar consumo”, que deverá exibir o formulário de coleta de dados para o ponto de consumo selecionado, neste caso haverá a interação com o armazenamento local para persistir o valor de consumo anotado, visando futura exportação para o repositório de dados externo.

3.5 Diagrama de Sequência

O diagrama de sequência demonstra como são realizadas as interações entre objetos de forma sequencial [19].

No diagrama exibido na figura 11 é possível verificar a sequência de interações encontradas no aplicativo de coleta de consumo.

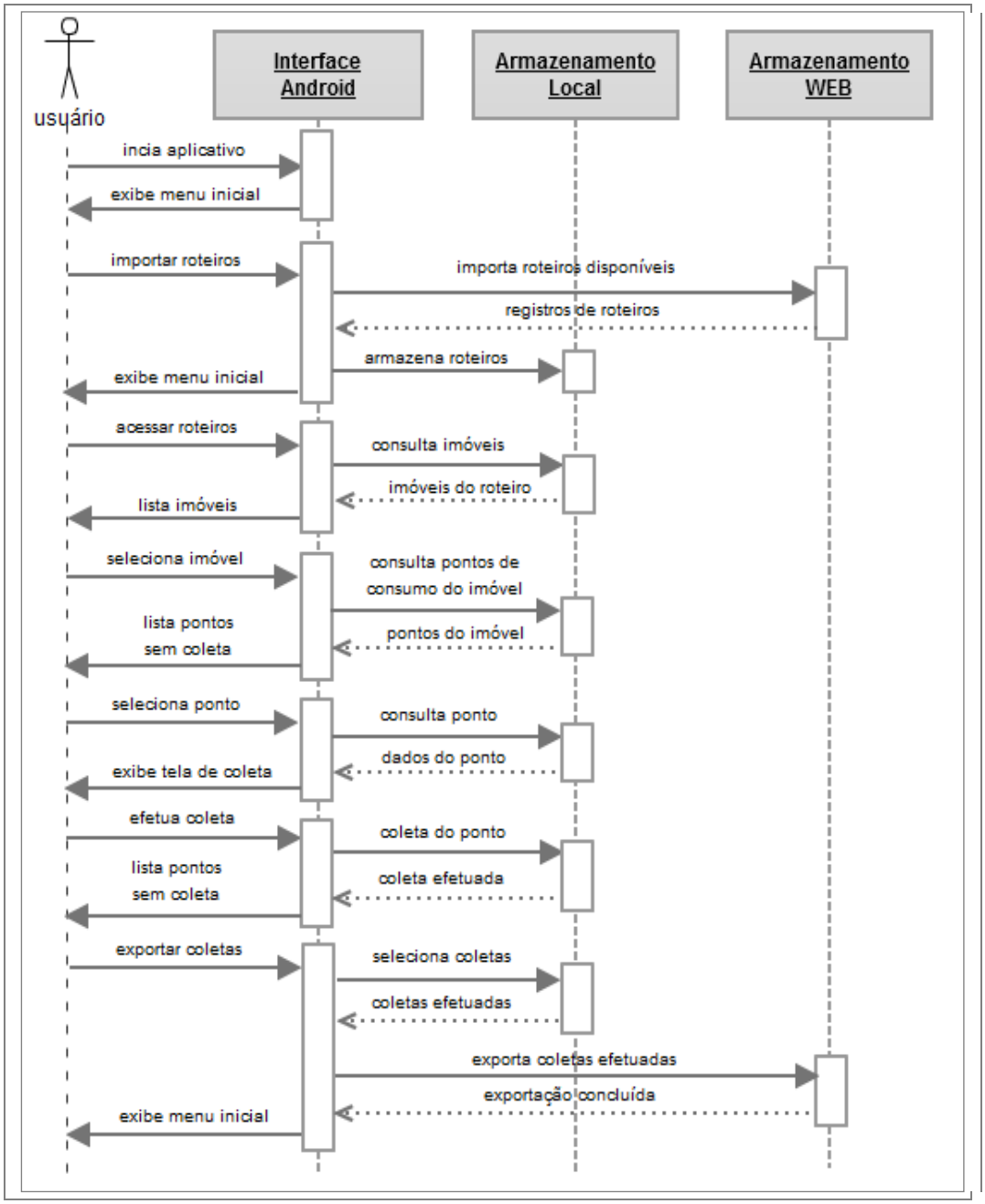


Figura 11: Diagrama de sequência do aplicativo de coleta de consumo.

3.6 Interface

O protótipo da aplicação é composto por quatro telas sendo que cada uma destas representa um componente atividade (*Activity*).

Estes componentes estão descritos no arquivo de manifesto (*AndroidManifest.xml*), bem como constam no mesmo arquivo as permissões para garantir ao protótipo o acesso à Internet.

A primeira tela a ser exibida é a tela que contém o *menu* inicial da aplicação (figura 12), esta tela provê quatro botões de comando que servirão para:

- Acionar a base de dados remota para importar a lista de roteiros com os pontos de consumo.
- Acessar os roteiros importados para dar início à coleta de dados.
- Enviar para a base de dados remota os dados coletados.
- Encerrar a aplicação voltando para o *menu* inicial do aparelho.

Figura 12: Tela com o *menu* inicial da aplicação.

Ao ser acionado o primeiro botão do *menu* inicial, uma mensagem de diálogo (fig. 13A) será exibida solicitando a confirmação da operação de importação de roteiros. No caso de aceitação, o sistema tentará acessar a base de dados remota com informações de roteiros, isto retornará a lista de roteiros disponíveis sendo que o roteiro mais atual substituirá o anterior caso este já exista na base de dados.

As informações já existentes no dispositivo relacionadas a roteiros anteriores serão perdidas caso ainda não tenha sido exportadas. No caso de negação de confirmação o aplicativo retorna ao *menu* inicial.

Figura 13: Confirmação de importação e exportação: (A) e (B).

Ao ser selecionado o botão que indica a exportação de dados, a aplicação apresenta outra caixa de diálogo (fig. 13B) e que também solicita a confirmação da operação. Se confirmada, tentará acessar a base de dados remota, porém desta vez com o objetivo de enviar os dados coletados e gravá-los remotamente. O cancelamento da operação faz a aplicação voltar à tela que exibe o *menu* inicial.

O acionamento do botão que dá o acesso ao roteiro irá exibir uma tela com a

listagem dos imóveis que fazem parte daquele roteiro (fig. 14A), acompanhando cada registro de imóvel é exibido o número de pontos de consumo que fazem parte daquele imóvel e que ainda não tiveram sua informação de consumo coletada.

A partir do momento que o consumo do ponto é coletado, este número é decrementado, e vai diminuindo até que as informações sejam coletadas para todos os pontos, neste caso, o item que contem a descrição do imóvel é eliminado da tela.

A partir do momento que não existirem mais imóveis listados na tela, indica que todos os imóveis constantes naquele roteiro tiveram seus pontos de consumo coletados.

Figura 14: Telas de listagens e coleta de dados: (A), (B) e (C).

Ao ser selecionado um item na lista de imóveis do roteiro, a aplicação exibe outra lista contendo os pontos de consumo do imóvel selecionado (fig. 14B), os itens desta lista vão deixando de ser exibidos a partir do momento que a coleta da informação de consumo do ponto é efetuada.

Ao ser selecionado um item na lista de pontos de consumo, a aplicação exibe o formulário de coleta de dados para aquele ponto (fig. 14C), com informações sobre o ponto selecionado, e dois campos para a informação da medição do consumo e, caso observado, alguma anomalia encontrada no momento da coleta ou que não possibilitou a coleta no respectivo ponto. Nesta tela, ao ser selecionado o botão de gravação o processo de coleta de consumo é efetuado, a aplicação volta então a exibir a lista de pontos de consumo já sem o item referente ao ponto anteriormente coletado.

3.7 Implementação

Na implementação do protótipo foi utilizada a linguagem Java que é a linguagem indicada para desenvolvimento de aplicações na plataforma Android, a estrutura do protótipo pode ser analisada segundo o digrama de classes exibido na figura 15.

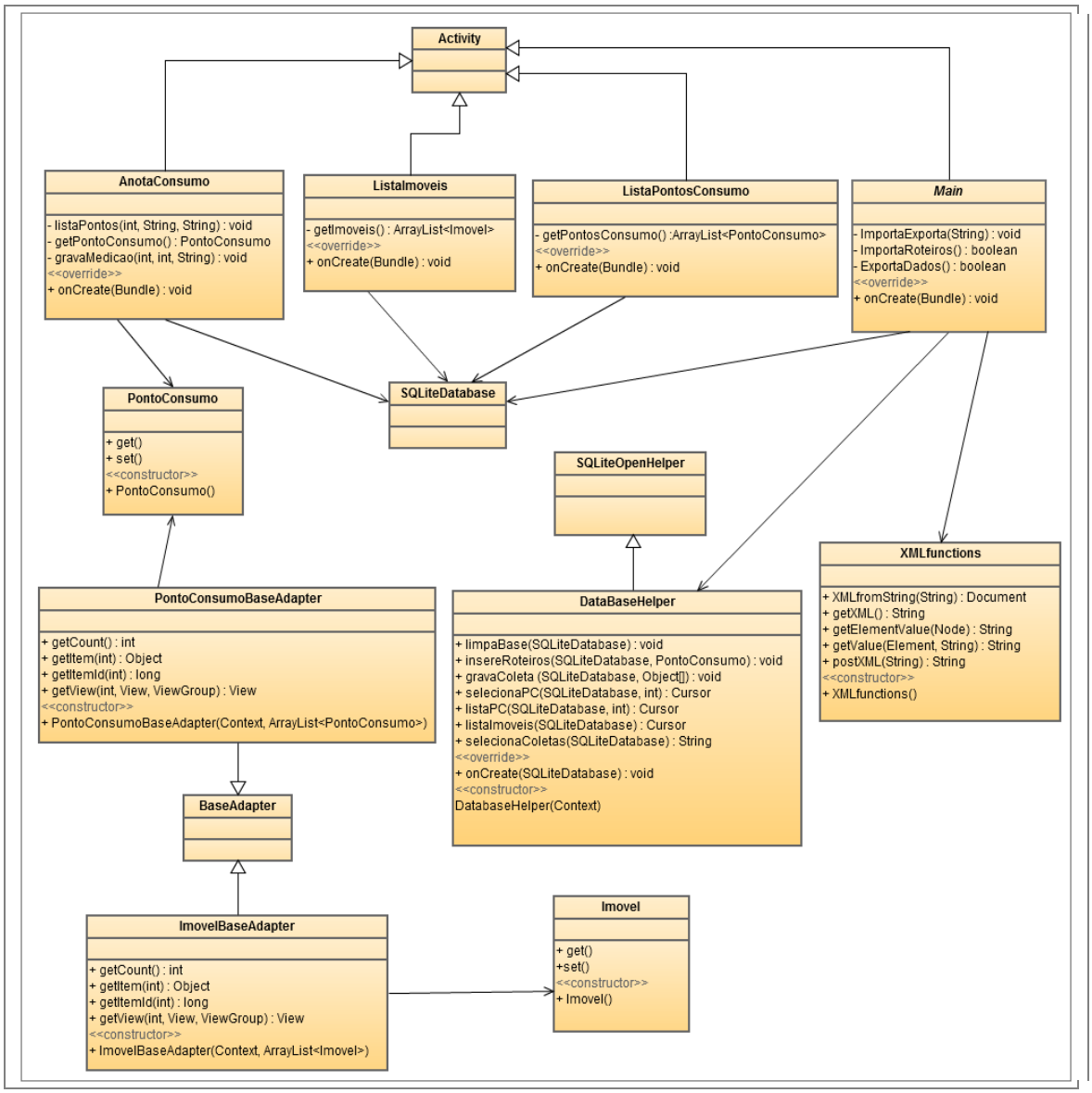


Figura 15: Diagrama de classes

Para o desenvolvimento foi utilizada a IDE Eclipse juntamente com o *plugin* ADT, a figura 16 exibe a estrutura do projeto no que se refere às classes implementadas.

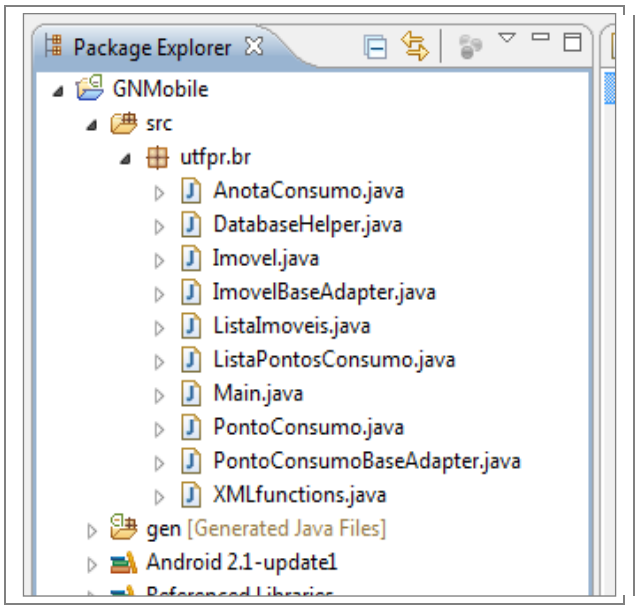


Figura 16: Classes do projeto na IDE Eclipse

Apesar de a plataforma Android permitir que a codificação de interfaces de forma procedural, permitindo codificar os elementos diretamente nas classes, é recomendada como uma boa prática de programação a construção das interfaces de forma declarativa utilizando descritores XML, pois proporcionam a separação dos elementos de *layout* (componentes gráficos de interface) do código propriamente dito, facilitando desta forma manutenções futuras da aplicação.

A figura 17 exibe a estrutura do projeto na IDE Eclipse com os arquivos XML implementados para a definição das interfaces do protótipo.

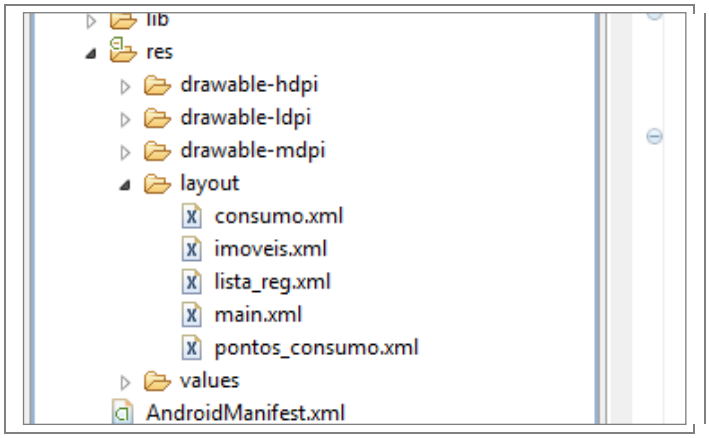


Figura 17: Arquivos XML do projeto na IDE Eclipse

O projeto do protótipo é composto por dez classes, as classe *Main*, *ListaImoveis*, *ListaPontosConsumo* e *AnotaConsumo* herdam da classe *Activity* e definem

uma interface que consiste em uma visualização que responde a eventos. Cada classe está associada a um determinado arquivo descritor XML criado na pasta *layout* e que define os componentes a serem exibidos na tela, cada componente é identificado por um *id* específico que é vinculado no código da classe.

As figuras 18 e 19 exibem os extratos de código da classe *Main* e do arquivo de *layout main.xml* que é vinculado a classe, demonstrando como este vínculo é feito.

```
public class Main extends Activity {

    private SQLiteDatabase db = null;
    private String TipoOperacao = "";
    private ProgressDialog pd = null;
    private XMLfunctions xmlf = new XMLfunctions();
    private DatabaseHelper helper = new DatabaseHelper(this);

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bt_importar = (Button) findViewById(R.id.bt_importar);
        bt_importar.setOnClickListener(new View.OnClickListener(){
            public void onClick(View arg0){
                ImportaExportaRoteiros("IMP");
            }
        });
```

Figura 18: Extrato de código de classe que estende *Activity*.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <ImageView
        android:layout_width="fill_parent" android:layout_height
        android:src="@drawable/fundo" />

    <LinearLayout xmlns:android="http://schemas.android.com/apk/
                  android:padding="4sp"  >
     <TextView android:id="@+id/rotuloLista"   android:textSize=
                  android:textColor="#FFF" />
       <Button android:text="IMPORTAR ROTEIROS"
        android:id="@+id/bt_importar" android:paddingTop="4dip"
        android:layout_height="wrap_content" />
```

Figura 19: Extrato de código de *layout* descrito em XML.

As classes *ImovelBaseAdapter* e *PontoConsumoBaseAdapter herdam* da classe *BaseAdapter* e trabalham como um adaptador que irá interpretar a fonte de dados e formatar da maneira que se deseja exibir a informação nos componentes de *ListView,*

estes são exibidos nas telas que listam os imóveis (fig. 14A) e os pontos de consumo disponíveis (fig. 14B). Em ambos os casos as fontes de dados serão respectivamente *arrays* de objetos das classes *Imovel* e *PontoConsumo*.

O componente *ListView* que irá exibir imóveis ou pontos de consumo é definido no arquivo *lista_reg.xml*, porém após ser vinculado na *activity* de exibição, é adaptado para o *layout* disponível respectivamente em *imóveis.xml* e *pontos_consumo.xml*, conforme pode ser conferido na figura 20 que exibe um trecho de código da classe *ListaImoveis*.

```
public class ListaImoveis extends Activity {

    private SQLiteDatabase db = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.lista_reg);

        ArrayList<Imovel> imoveis = GetImoveis();

        Button bv = (Button) findViewById(R.id.bt_voltar);
        bv.setOnClickListener(new View.OnClickListener(){
            public void onClick(View arg0){
               startActivity(new Intent(ListaImoveis.this, Main.class));
            }
        });

        final ListView lv1 = (ListView) findViewById(R.id.Lista);
        lv1.setAdapter(new ImovelBaseAdapter(this, imoveis));
        lv1.setOnItemClickListener(new OnItemClickListener() {
```

Figura 20: Trecho de código de *ListaImoveis* utilizando *ListView* adaptada.

A persistência dos dados é feita através da classe *DatabaseHelper* utilizando uma base de dados relacional (SQLite) que é armazenada localmente, esta classe herda de *SQLiteOpenHelper* e é a classe responsável por criar a estrutura e popular a base com os roteiros de coleta, retornar a seleção dos imóveis e pontos de consumo existentes e gravar os dados de consumo coletados.

Para popular a base de armazenamento local com os roteiros de coleta, foi necessária a integração do protótipo com um sistema web que acesse um repositório de dados remoto. Esta responsabilidade fica a cargo da classe *XMLfunctions* que irá fazer a

integração entre as aplicações utilizando somente as APIs e recursos nativos do Android, sem a necessidade de utilizar componentes proprietários ou de terceiros.

Com a intenção de demonstrar que a plataforma Android pode ser integrada com linguagens e ambientes distintos, optou-se por fazer esta integração de dados com um sistema web rodando em PHP com acesso a uma base de dados MySQL.

O *PHP: Hypertext Preprocessor* (PHP) é uma linguagem para criação de *scripts*, licenciada como *software* livre e largamente utilizada para construção de sistemas *web*. Já o MySQL e um Sistema Gerenciador de Banco de Dados (SGBD) que utiliza a linguagem SQL e assim como o PHP também é licenciado como *software* livre, sendo igualmente popular na solução para armazenamento de dados em sistemas web.

Para realizar esta integração com o script em PHP, a classe *XMLfunctions* utiliza a classe *DefaultHttpClient* e *HttpPost*, que implementam a comunicação através do protocolo HTTP via método POST.

O Hypertext Transfer Protocol (HTTP) é um protocolo de rede responsável pela transferência de dados e pela comunicação entre cliente e servidor no ambiente da Internet, o qual permite a transferência de dados em hipermídia (texto, imagem e som). Um de seus principais métodos de transferência é o POST que permite que um cliente envie mensagens ao servidor, o qual irá manipular os dados da forma desejada, sendo que a informação é incluída no corpo do comando dando mais segurança entre a requisição e a resposta.

Todo este processo é implementado pela classe *XMLfunctions* nos métodos *getXML()* e *postXML()*. Para o caso da importação de roteiros que retornará um arquivo XML com o conteúdo solicitado, será necessário ainda um procedimento de *parser* sobre este retorno para que a aplicação possa ler o arquivo XML e navegar no seu conteúdo, este procedimento fica a cargo do método *XMLfromString*.

O código exibido na figura 21 demonstra um trecho da implementação necessária para enviar uma requisição POST para o script PHP que conecta a base de dados remota, e retorna em formato XML os roteiros disponíveis para coleta.

```
public String getXML() {

        DefaultHttpClient httpclient = new DefaultHttpClient();
        String line = null;

        try {

            HttpPost httpPost = new HttpPost("http://jcgcwb.atwebpages.com/new/import.php");

            HttpResponse httpResponse = httpclient.execute(httpPost);
            HttpEntity httpEntity = httpResponse.getEntity();
            line = EntityUtils.toString(httpEntity);

        } catch (UnsupportedEncodingException e) {
```

Figura 21: Trecho de código para POST que retorna XML.

O código exibido na figura 22 demonstra um trecho da implementação necessária para o procedimento de *parser*, que carrega em um objeto *Document* o conteúdo do XML formatado.

```
public final Document XMLfromString(String xml){

    Document doc = null;

    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    try {

        DocumentBuilder db = dbf.newDocumentBuilder();

        InputSource is = new InputSource();
        is.setCharacterStream(new StringReader(xml));
        doc = db.parse(is);

    } catch (ParserConfigurationException e) {
```

Figura 22: Trecho de código para parser do XML retornado.

Isto é necessário para que a aplicação possa ler o arquivo XML e navegar no seu conteúdo recuperando as informações encontradas nos nós e armazenamento estas informações na base de dados do dispositivo.

O código exibido na figura 23 demonstra um trecho da implementação do método da classe *Main*, que é o responsável por receber as informações importadas no formato XML, acionar o *parser*, e navegar pelas informações encontradas em nós do XML e que serão armazenadas na base de dados do dispositivo.

```
private boolean ImportaRoteiros() {

    try {
        Toast.makeText(Main.this, "iniciando importação...", Toast.LENGTH_SHORT).show();

        db = helper.getWritableDatabase();
        PontoConsumo pc = new PontoConsumo();

        String xml = xmlf.getXML();
        Document doc = xmlf.XMLfromString(xml);

        NodeList nodes = doc.getElementsByTagName("ponto_consumo");
        int n = nodes.getLength();

        if (n==0) {
            Toast.makeText(Main.this, "A importação de roteiros não retornou informações
        } else {
            for (int i = 0; i < nodes.getLength(); i++) {
                Element e = (Element)nodes.item(i);
                pc.setCIL(Integer.parseInt(xmlf.getValue(e, "cil")));
                pc.setRoteiro(xmlf.getValue(e, "roteiro"));
                pc.setImovel_ID(xmlf.getValue(e, "imovel_id"));
                pc.setImovel(xmlf.getValue(e, "imovel"));
```

Figura 23: Trecho de código de importação de roteiros.

O código exibido na figura 24 demonstra um trecho da implementação necessária para o envio via POST de um arquivo em formato XML para o script PHP, neste arquivo constam às informações de consumo coletadas.

```
public String postXML(String xml) {

    DefaultHttpClient httpclient = new DefaultHttpClient();

    HttpPost httppost = new HttpPost("http://jcgcwb.atwebpages.com/new/export.php");

    try {
        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(1);
        nameValuePairs.add(new BasicNameValuePair("coletados", xml));
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

        HttpResponse response = httpclient.execute(httppost);
        String result = EntityUtils.toString(response.getEntity());
        return result;

    }
    catch (ClientProtocolException e) {
```

Figura 24: Trecho de código para envio de XML via POST.

Um aplicativo em Android se caracteriza por um conjunto de atividades (*Activities*) que vão sendo adicionadas a uma pilha e disponibilizadas para navegação do usuário. Esta atividades devem estar definidas em um arquivo de manifesto denominado *AndroidManifest.xml*.

Neste arquivo também deverão estar presentes todas as configurações e

permissões necessárias para execução das funcionalidades do aplicativo, no caso do protótipo deve ser observada a permissão para acesso a Internet, caso contrário os procedimentos de integração com a base de dados remota não funcionam, a figura 25 exibe o conteúdo do arquivo de manifesto *AndroidManifest.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="utfpr.br"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

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

        <activity android:name=".ListaImoveis" android:label="Imóveis no Roteiro">
        </activity>

        <activity android:name=".ListaPontosConsumo" android:label="Pontos de Consumo no Imóvel">
        </activity>

        <activity android:name=".AnotaConsumo" android:label="Coleta de Consumo">
        </activity>

    </application>

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

Figura 25: Trecho do conteúdo de *AndroidManifest.xml.*

4 VALIDAÇÃO DO PROTÓTIPO

Para validação do protótipo foram utilizados dois *scripts* simples em PHP que consultam e atualizam dados em uma base de dados MySQL remota. Tanto os scripts quanto a base de dados foram hospedados num servidor externo ao ambiente no qual estava sendo desenvolvida a aplicação.

Para tal, foi escolhido um servidor de hospedagem gratuito (*awardspace.com*), que possibilitou rodar os *scripts* PHP e criar a base MySQL.

Antes de iniciar a validação das funcionalidades do protótipo, foi necessário carregar a base de dado remota com alguns dados de roteiros fictícios, conforme demonstra a figura 26 (foram criados dez registros).

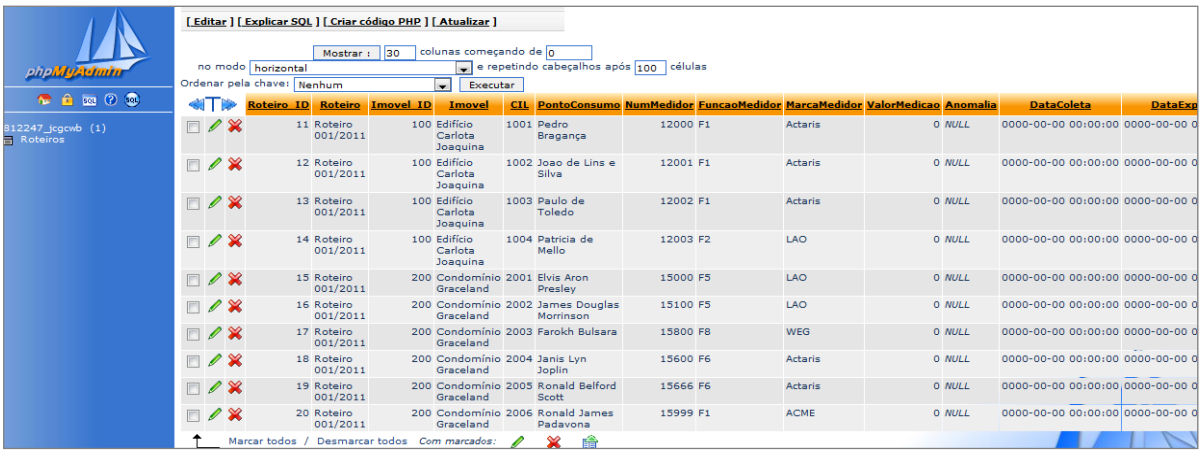


Figura 26: Tabela ROTEIROS na base de dados remota.

A aplicação foi validada primeiramente no emulador disponibilizado pelo Android SDK (ver figura 27), porém o emulador tem limitações comparadas à execução em um dispositivo real como, por exemplo, simular o toque de tela. Então, depois de validadas todas as funcionalidades no emulador foram validadas as funcionalidades em um dispositivo real.

Para tanto foi utilizado o aparelho fabricado pela empresa Samsung cujo modelo é denominado Galaxy 5 e que pode ser visto na figura 28.

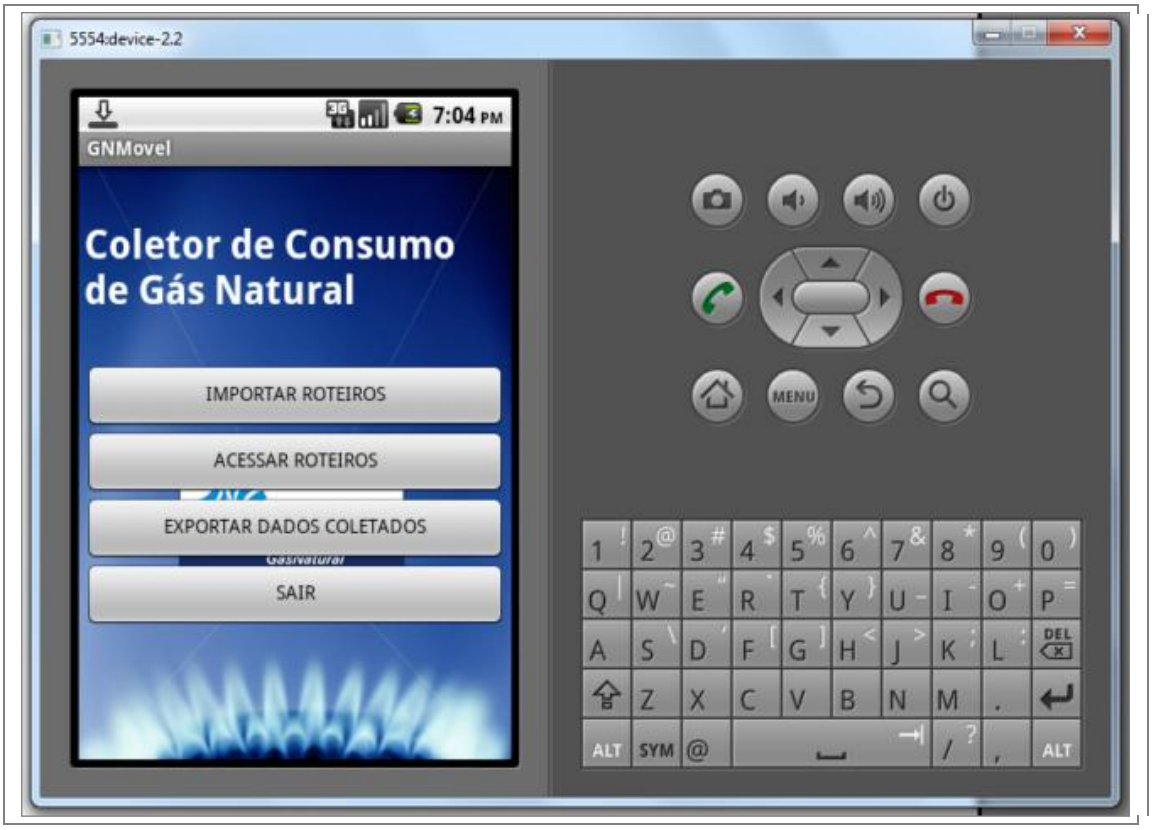


Figura 27: Emulador Android SDK utilizado para validação.

Figura 28: Dispositivo Galaxy 5 utilizado para validação.

Depois de carregada a base remota, a primeira funcionalidade do protótipo validada foi a de importação de roteiros, nesta funcionalidade é feita a comunicação com a base remota através da chamada ao *script* PHP, que por sua vez retorna o XML contendo os roteiros de coleta. Esta funcionalidade foi validada a contento, de acordo com o que exibe a figura 29 a aplicação informa ao usuário conforme vai carregando no dispositivo os pontos de consumo.

Figura 29: Tela informando o processo de importação.

O próximo teste foi acessar os roteiros importados e efetuar um registro de coleta de consumo, neste caso algumas consistências deveriam ser testadas como a certificação que apenas números inteiros deveriam ser aceitos como valor de consumo e, para o caso de impossibilidade de coleta, certificar que foi informada a anomalia que causou a impossibilidade da coleta.

Novamente a validação foi a contento conforme demonstra a figura 30 (A e B), ao ser acessada a lista de roteiros a soma da quantidade indicada para pontos de consumo em cada imóvel equivale ao número de registros criados na base de dados remota (dez registros). Ao selecionar um imóvel foram exibidos os pontos de consumo

daquele imóvel, e quando selecionado o ponto de consumo foi exibido o formulário de coleta.

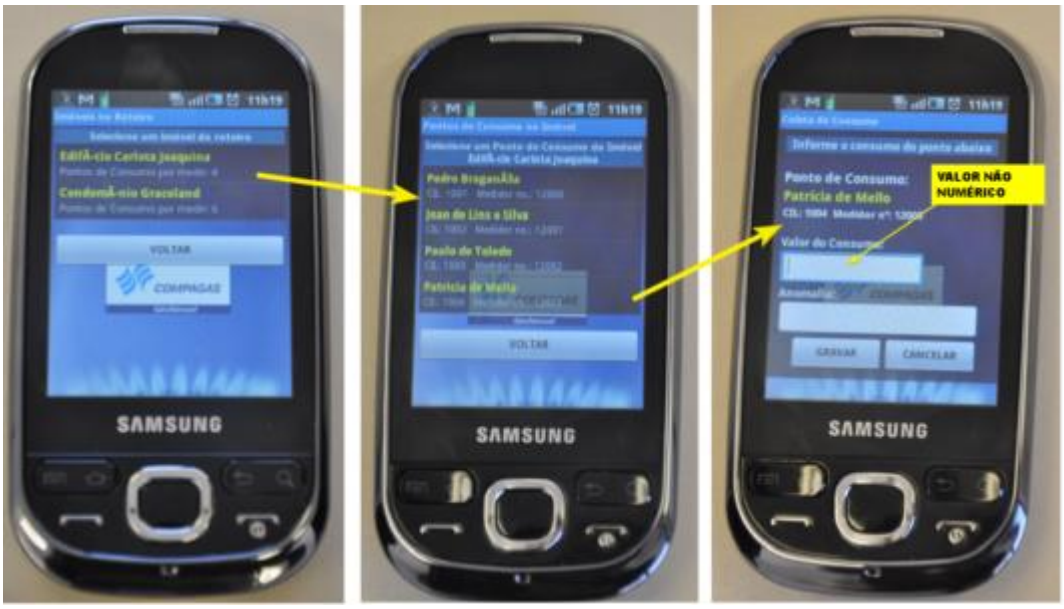


Figura 30: Seqüência para validação da funcionalidade de coleta: (A), (B) e (C).

Conforme demonstrado na figura 30(A), ao ser digitado no campo reservado ao valor do consumo uma informação que não correspondia a um número inteiro, foi exibida uma mensagem alertando a inconsistência. Da mesma forma, na figura 31(B) e 31(C), quando indicado um valor 0 (zero) para o valor do consumo correspondendo impossibilidade de coleta e não informando qual foi a anomalia que gerou a impossibilidade, foi também exibida uma mensagem alertando a inconsistência das informações.

Figura 31: Validação dos alertas de inconsistência. (A), (B) e (C).

Por fim, conforme demonstra a figura 32(A) e 32(B), ao ser inserido um valor válido para a informação de consumo, é indicado que a operação de coleta foi efetuada, voltando à lista de pontos de consumo já sem o ponto anteriormente coletado. Ao voltar à lista de roteiros, a informação referente ao número de pontos de consumo por medir está diminuída em uma unidade conforme demonstra a figura 32(C).

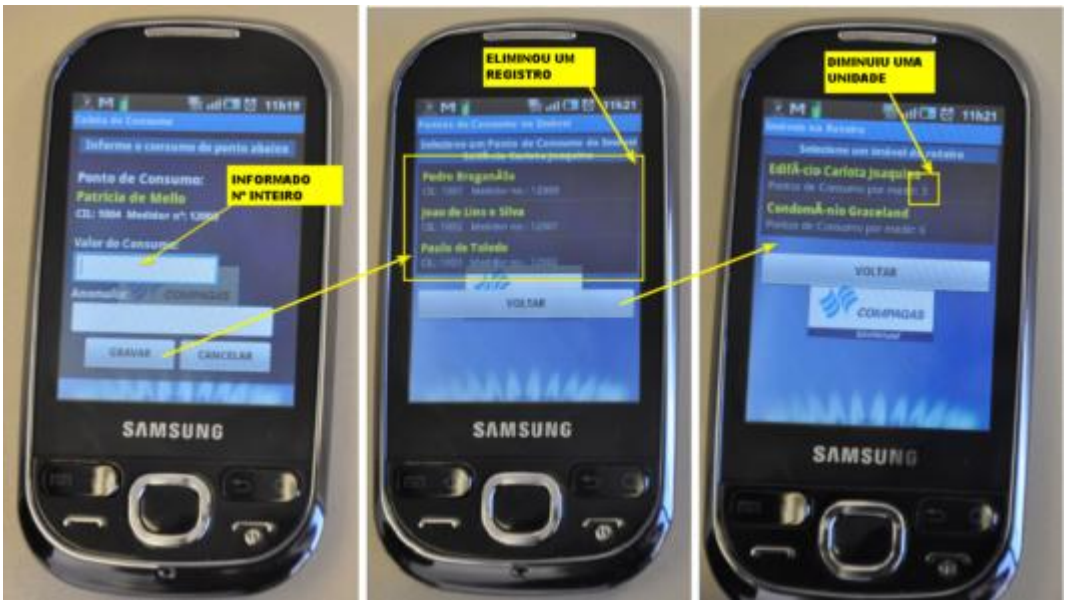


Figura 32: Validação da funcionalidade de coleta do consumo: (A), (B) e (C).

Encerrando as validações da funcionalidade de coleta, foi verificada a funcionalidade de exportar as coletas efetuadas para a base de dados remota, novamente o funcionamento foi a contento conforme pode ser observado na figura 33 que exibe comportamento da aplicação, bem como na figura 34 que exibe a lista de registros da base de dados remota já com o valor de coleta atualizado.

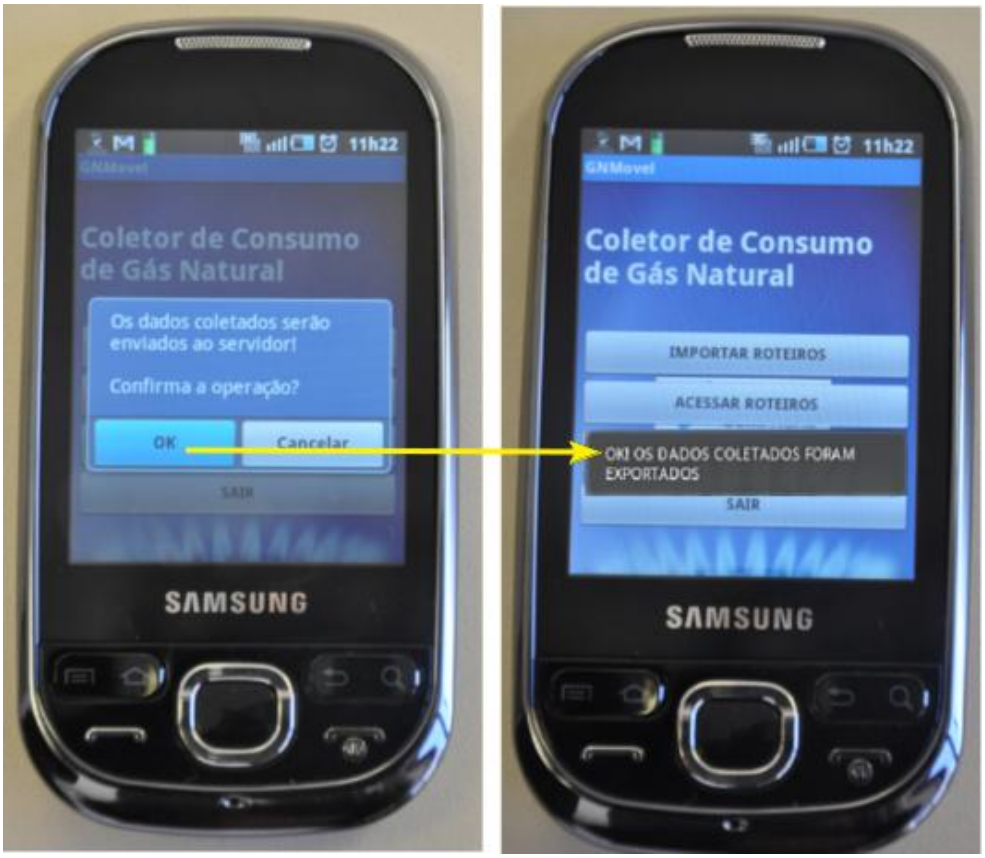


Figura 33: Validação da funcionalidade de exportação.

Executar

vel_ID	Imovel	CIL	PontoConsumo	NumMedidor	FuncaoMedidor	MarcaMedidor	ValorMedicao	Anomalia	DataColeta	DataExport
100	Edifício Carlota Joaquina	1001	Pedro Bragança	12000	F1	Actaris	0	NULL	0000-00-00 00:00:00	0000-00-00 00:00:
100	Edifício Carlota Joaquina	1002	Joao de Lins e Silva	12001	F1	Actaris	0	NULL	0000-00-00 00:00:00	0000-00-00 00:00:
100	Edifício Carlota Joaquina	1003	Paulo de Toledo	12002	F1	Actaris	0	NULL	0000-00-00 00:00:00	0000-00-00 00:00:
100	Edifício Carlota Joaquina	1004	Patricia de Mello	12003	F2	LAO	0	NULL	0000-00-00 00:00:00	0000-00-00 00:00:
200	Condomínio Graceland	2001	Elvis Aron Presley	15000	F5	LAO	0	NULL	0000-00-00 00:00:00	0000-00-00 00:00:
200	Condomínio Graceland	2002	James Douglas Morrinson	15100	F5	LAO	25	NULL	2011-08-29 06:05:04	2011-08-29 06:07:
200	Condomínio Graceland	2003	Farokh Bulsara	15800	F8	WEG	0	NULL	0000-00-00 00:00:00	0000-00-00 00:00:

Figura 34: Base de dados remota com consumo coletado.

Todas as funcionalidades da aplicação foram validadas diversas vezes sendo que para todas foram retornados os comportamentos esperados. Apenas com uma ressalva para as funcionalidades de importação e exportação, que em alguns casos apresentaram demora no retorno da resposta.

Uma hipótese para este comportamento seria a característica do servidor no qual se encontrou instalada a parte implementada em PHP/MySQL, pois servidores de hospedagem gratuita por muitas vezes podem apresentar instabilidade. Infelizmente não foi possível validar o protótipo acessando um servidor PHP/MySQL empresarial.

5 CONCLUSÃO

Em geral as concessionárias de distribuição de gás natural localizadas pelos diversos estados do país possuem metodologia de coleta de consumo pouco automatizada, sendo em muitos casos utilizados métodos pouco convencionais como anotação do valor consumido em planilhas impressas para posterior digitação em sistema apropriado.

Por outro lado observa-se que o mercado de distribuição de gás natural deverá apresentar um crescimento nos próximos anos, colocando o segmento de consumo residencial, até então pouco explorado, com um segmento de potencial crescimento. Sendo assim, o aumento de pontos individuais devido à expectativa relacionada ao mercado residencial, deverá levar a necessidade das concessionárias de encontrar soluções para incrementar o processo de coleta de consumo proporcionando um levantamento de informações de forma mais ágil e consistente.

A partir desta motivação, este trabalho apresenta uma abordagem focada na mobilidade e conectividade presente em dispositivos móveis como os *smartphones*, propondo um protótipo de aplicação desenvolvido sob plataforma Android e verificando a utilidade destes dispositivos como um agente principal na automação do processo de coleta em campo de informações referentes ao consumo de gás natural.

A escolha da plataforma Android foi motivada principalmente por ser um projeto de código aberto com disponibilidade de vasta documentação e códigos de exemplo, bem como de ferramentas gratuitas que auxiliam o desenvolvimento, aspectos que seguramente facilitam e aceleram o processo de criação de aplicações para dispositivos móveis.

O protótipo desenvolvido nesse trabalho comprovou cumprir seus objetivos, conforme as validações realizadas, dentre os quais permitir a importação de informações de roteiros de uma base de dados remota, acessar os imóveis encontrados nesta lista de roteiros, acessar a lista de pontos de consumo de determinado imóvel e possibilitar ao usuário da aplicação fornecer informações de consumo válidas referentes ao ponto de consumo escolhido. Persistir esses dados localmente ao dispositivo e enviá-los corretamente para a base de dados remota disponibilizando as informações para sistemas externos.

Por fim, obteve-se um protótipo funcional de uma aplicação voltada a coleta de

dados, utilizando-se de considerável variedade de conceitos e recursos referentes à plataforma de desenvolvimento Android, focando na interoperabilidade entre diferentes aplicações e possibilitando a integração entre mobilidade e outros ambientes tradicionais de sistemas.

Espera-se que este trabalho possa ser uma alternativa de pesquisa interessante para concessionárias de distribuição de gás que ainda tenham seu processo de coleta de informações de consumo baseado em anotações manuais através de planilhas e que pretendam agregar o aspecto da mobilidade ao seu processo, proporcionando uma forma mais ágil e mais precisa no trabalho de levantamento de informações de consumo.

6 CONTRIBUIÇÕES E TRABALHOS FUTUROS

Foram exploradas neste trabalho diversas características da plataforma Android, procurando contribuir com um conteúdo de leitura aos interessados em explorar o desenvolvimento nesta plataforma, bem como aos que se interessam pelo desenvolvimento para dispositivos móveis em geral.

Com a modelagem do protótipo esperasse contribuir para que empresas que tenham o processo de coleta de consumo em seu negócio e pretendam agregar a mobilidade ao seu processo, obtenham uma noção de como a plataforma Android pode ajudar na concepção de uma solução móvel para coleta de informações.

Sendo a aplicação apresentada um protótipo, pode ainda possuir muitas funcionalidades, algumas das quais descrevemos a seguir com o objetivo de serem incorporadas em trabalhos futuros:

- Controle através de acesso por *login* e senha.
- Implementar função que liste as coletas efetuadas com possibilidade, de um número determinado de correções do valor coletado.
- Baseado no histórico de consumo do cliente, validar a informação de consumo anotada conforme a previsão de consumo.
- No caso de inserção de valores não validados, contar quantas vezes o valor do consumo foi redigitado, tentando eliminar coletas efetuadas através do método de tentativa e erro.
- Medir o tempo transcorrido entre a seleção do imóvel e a gravação do valor de consumo de todos os seus pontos de consumo, bem como, através de GPS (ou API de localização), identificar e gravar a localização do leiturista no momento que este inicia o processo de coleta em determinado imóvel.
- Utilizar criptografia na integração de dados.
- *Listview* ou autocompletar no campo para anotação de anomalias.
- Possibilitar a anotação através de reconhecimento de voz, para os dispositivos que permitam este recurso.
- Adaptar imagens para poder utilizar a aplicação no *layout* horizontal.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] LORENZI, Sabrina. **Com mais gás, Petrobras avança no setor elétrico**. Portal IG Economia – Empresas. Disponível em: http://economia.ig.com.br/empresas/industria/com+mais+gas+petrobras+avanca+no+setor+eletrico/n1300082686367.html. Acesso em: 30 abr. 2011.

[2] Assesioria de Comunicação. **Compagas fecha 2010 em alta e projeta investimentos para 2011**. Disponível em: http://www.compagas.com.br/index.php/web/noticias/sala_de_imprensa/noticias/2011/compagas_fecha_2010_em_alta_e_projeta_investimentos_para_2011. Acesso em: 30 abr. 2011.

[3] **SCGÁS cresce em 2010 e planeja expansão até 2015**. Disponível em: http://www.scgas.com.br/noticia/index/idse/396/id/5525?dt_ini=&dt_fim. Acesso em: 01 mai. 2011.

[4] KLEIN, Jefferson. **Sulgás quer crescer em receita e número de clientes**. Jornal do Comércio. Disponível em: http://www.sulgas.rs.gov.br/index.asp?SECAO=8&SUBSECAO=36&EDITORIA=1690. Acesso em: 01 mai. 2011.

[5] Johnson, Thienne M. **Java para dispositivos móveis. Desenvolvendo aplicações com J2ME.** Editora Novatec. Novembro 2007.

[6] NIELSENWIRE. **U.S. Smartphone Market: Who's the Most Wanted?** http://blog.nielsen.com/nielsenwire/?p=27418. Acesso em 15 mai 2011

[7] GAERTNER. **Smartphone Operating System Market by Year-End 2012** http://www.gartner.com/it/page.jsp?id=1622614 Acesso em 18 mai 2011

[8] GOOGLE. **Data Storage**. Disponível em: http://developer.android.com/guide/topics/data/data-storage.html Acesso em: 30 mai. 2011.

[9] LECHETA, R. R. Google Android: **Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. Novatec, 2ª edição, 2010.

[10] DISTMO: **App Store Overview**. Disponível em: http://www.distimo.com/appstores/. Acesso em 01 jun. 2011.

[11] GOOGLE. **System Requirements**. Disponível em: http://developer.android.com/sdk/requirements.html Acesso em 04 jun. 2011.

[12] GOOGLE. **Installing the SDK**. Disponível em: http://developer.android.com/sdk/installing.html Acesso em 04 jun. 2011.

[13] GOOGLE. **Managing AVDs with AVD Manager**. Disponível em: http://developer.android.com/guide/developing/devices/managing-avds.html Acesso em 04 jun. 2011.

[14] SOMMERVILE, Ian. **Engenharia de software.** Pearson, 6ª edição, 2003.

[15] GOOGLE. **What is Android**. Disponível em: http://developer.android.com/guide/basics/what-is-android.html Acesso em 04 jun. 2011.

[16] GOOGLE. **Activity Reference**. Disponível em: http://developer.android.com/reference/android/app/Activity.html. Acesso em 04 jun. 2011.

[17] TOSIN, Carlos. **Conhecendo o Android**. Disponível em: http://www.softblue.com.br/blog/home/postid/11/CONHECENDO+O+ANDROID Acesso em 04 jun. 2011.

[18] FAEBER, N., HILZINGER, M. Sistemas Operacionais Móveis. **Linux Magazine**. São Paulo: LNMB, n.75, p. 32-37, fev. 2011.

[19] MELO, Ana Cristina. **Desenvolvendo aplicações com UML**. . Rio de Janeiro: Brasport, 2002.

[20] IDGNOW. **Desoneração pode reduzir preço do tablet**. Disponível em: http://www.brasil.gov.br/noticias/arquivos/2011/04/05/desoneracao-pode-reduzir-preco-do-tablet-brasileiro-em-mais-de-30. Acesso em 15 jul. 2011.

[21] IDGNOW. **Governo decide incluir tablet na MP do Bem**. Disponível em: http://idgnow.uol.com.br/computacao_pessoal/2011/05/14/governo-decide-incluir-tablet-na-mp-do-bem-preco-deve-cair/. Acesso em 15 jul. 2011.

[22] SQLite. **About SQLite**. Disponível em: http://www.sqlite.org/about.html. Acesso em 15 jul. 2011.