Universidade Federal de Santa Catarina

Leonardo Lino Vieira

API para Desenvolvimento de Módulo Móvel Coletor de dados para Sistemas Computacionais

Volume Único

Florianópolis(SC)

2013/1

API para Desenvolvimento de Módulo Móvel Coletor de dados para Sistemas Computacionais

Volume Único

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal de Santa Catarina como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Área de Concentração: Ferramentas de TI e Sistema móvel para celular.

Orientador: Prof. Fernando Cruz e Co-Orientador: Dr. Jean Carlo Rossa Hauck

Florianópolis(SC) 2013/1

Agradecimentos

Agradeço aminha namorada Muriel,
meus avaliadores, minha família,
orientador e co-orientador e todos
aqueles que de alguma forma
contribuíram para este trabalho:
Leonardo
<leonardolinovieira@gmail.com>.

Resumo

Depois da revolução que foi o computador pessoal, vivenciamos uma nova abordagem tecnológica. É a era *dos smartphones*, *tablets* e derivados, o computador pessoal sendo cada vez mais portátil/prático de acessar pelo usuário. O que há anos atrás funcionava apenas para fazer ligações, hoje é uma poderosa máquina capaz de realizar diversas tarefas e que oferece uma grande quantidade de novas idéias e soluções a serem desenvolvidas. Motivado por esta eminente necessidade, este trabalho tem como objetivo o desenvolvimento de uma API para a plataforma para que facilite e agilize o desenvolvimento de aplicativos que utilizem formulários e envio de dados capturados e validados para outros sistemas. Por fim, pretende-se testar e demonstrar como utilizar e o ganho que a API oferece com casos práticos.

Palavras chaves: Android, API, Captura dados, Desenvolvimento aplicativos

Abstract

After the personal computer revolution, we are living a new tecnologhy moment. It is the smartphones, tablets and derived moment, the personal computer being more practical and portable than ever. Today, these devices are powerful machines capable of accomplish a lot of tasks and offers a big quantity of new ideas and solutions to be develop. Motivated by this eminent need, this paperwork has as a goal the development of an API that facilitates and accelerates apps development which use forms and send captured and validated data to other systems. At last, it is intended to demonstrate how to use it and the gain of using it with practical cases.

Key Words: Android, API, Data Capture, App Development

Lista de tabelas

Lista de abreviaturas e siglas

S.O	Sistema Operacional
SDK	Kit Desenvolvimento de Software
API	Application Programming Interface
Req	Requisito
JSON	Java Script Object Notation
SWT	Standard Widget Toolkit
UML	Unified Modeling Language
UI	User Interface

Lista de Figuras

52	Formulário Concluído por Usuário
53	Mensagem de resposta do servidor
54	web.xml projeto CadastroEqptoWEB
55	Método doPost() do servlet CadastroEpqtoServlet
56	Servidor com projeto CadastroEpqtoWEB iniciado
57	Console com Servidor iniciado
58	Console após receber dados no servlet

Sumário

1 Introdução

Depois da revolução que foi o computador pessoal, vivencia-se uma nova abordagem tecnológica. É a era *dos smartphones*, *tablets* e derivados, o computador pessoal sendo cada vez mais portátil/prático de acessar pelo usuário. O que há anos atrás funcionava apenas para fazer ligações, hoje é uma poderosa máquina capaz de realizar diversas tarefas e que oferece uma grande quantidade de novas idéias e soluções a serem implementadas.

Nos últimos anos, grandes empresas estão investindo em novos Sistemas Operacionais e plataforma para smartphones, tendo como principais Google Inc.[1] com o SO Android [2] e Apple Inc. [3], por sua vez com o iOS[4].

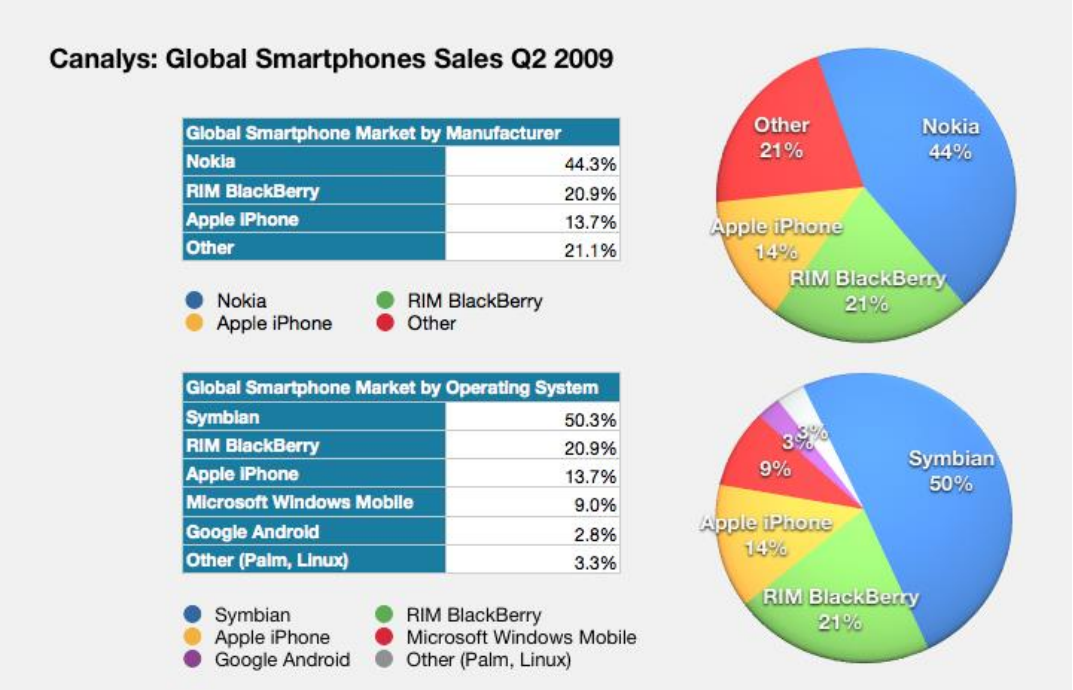

Figura 1:Participação no Mercado Mobile 2009[5]

[1] Empresa multinacional norte-americana de serviços online e software. Disponível em: <http://pt.wikipedia.org/wiki/Google>. Acesso em Dez. 2012

[2] Disponível em: <http://www.android.com/>. Acesso em Dez. 2012

[3] Empresa multinacional norte-americana que tem o objetivo de projetar e comercializar produtos eletrônicos de consumo, software de computador e computadores pessoais. Disponível em: <http://pt.wikipedia.org/wiki/Apple>. Acesso em Dez. 2012

[4] Disponível em: <http://www.apple.com/ios/ >. Acesso em Dez. 2012

[5] Disponível em: <http://appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html>. Acesso em 16 Ago. 2012.

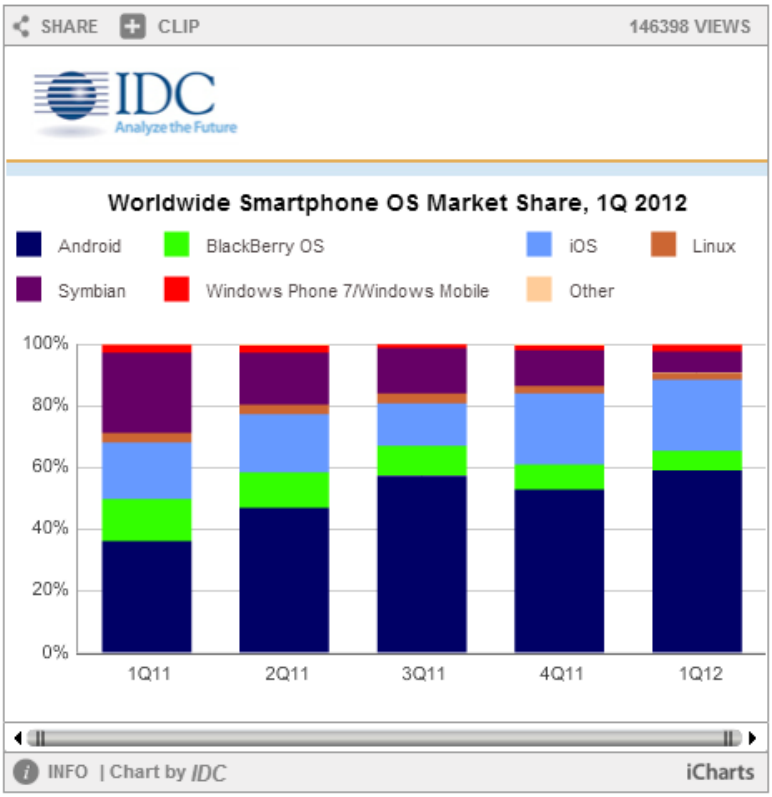

Figura 2: Participação no Mobile 2011 e 2012[6]

No desenvolvimento de sistemas normalmente é necessário criação de formulários para entradas de dados por parte de usuários e no *Android* não é diferente. As implementações de formulário normalmente seguem um padrão de *layout* e de ações que serão desenvolvidas, como *Preencher* , *Salvar Formulário*, etc. O desenvolvimento de um formulário em um sistema gera trabalho repetitivo e com grande investimento de tempo, que são alguns dos pontos que o presente trabalho visa aprimorar.

A Google fornece um kit de desenvolvimento *Android* muito completo, porém verboso[7], para os desenvolvedores implementarem suas aplicações. Para criação de interfaces de usuário, são providos componentes prontos que são definidas em um *XML* [8] e uma classe que a represente. É possível também, desenvolver interfaces sem *XML*, de maneira dinâmica, porém não é encorajado, além de ser mal documentado por parte da Google. Neste trabalho será desenvolvida uma biblioteca para auxiliar e facilitar o desenvolvimento de aplicativos que façam uso de formulários. Por consequência, será reduzido o

[6] Disponível em: < http://www.digitaltrends.com/wp-content/uploads/2012/05/idc-worldwide-smartphone-marketshare-q1-2012.png />. Acesso em: 12 Dez. 2012.
[7] No contexto, significa código de programação desnecessariamente longo.
[8] É uma linguagem de marcação. Disponível em: < http://www.tecmundo.com.br/programacao/1762-o-que-e-xml-.htm#ixzz2Y8JBFRNO/>. Acesso em: 12 Maio 2013

tempo de desenvolvimento, a quantidade de linhas de código a serem escritas e o retrabalho do desenvolvedor. Seus serviços serão invocados por uma API, que é a interface de acesso.

1.1 Objetivos

1.1.1 Objetivo Geral

Esse trabalho possui como objetivo geral, a criação de uma API para o SO Android que auxilie e acelera o desenvolvimento de aplicativo *stand-alone*[9] ou módulo móvel para sistemas existentes ou em desenvolvimento, que desejam usá-lo em sua solução.

Tal API se chamará **MobiMod** e irá prover funcionalidades para a criação de formulários, contando com validação e captura de dados, captura de fotos, criação de telas de forma fácil e dinâmica (sem a necessidade de criar arquivos no formato *XML* para layout) e proverá funcionalidade para envio de dados do módulo móvel **Y** (parte cliente) para sistema **X** (parte servidor) externo.

1.1.2 Objetivos Específicos

- Desenvolver uma API para dar suporte ao desenvolvimento de aplicativos ou módulo móvel que faça uso de formulário, com propósito de coletar dados, para o próprio sistema ou outros sistemas a serem integrados. Será desenvolvido para dispositivos que possuem sistema operacional Android, da Google, versão *2.2* ou acima;

- Implementar casos práticos utilizando a API, para fazer a validação,provar e demonstrar o funcionamento do mesmo. A comunicação entre o módulo móvel e o sistema se dará via protocolo HTTP;

- Fazer comparação entre desenvolvimento de formulário com uso da API desenvolvida e sem uso da API e demonstrar os resultados.

[9] Programas completamente auto-suficientes: para seu funcionamento não necessitam de um software auxiliar. Disponível em: <http://pt.wikipedia.org/wiki/Standalone>. Acesso em 16 Ago. 2012.

1.1.3 Justificativa

As tecnologias móveis têm despertado interesse em diversos segmentos. Cada vez mais surgem necessidades as quais aplicativos suprem de maneira eficiente para o usuário final. Simultaneamente, sabemos que grandes soluções e sistemas de informação que existem no mercado são fruto de um aglomerado de módulos específicos para realização dos requisitos.

A tecnologia móvel começa a ganhar espaço, impondo novos requisitos para novos aplicativos para usuário final ou módulo para sistemas. Alguns desses reincidentes requisitos podem ser simplificados com a criação de específicas APIs e frameworks para a área, como é o caso da *Mobimod*.

Neste trabalho será utilizada a plataforma *Android* para desenvolvimento de uma API. Como visto na figura 1, segundo (APPLE INSIDER, 2009), o *Android* em 2009 tinha menos de 3% do mercado mobile. Segundo (IDC, 2012), em 2012 já era líder no mercado, superando seu maior concorrente, o iOS da Apple (Figura 2).

Além de ser líder no mercado, a Google fornece gratuitamente o Android SDK e utiliza a linguagem Java para desenvolvimento, que já é de conhecimento do desenvolvedor. Não é necessário possuir Sistema Operacional específico na máquina de trabalho. Enquanto a Apple, segunda colocada no mercado (Figura 2), utiliza linguagem Objetive C [10], a qual não é de conhecimento do desenvolvedor e, segundo (APPLE, 2013), obriga o desenvolvedor a obter um computador da própria marca, da linha Macintosh.

A versão mínima escolhida para suportar a API será a Android 2.2 *Froyo*, que apesar de não ser a última lançada, é compatível com aparelhos mais antigos, aumentando assim a compatibilidade com o mercado. Além disso, provê todas funcionalidades necessárias para desenvolvimento da API.

[10] Disponível em: <http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/Introduction.html >. Acesso em 2 Jul. 2013.

2 Fundamentação teórica

2.1 Biblioteca, API e Framework

Existem algumas diferentes formas de reuso de código, como *componentes*, *biblioteca* e *frameworks. Biblioteca* refere-se a um conjunto de classes, que podem ser utilizadas separadamente, onde cabe ao usuário da Biblioteca estabelecer a ligação entre elas. Normalmente fornecem serviços comuns, como tratamento de arquivos, estrutura de dados, funções específicas. Estes serviços são disponibilizados e acessados por desenvolvedores de software através da API[11] da biblioteca.

"Um *Framework* é uma estrutura de classes inter-relacionadas, que corresponde a uma implementação incompleta para um conjunto de aplicações de um domínio"(JOH, 92). Já segundo (VILIJAMAA, 2001):

- *Ao utilizar frameworks, a ideia é reutilizar toda a arquitetura do sistema e não somente classes individuais.*
- *Frameworks utilizam o principio de Hollywood (Não nos chame, nós chamaremos você), onde o próprio framework é responsável por manter o controle do fluxo e da aplicação, ao contrário das bibliotecas, onde a aplicação é responsável por manter o fluxo.*

Nesse trabalho foi escolhido desenvolver uma API, pois esta forma de reuso oferece ao desenvolvedor flexibilidade e menor complexidade para o mesmo implementar requisitos repetitivos. Além disso, dá ao desenvolvedor a possibilidade de escolher em que momentos fará uso da solução, já que é ele quem controla o fluxo da aplicação.

2.2 Ferramentas de ambiente desenvolvimento

2.2.1 Eclipse

Eclipse é um IDE desenvolvido em Java, seguindo o modelo open source (código aberto) de desenvolvimento de software. Segundo (ECLIPSE, 2012), o projeto Eclipse foi iniciado em 2001 na IBM que desenvolveu a primeira versão do produto e doou-o como software livre para a comunidade. Em 2004 foi criado o Eclipse Foundation, que se responsabilizou pela

[11] API é o acrônimo de *Application Programming Interface* ou, em português, Interface de Programação de Aplicativos. Disponível em: <http://www.tecmundo.com.br/programacao/1807-o-que-e-api-.htm#ixzz2Xx3jqS78 >. Acesso em 10 Maio. 2013.

continuação do projeto. Hoje, o Eclipse é mantido por organizações do setor de software.

Possui como características marcantes o uso da SWT e não do Swing como biblioteca gráfica, a forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plug-ins que procuram atender as diferentes necessidades de diferentes programadores.

2.2.2 Android Development Tools (ADT)

Android Development Tools (ADT) é um plugin para o Eclipse IDE que é projetado para fornecer um ambiente poderoso, integrado, para criação de aplicativos Android.

"O ADT amplia os recursos do Eclipse para possibilitar a criação de projetos Android, criar e testar interface de aplicativo, adicionar pacotes baseados no Android Framework API, depurar seus aplicativos usando as ferramentas do Android SDK, e até mesmo exportar arquivos .Apk (formato de pacote de arquivos Android para distribuição) assinados (ou não assinado) a fim de distribuir a sua aplicação" (DEVELOPERS,2012).

Desenvolver no Eclipse com ADT é altamente recomendado pela Google, é o caminho mais rápido para começar. Com a configuração do projeto guiada proporciona, bem como ferramentas de integração, os editores *XML* personalizados, emulador de dispositivos e painel de saída de depuração.

2.2.3 Apache Subversion (SVN)

É um sistema multiusuário de controle de versão open-source desenvolvida pela fundação Apache, que permite aos usuários armazenar qualquer tipo de documento. Qualquer alteração em um documento é registrada e armazenada num repositório, onde é mantido em histórico, o que possibilita aos usuários retornarem à versões anteriores em qualquer momento.

O SVN, também possibilita que mais de um usuário acesse um arquivo ao mesmo tempo, ou seja, faz o controle de concorrência, gerenciando a sua atualização em forma de versão. Para esse projeto, será utilizado um plugin do SVN para Eclipse chamado subclipse, que simplifica o uso desta ferramenta.

2.2.4 Visual Paradigmn (Community Edition)

O Visual Paradigm Community Edition, é uma ferramenta para modelagem de software utilizando UML versão 2.x. Com esta ferramenta, é possível modelar 13 tipos de diagramas:

- Diagrama de Classes
- Diagrama Casos de Uso
- Diagrama de Sequencia
- Diagrama de Comunicação
- Diagrama de Estados
- Diagrama de Atividades
- Diagrama de Componentes
- Diagrama de Deployment
- Diagrama de Pacotes
- Diagrama de Objetos
- Composite structure diagram
- Diagrama de Tempo
- Diagrama de Interação

Neste trabalho serão utilizados os diagramas de: Caso de Uso, Classes, Atividades, Sequência e Interação. Tais diagramas foram escolhidos por relevância em relação ao escopo do projeto. Além disso, segundo (AGILE MODELING, 2013), os diagramas de maior prioridade[12] constam no grupo escolhido.

2.3 Tecnologias Envolvidas

2.3.1 Protocolo HTTP

O **Hypertext Transfer Protocol (HTTP**) - *Protocolo de Transferência de Hipertexto* - é um protocolo de comunicação utilizado para sistemas de informação de hipermedia distribuídos e colaborativos. Seu uso para a obtenção de recursos interligados levou ao estabelecimento da World Wide Web. “O Consórcio World Wide Web (W3C) é um consórcio internacional no qual organizações filiadas, uma equipe em tempo integral e o público trabalham

[12] Disponível em: <http://www.agilemodeling.com/essays/umlDiagrams.htm >. Acesso em 2 Jul. 2013.

juntos para desenvolver padrões para a Web" (W3C, 2013). Tal consórcio é o responsável por manter o protocolo HTTP.

2.3.2 Java Servlet

Java Servlet é um componente como um servidor, que gera dados *HTML* e *XML* para a camada de apresentação de uma aplicação Web. É basicamente uma classe na linguagem de programação Java que dinamicamente processa requisições e respostas, proporcionando dessa maneira novos recursos aos servidores. A definição mais usada considera-os extensões de servidores. Em outras palavras, Servlet é um módulo que estende a funcionalidade de um servidor Web, através de módulos de aplicação implementados em Java.

"Essa tecnologia disponibiliza ao programador uma interface para o servidor web (ou servidor de aplicação), através de uma API. As aplicações baseadas no Servlet geram conteúdo dinâmico (normalmente HTML) e interagem com os clientes, utilizando o modelo *request/response*. Os servlets normalmente utilizam o protocolo HTTP, apesar de não serem restritos a ele. Um Servlet necessita de um container Web para ser executado"(WIKIPEDIA, 2013).

2.3.3 JSON

Segundo (WIKIPEDIA, 2013), JSON, um acrônimo para JavaScript Object Notation, é um formato leve para intercâmbio de dados computacionais. JSON é um subconjunto da notação de objeto de JavaScript, mas seu uso não requer JavaScript exclusivamente. A simplicidade de JSON tem resultado em seu uso difundido, especialmente como uma alternativa para *XML* em AJAX. Uma das vantagens reivindicadas de JSON sobre *XML* como um formato para intercâmbio de dados neste contexto, é o fato de ser muito mais fácil escrever um analisador JSON.

Existem diversas bibliotecas para trabalhar com JSON e Java. Neste projeto será usada a biblioteca *org.json* que já se encontra no Android SDK nativo.

2.3.4 Sistema Operacional Android

Desenvolvido pela Google - principalmente - e *Open Handset Alliance*, o Android é um sistema operacional baseado em núcleo Linux, além de ferramentas e aplicações voltadas para funções usualmente embarcadas em dispositivos móveis. Através da máquina virtual Dalvik, o sistema executa aplicações escritas em Java. No entanto, esse ambiente usa a sintaxe Java, mas não é propriamente uma máquina virtual Java. Dessa forma, não se pode

dizer que a plataforma é compatível com qualquer padrão Java estabelecido pela Sun Microsystems, como Java ME ou SE. Também é possível escrever bibliotecas em C e outras linguagens que possam ser compiladas para código nativo ARM e instaladas utilizando o kit de desenvolvimento para Android.

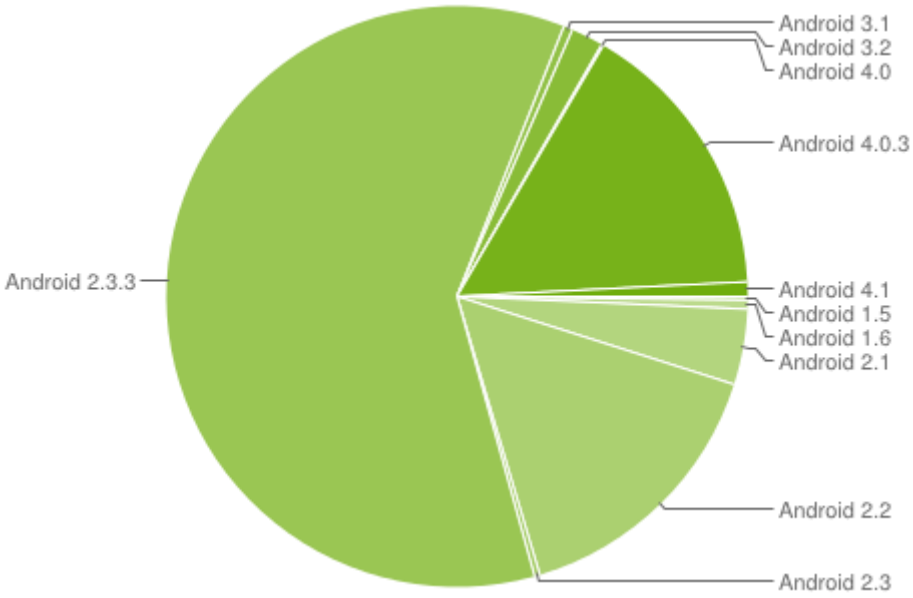

Figura 3 – Gráfico de Utilização de versões Android[13]

2.3.5 Linguagem Java e Máquina Virtual Dalvik

A linguagem Java teve grande ascendência no mercado por rodar em uma máquina virtual, permitindo que aplicações sejam portáveis para qualquer SO. Os arquivos com código-fonte (.java) são nativamente compilados para *bytecode* (.class) e compactados em um arquivo JAR (Java Archive, com extensão .jar) para ser executada em uma máquina virtual (JVM). Segundo Gonçalves (2007), milhões de pessoas já aprenderam essa linguagem e grandes empresas como NASA, IBM, ESPN entre outros demonstram a confiabilidade que a linguagem Java possui. Atualmente a linguagem é mantida pela Oracle Corporation. Segundo pesquisa[14] realizada pela empresa Tiobe em junho de 2013, Java é a segunda linguagem de programação mais popular do Mundo.

"A plataforma Android utiliza Java como linguagem para construir aplicações. A grande diferença do desenvolvimento nativo em Java para o

[13] Disponível em: < http://pt.wikipedia.org/wiki/Ficheiro:Android_chart.png>. Acesso em Dez. 2012

[14] Disponível em: < http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Acesso em 1 Jul. 2013

Android está na maquina virtual utilizada para rodar as aplicações. No primeiro utiliza-se a máquina virtual Java (JVM), e no segundo utiliza-se a máquina virtual Dalvik, que é otimizada para execução em dispositivos móveis. No desenvolvimento das aplicações, o código-fonte Java é compilado normalmente para *bytecode* (.class). Para rodar na Dalvik ele é convertido para o formato *Dalvik Executable* (.dex). Depois de convertidos, os arquivos .dex e todos os recursos como imagens são compactados em um único arquivo *Android Package File* (.apk), que representa a aplicação final, pronta para ser distribuída e instalada". (LECHEDA, 2011).

3 Materiais e Métodos

3.1 Materiais

Este capítulo trata das ferramentas que serão utilizadas como suporte para o desenvolvimento da solução proposta após ter sido feito um estudo abrangente das tecnologias existentes possíveis de serem utilizadas, sendo a escolha feita por terem sido eleitas as ferramentas mencionadas como as mais adequadas para este projeto.

3.1.1 Plataforma Android

3.1.1.2 Plugin ADT Android

Para desenvolver para a plataforma Android é obrigatório o uso do plugin ADT Android da Google, que possibilita a criação, validação, testes e edição das diversas classes e arquivos *XML*, tanto de configuração quanto para interface. Esse plugin será usado basicamente para configurar os arquivos necessários para aplicações android e para usar o emulador que ele fornece (AVD – Android Virtual Device).

Figura 4: Emulador Android.

3.1.1.3 Estrutura de um projeto Android

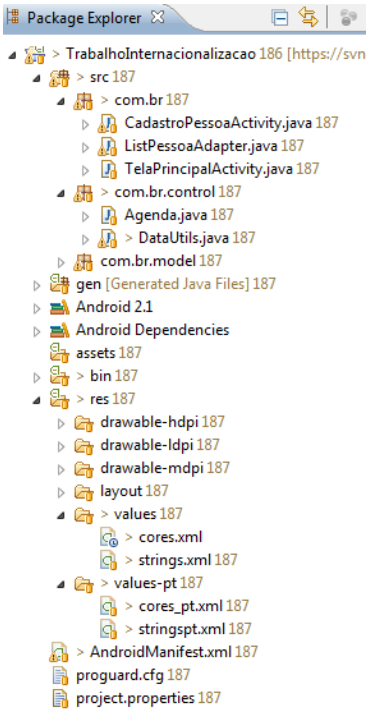

Figura 5: Estrutura de projeto Android.

3.1.1.4 Arquivo Android Manifest.xml

Todo aplicativo desenvolvido deve possuir um arquivo no diretório *root* chamado *AndroidManifest.xml*. Esse arquivo é responsável por apresentar informações essenciais para o sistema operacional Android antes de rodar qualquer código da aplicação. Conforme (DEVELOPERS, 2012), o Android Manifest:

- Define o nome do pacote da aplicação. Este nome deve ser único se for a mesma for inserida no mercado do Android, o Google Play.
- Define a versão mínima da Android API e versão do aplicativo em si.
- Determina quais processos serão definidos como *host* dos componentes.
- Declara as permissões que o aplicativo deve ter para acessar partes protegidas da API Android e para interagir com outras aplicações. Como por exemplo, acesso à internet e ao SD Card do dispositivo, pelo aplicativo desenvolvido.

- Declara também quais permissões obrigatórias outras aplicações devem possuir para interagir com os componentes da aplicação.
- Descreve todos os componentes – Serviços, *Activities,* Provedores de conteúdo que compõem a aplicação – nomeando as classes que implementam cada componente e definindo quando cada classe deverá ser instanciada. Com isso o a aplicação possui informação de como cada componente irá se comportar e em quais condições eles são iniciados.

3.1.1.5 Activity

Segundo (LECHETA, 2011), a classe *android.app.Activity* representa basicamente uma tela da aplicação. Uma tela é composta de vários elementos visuais, os quais na API nativa do Android são representados pela classe *android.view.View*.

Segundo (DEVELOPERS, 2012), uma *Activity* é um componente da aplicação que é definido para interação com o usuário, por exemplo: um formulário, tirar uma foto, enviar e-mail, visualizar um mapa. No desenvolvimento de uma interface de interação com usuário, obrigatoriamente deve-se herdar a classe *Activity* ou alguma subclasse desta. Cada *Activity*, como visto anteriormente, deve ser declarada no arquivo *AndroidManifest.xml.*

3.1.1.6 View

Segundo (LECHETA,2012), a classe *android.view.View* é a superclasse de todos os componentes visuais disponíveis na API do Android. Existem dois tipos de componentes, os componentes simples, que são os elementos de interface como Botões, Campos de Texto, Campo de exibição de texto que herdam diretamente a classe *View*. O outro tipo são os gerenciadores de layout de tela, os quais herdam uma subclasse de *View* chamada de *ViewGroup*.

3.1.2 Dispositivo SmartPhone com Android

Além dos testes feitos no emulador do Android, serão feitos testes mais realísticos fazendo uso do dispositivo Samsung Galaxy SII (S 2).

Figura 6: Dispositivo Samsung Galaxy SII[15]

O dispositivo em questão foi apontado por muito tempo como um dos melhores Smartphones do mercado[16]. Possui as características necessárias para testar e validar o uso da API.

3.2 Métodos

Este subcapítulo mostra como utilizar algumas das tecnologias abordadas no escopo deste trabalho, para melhor compreensão. Foi feita uma pesquisa explicativa – experimental em cima destas tecnologias.

[15] Disponível em: < http://www.mysmartprice.com/mobile/samsung-galaxy-s2-i9100-msp1170>. Acesso em 16 Fev. 2013.

[16] Diversas críticas de especialistas mostraram esta constatação. Disponível em:< http://www.theverge.com/2011/10/4/2470276/iphone-4s-vs-iphone-4-vs-samsung-galaxy-s-ii-by-the-numbers>. Acesso em 13 Abril 2013.

3.2.1 AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.contraprovacadeqpto"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="8" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainActivity"
            android:configChanges="orientation|keyboardHidden|navigation|screenLayout|uiMode|touchscreen"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".TelaActivity"
            android:configChanges="orientation|keyboardHidden|navigation|screenLayout|uiMode|touchscreen"
            android:screenOrientation="portrait">
        </activity>
    </application>

</manifest>
```

Figura 7: Exemplo de AndroidManifest.xml

A figura 7 mostra um exemplo do arquivo AndroidManifest.xml. “Esse arquivo é responsável por apresentar informações essenciais para o sistema operacional Android antes de rodar qualquer código da aplicação” (DEVELOPERS, 2012).

Na notação <*uses-sdk*> são informadas: a mínima versão que os dispositivos devem ter do SO Android para poderem suportar a aplicação e a versão-alvo também. Dentro da última notação <*activity*> é declarada a activity exemplo, *TelaActivity.java* e definidas propriedades como a orientação de tela suportada, neste caso apenas *portrait*[17].

3.2.2 Interface Gráfica Android

Segundo (DEVELOPERS,2012), existem duas formas de desenvolver a interface com usuário:

- **Dinamicamente:** Onde a interface é desenvolvida e construída diretamente no código-fonte java, sem arquivos xml.
- **Com *XML*:** A interface é construída através de arquivos *XML* que seguem uma estrutura de árvore de componentes. Além do arquivo *XML*, utiliza-se também uma classe java correspondente.

[17] Orientação onde o dispositivo fica na posição vertical.

3.2.2.1 Arquivo xml – tela_cadastro.xml

Segundo (DEVELOPERS, 2013), android provê um vocabulário *XML* correspondente à subclasses de *View* e *ViewGroup* para o desenvolvedor definir sua *UI* em *XML*, utilizando a hierarquia de elementos *UI*[18].

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

    <LinearLayout
        android:id="@+id/enviarCancelarPainel"
        android:layout_width="fill_parent"
        android:layout_height="100dp"
        android:padding="5dp"
        android:orientation="horizontal" >
        <Button
           android:id="@+id/okButton"
           android:layout_height="50dp"
           android:layout_width="100dp"
           android:text="Ok">
        </Button>
         <Button
           android:id="@+id/cancelarButton"
           android:layout_height="50dp"
           android:layout_width="100dp"
           android:layout_gravity="right"
           android:text="Cancelar">
        </Button>

    </LinearLayout>
```

```
<EditText
   android:id="@+id/nomeField"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="Nome Completo"
   android:maxLength="40"
   android:focusableInTouchMode="true"
   android:inputType="textCapWords"
   android:background="@android:drawable/editbox_background"/>

<EditText
   android:id="@+id/numSerieField"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="Num Serie"
   android:maxLength="5"
   android:inputType="numberDecimal"
   android:background="@android:drawable/editbox_background"/>

<Spinner
    android:id="@+id/categoriaSpinner"
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    android:prompt="@string/categoria_prompt"/>
 <Button
  android:id="@+id/dataNascButton"
  android:layout_height="50dp"
  android:layout_width="fill_parent"
  android:layout_gravity="center"
  android:text="Data Compra">
</Button>
```

Figura 8 Arquivo tela_cadastro.xml

Na figura 8 acima, é exemplificado a criação estrutural em *XML* de uma tela de cadastro com componentes das classes filhas de *View*, organizadas com um Layout específico, no caso o *LinearLayout.* Com a API desenvolvida neste trabalho, a necessidade de criar este tipo de arquivo será eliminada.

3.2.2.2 Classe Activity – TelaActivity.java

Segundo (LECHETA, 2012), a classe *android.app.Activity* representa basicamente uma tela da aplicação. Uma tela é composta de vários elementos visuais, os quais na API nativa do Android são representados pela classe *android.view.View*.

[18] Disponível em: < http://developer.android.com/training/basics/firstapp/building-ui.html>. Acesso em 16 Fev. 2013.

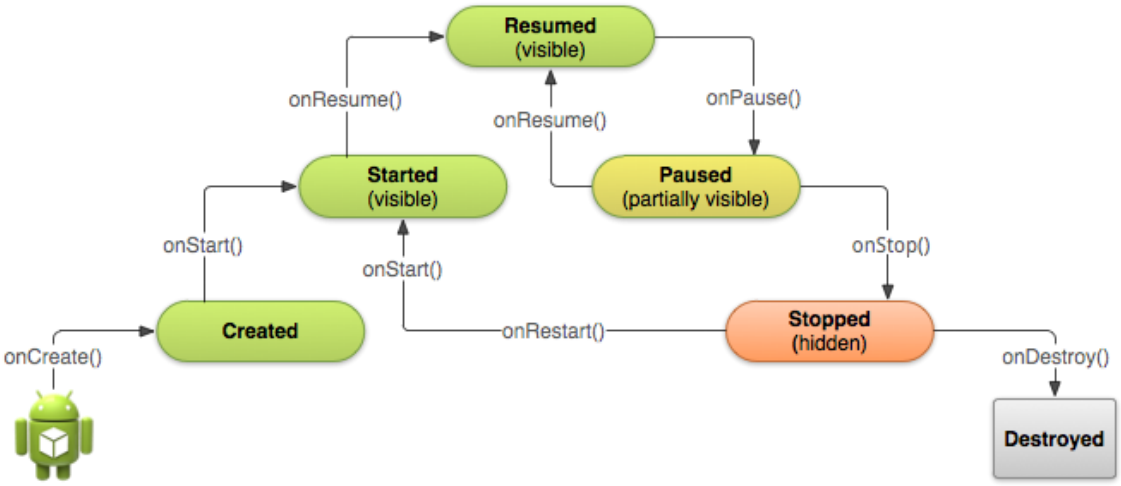

Figura 9 Estados de Activity[19]

```
public class TelaActivity extends Activity{
    protected LinearLayout painelEnviarCancelar;
    protected EditText nomeEt, numEt;
    protected Button okButton, cancelarButton,dataNasc;
    protected Spinner spinner;
    private static final int DATE_DIALOG_ID = 0;
     protected int mYear,mMonth,mDay;
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.tela_cadastro);
        painelEnviarCancelar = (LinearLayout) findViewById(R.id.enviarCancelarPainel);
        okButton = (Button) findViewById(R.id.okButton);
        dataNasc = (Button) findViewById(R.id.dataNascButton);
        cancelarButton = (Button) findViewById(R.id.cancelarButton);
        nomeEt = (EditText) findViewById(R.id.nomeField);
        numEt = (EditText) findViewById(R.id.numSerieField);
        iniciaSpinner();
```

Figura 10 Classe TelaActivity.java

Na figura 9, são mostrados todos estados de uma activity, por motivo de contextualização. Segundo (DEVELOPERS, 2013), dependendo da complexidade da activity, é provável que não seja necessário implementar todos estados.

O método onCreate() deve ser sempre implementado e dentro dele, deve fazer-se a ligação com o arquivo .xml correspondente. Na figura 10 acima, nota-se que a classe em questão estende a classe Activity. Além disso, na linha *setContentView(R.layout.tela_cadastro)* está sendo feita a ligação entre a classe *TelaActivity.java* e o arquivo *tela_cadastro.xml*. Nas linhas subsequentes, são referenciados e armazenados em instâncias java alguns dos componentes de tela declarados do arquivo *tela_cadastro.xml*.

[19] Disponível em: < http://developer.android.com/images/training/basics/basic-lifecycle.png >. Acesso em Jul. 2013

Figura 11 Tela Pronta

3.2.2.3 Comparação - Desenvolvimento de interface

Dinamicamente

```
TextView tvDataLabel = new TextView(this);
tvDataLabel.setText(getString(R.string.data_avaliacao));
tvDataLabel.setTextSize(tamanhoFonteData);
```

Com XML

```
<TextView
        android:textSize="16sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="data"/>
```

Figura 12 Comparação - Desenvolvimento Interface

Na figura 12 acima, pode-se ver o componente *TextView*, subclasse da classe *View*, sendo declarado e estruturado nas duas possíveis formas.

Para atingir os objetivos propostos, a parte da API responsável por criar telas de interface gráfica fará uso da forma dinâmica, pois mantendo esta parte na estrutura java ganha-se facilidade em manipulação e encapsulamento [20].

[20] Significa separar o programa em partes, o mais isoladas possível. Ganha-se flexibilidade.

3.2.3 Servlet

```
  <servlet>
    <description></description>
    <display-name>CadastroEqptoServlet</display-name>
    <servlet-name>CadastroEqptoServlet</servlet-name>
    <servlet-class>com.example.servlet.CadastroEqptoServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>CadastroEqptoServlet</servlet-name>
    <url-pattern>/CadastroEqptoServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Figura 13 Web.xml exemplo

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String pDados = request.getParameter("dados_capturados");
    System.out.println(pDados);

    ServletOutputStream out = response.getOutputStream();
    out.write(pDados.getBytes());
    out.flush();
}
```

Figura 14 Método doPost() exmplo

Na figura 13 , o arquivo web.xml especifica o servlet CadastroEqptoServlet que recebe os dados do formulário. A figura 14 mostra o método doPost() do servlet, o qual – na linha *request.getParameter("dados_capturados")* - recebe dados enviados por uma requisição HTTP. Além disso é possível enviar uma resposta:

ServletOutputStream out = response.getOutputStream();
out.write(pDados.getBytes());
out.flush();

No trecho acima, o *servlet* está mandando uma resposta ao sistema que enviou a requisição.

4 Especificação

Nesta etapa é apresentada a especificação da API.

4.1 Requisitos

Os requisitos listados neste subcapítulo foram baseados em experiências profissionais e acadêmicas do autor na área de pesquisa e desenvolvimento de software. Acredita-se que tais requisitos cumprem de forma genérica grande parte das necessidades ao se criar um formulário.

4.1.2 Requisitos não funcionais

1. A API deverá ser desenvolvida utilizando a linguagem Java.

2. A API deverá dar suporte à criação de módulo móvel para dispositivos que rodem o sistema operacional *Android* na versão 2.2 ou mais recente do Android *SDK*.

3. A API deverá ser configurada como *library*, sendo assim apenas referenciada por outros projetos.

4.1.3 Requisitos funcionais

4. Define Formulário: A API deve prover funcionalidades para o desenvolvedor (usuário da API) definir o formulário e qual componente de tela irá ser usado.

a. Componente de Texto Livre (EditText);

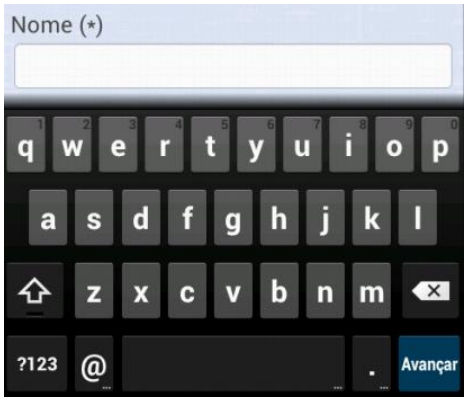

Figura 15: Exemplo EditText

b. Componente de Números

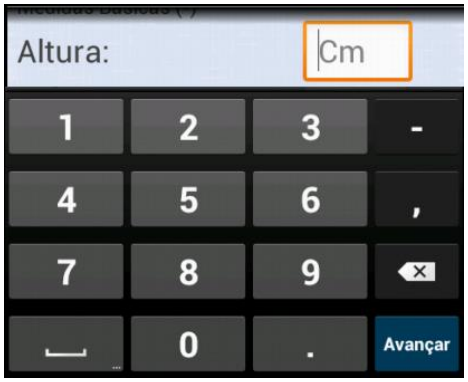

Figura 16: Exemplo EditText - Numeral

c. Componente de Data (DatePicker);

Figura 17: Exemplo DatePicker

d. Componente de Seleção (Spinner);

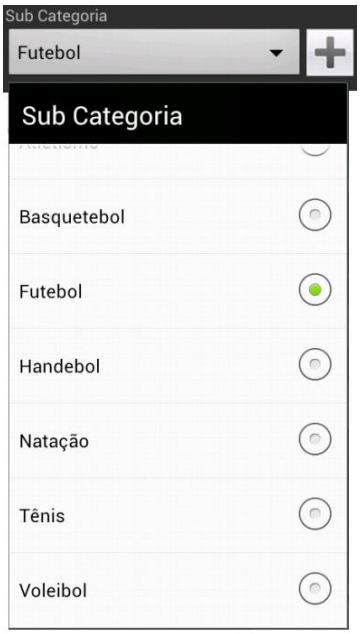

Figura 18: Exemplo Spinner

e. Componente Captura foto;

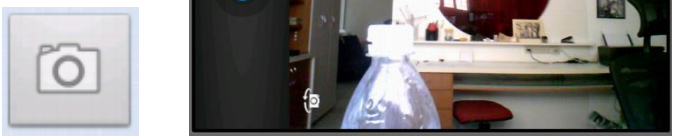

Figura 19: Exemplo Captura Foto

5. Monta Tela: A API deve ser responsável por montar a tela da interface gráfica de maneira dinâmica a partir das informações passadas pelo desenvolvedor.

6. Validação de Campos: A API deve ser responsável por fazer validação dos dados inseridos pelo usuário, a partir das especificações definidas pelo desenvolvedor, de acordo com req.4

7. Captura dados: A API deve capturar os dados inseridos pelo usuário final do aplicativo e liberar ao desenvolvedor.

8. Envio dados a sistemas: A API deve prover funcionalidades para simples envio dos dados capturados aos outros módulos do sistema a ser integrado.

9. Mostrar resposta do sistema: A API deve prover funcionalidades para que a resposta do sistema o qual recebeu dados seja mostrada, dando certeza que os dados recebidos no sistema são os que foram preenchidos pelo usuário.

10. Personaliza Fundo: A API deve prover temas padrões para fundo do formulário e além disto, permitir que o desenvolvedor utilize imagem própria para personalizar o fundo.

4.2 Diagramas

4.2.1 Diagramas de Casos de uso

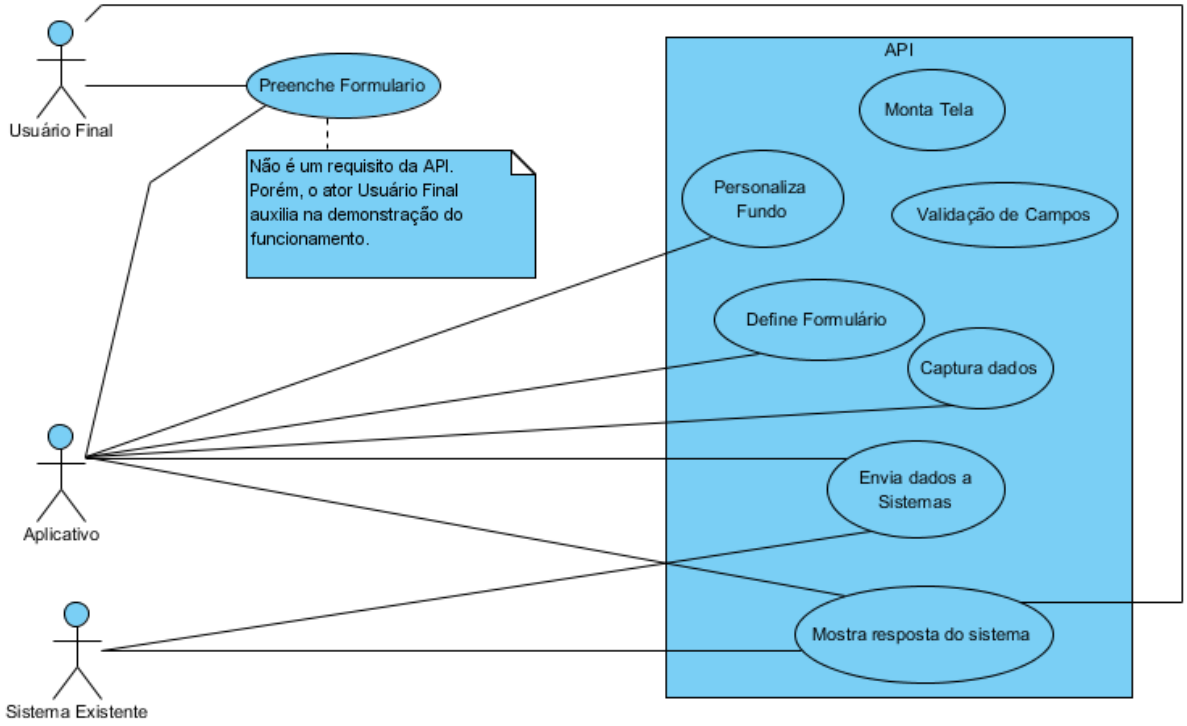

Figura 20: Diagrama de Casos de Uso

A figura 20 representa o diagrama de casos de uso do projeto como um todo. Com exceção do caso de uso *Preenche Formulário*, cada caso de uso representa um requisito funcional. O diagrama conta com a *boundary* [21] API e os seguintes atores:

- **Ator Aplicativo**: Representa o aplicativo que será desenvolvido por quem estiver utilizando a API Mobimod.
- **Ator Sistema Existente**: Representa Sistema externo o qual receberá dados capturados (via requisição HTTP) pelo aplicativo.
- **Ator Usuário Final**: Representa o usuário final e direto do aplicativo.

[21] Em UML, termo utilizado para referenciar à solução a ser desenvolvida.

4.2.2 Diagrama de Interação

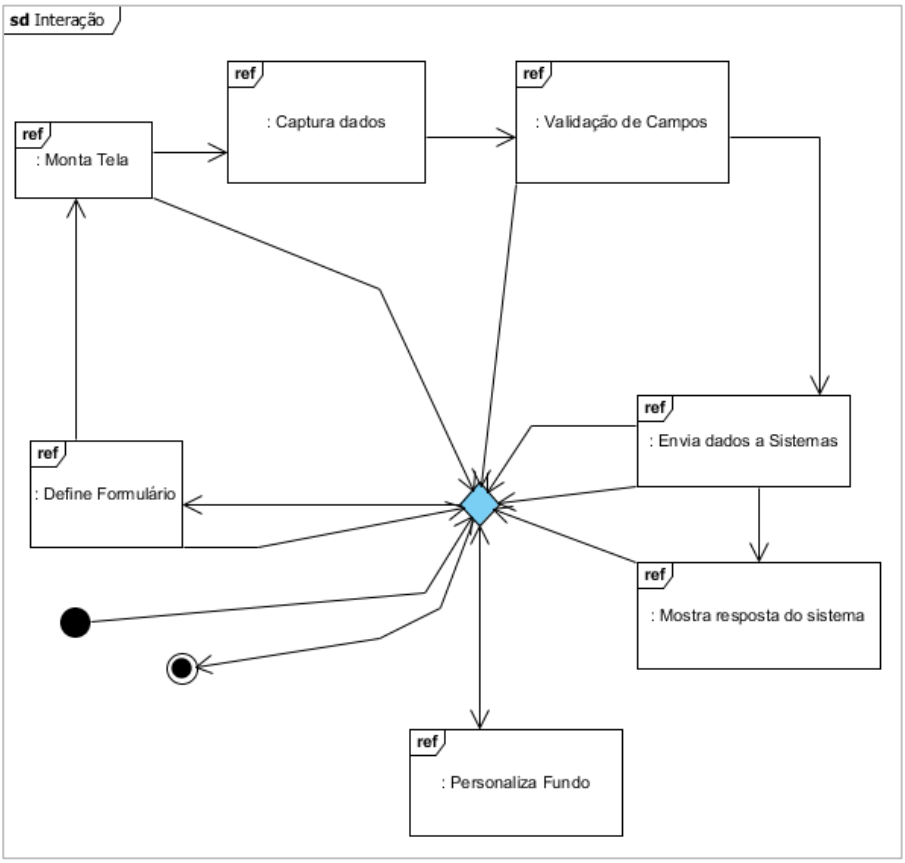

Figura 21: Diagrama de Interação

4.2.2 Diagramas de Atividades

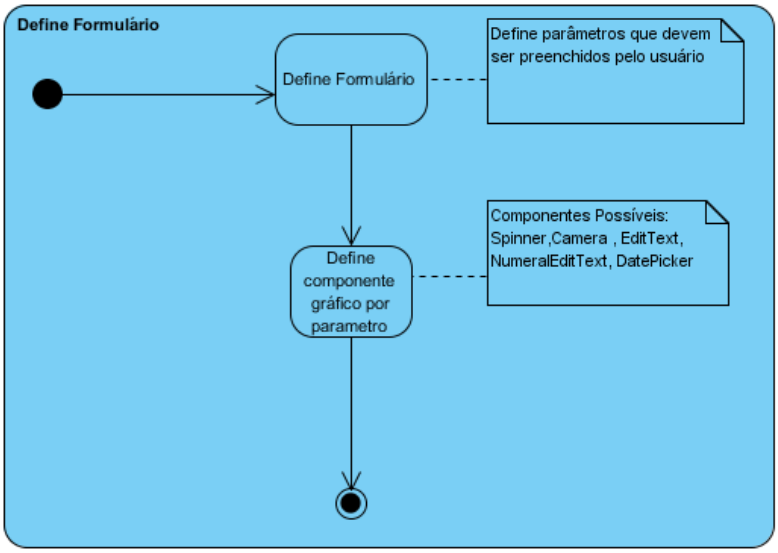

Figura 22: Diagrama Atividades – Define Formulário

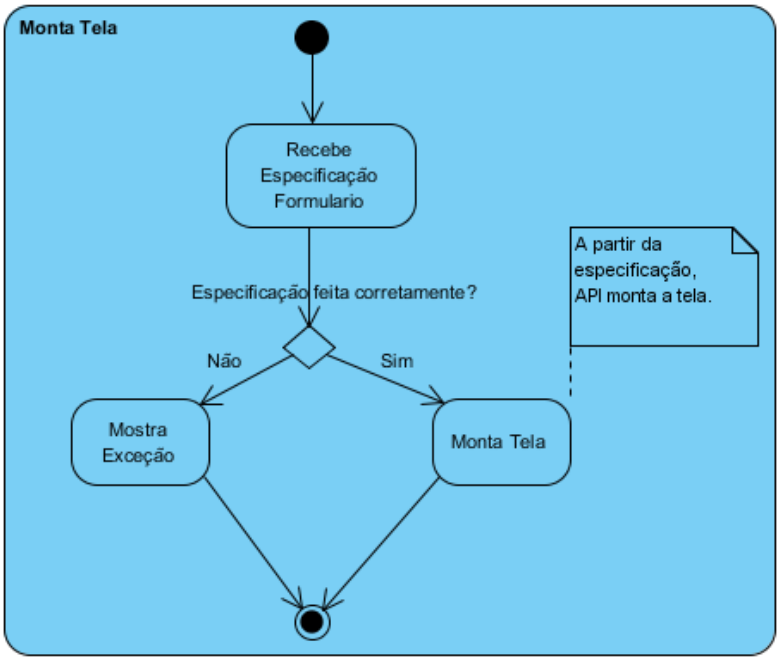

Figura 23: Diagrama Atividades – Monta Tela

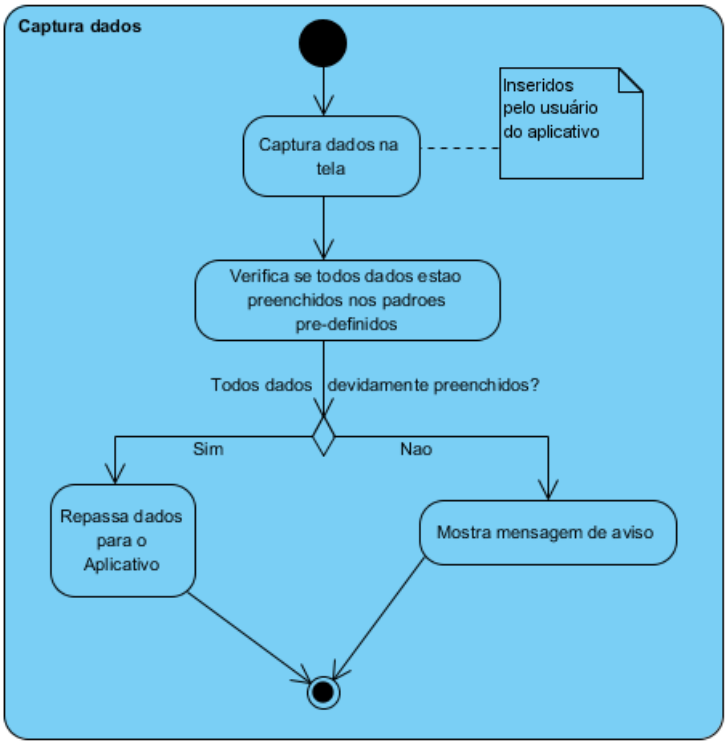

Figura 24: Diagrama Atividades – Captura Dados Inseridos

4.2.3 Diagramas de Classes

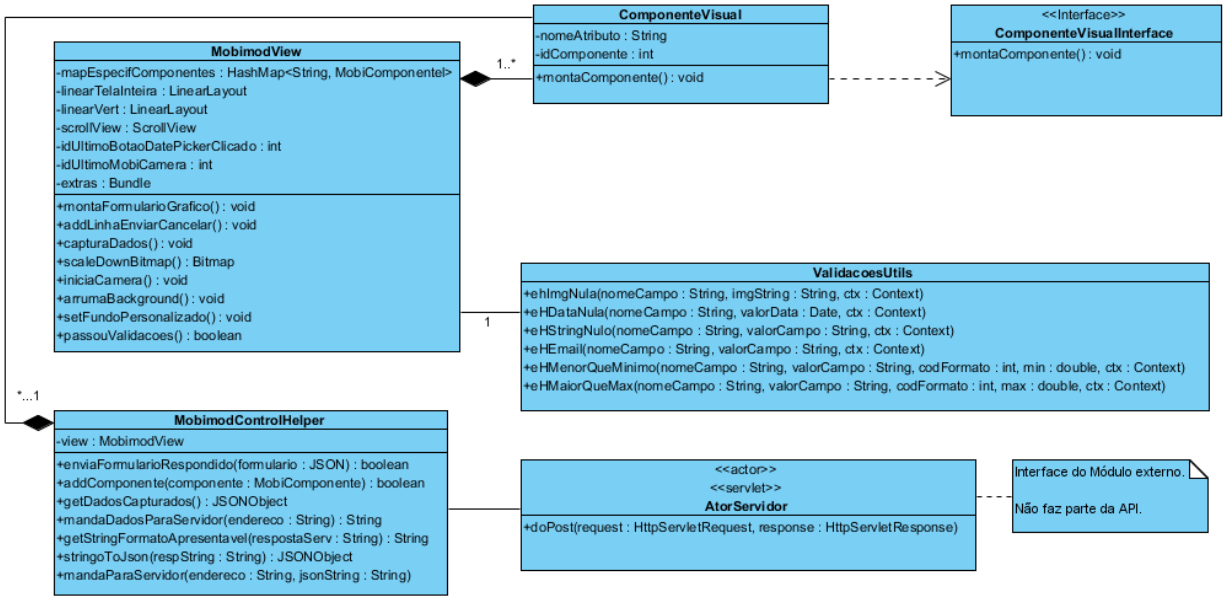

Figura 25: Diagrama de Classes – Geral API

A figura 25 acima representa as classes principais da API e como elas se relacionam.

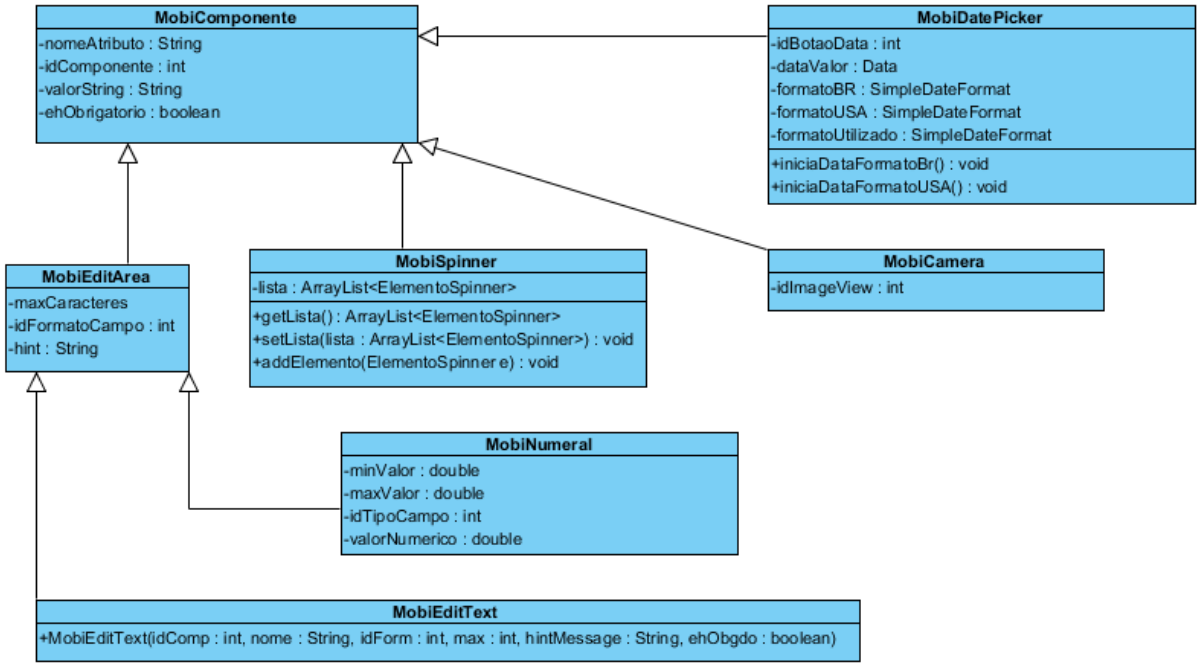

Figura 26: Diagrama de Classes – Especificação de Componentes

A figura 26 acima representa as classes de especificação de componentes que serão utilizados nos formulários. O desenvolvedor (ator aplicativo) de aplicativos interage diretamente com estas classes.

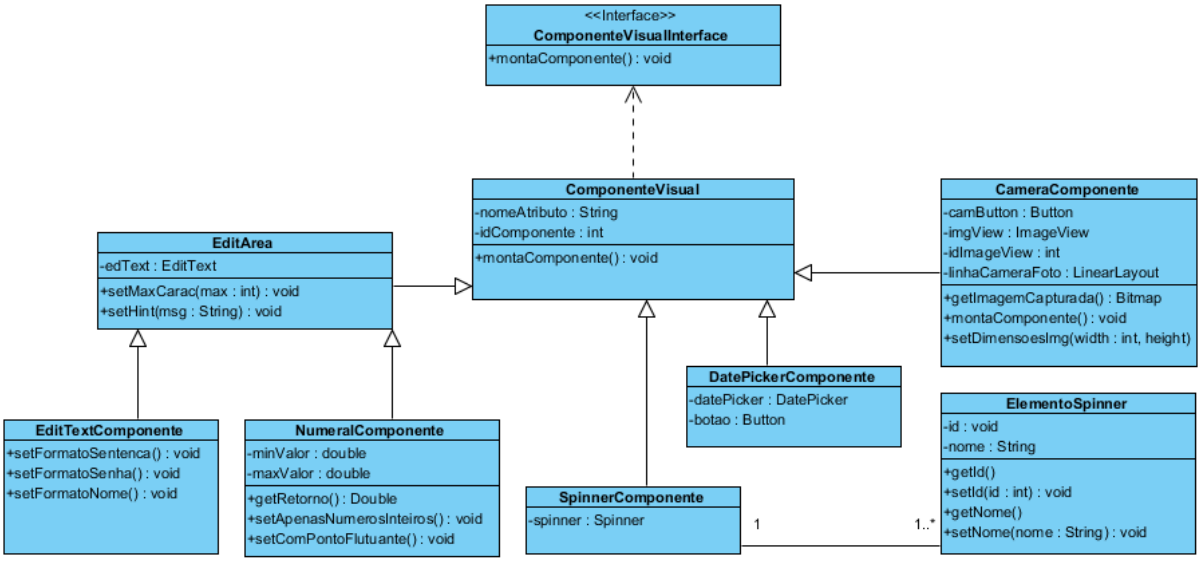

Figura 27: Diagrama de Classes – Componentes Visuais

A figura 27 acima, mostra o diagrama de classes que detalha a relação entre as classes de componentes visuais da API. O desenvolvedor de aplicativos não necessita estudar estas classes, são internas da API.

4.2.4 Diagramas de Sequência

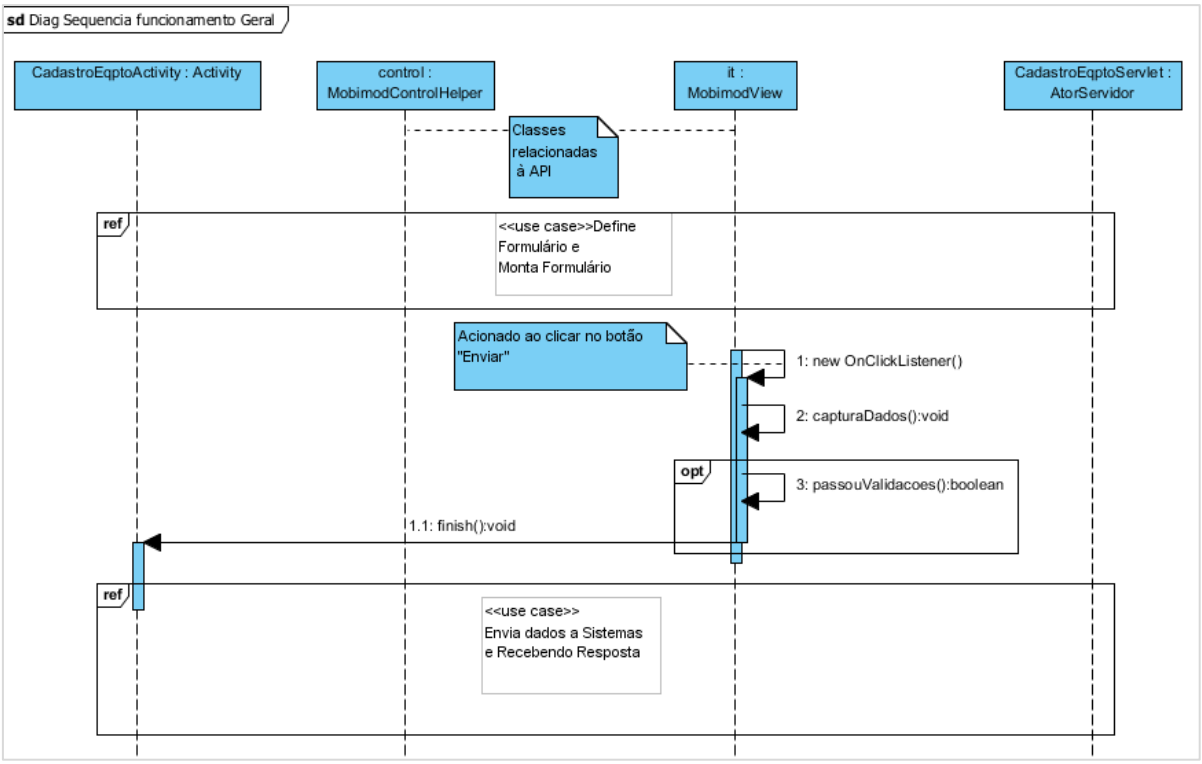

Figura 28: Diagrama Seq. Funcionamento Geral

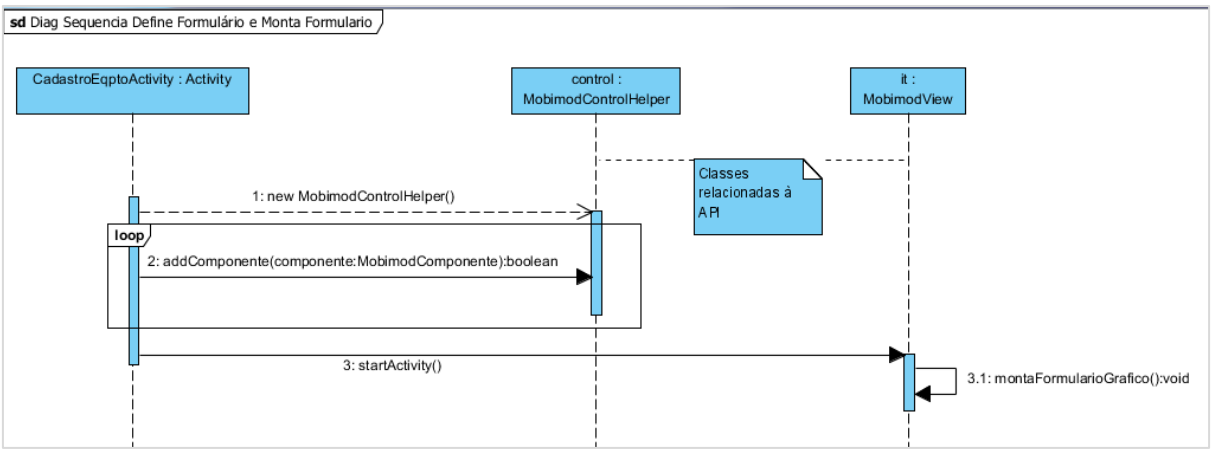

Figura 29: Diagrama Seq. Define Formulário e Monta Formulário

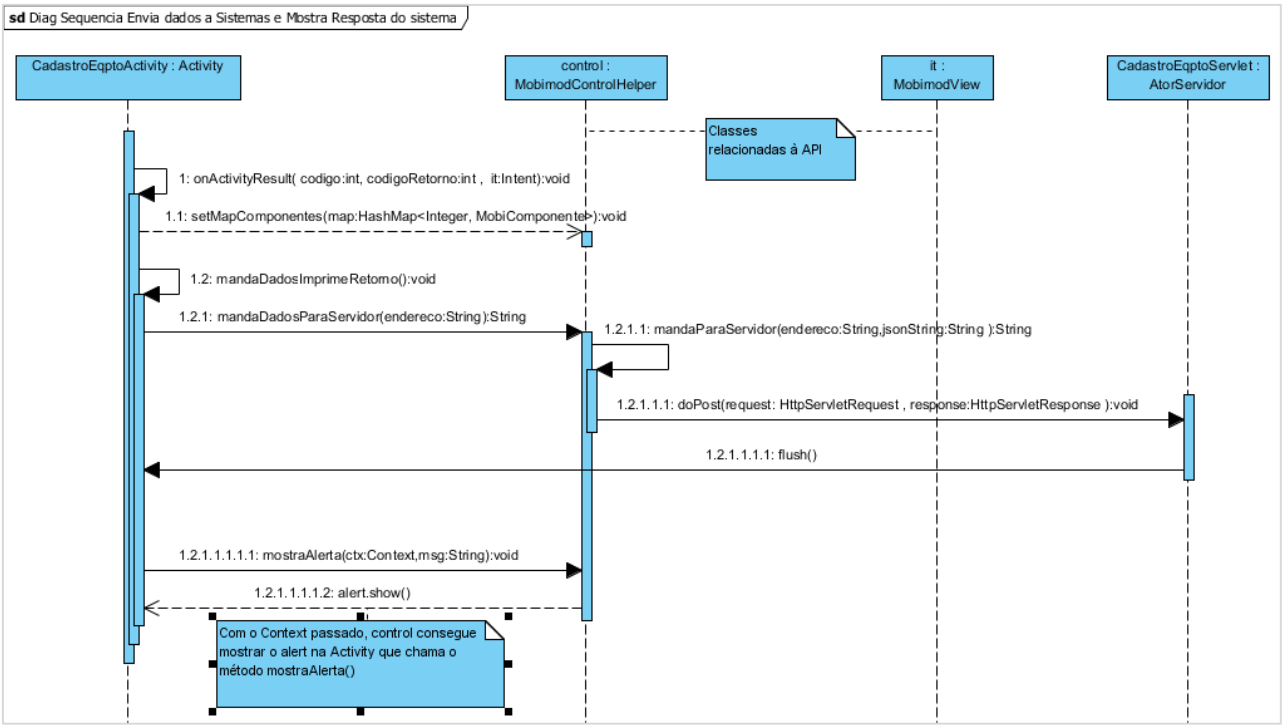

Figura 30: Diagrama Seq. Envia Dados e Mostra resposta do sistema

5 Resultado Experimental

Neste capítulo será apresentado como se deu o desenvolvimento, tanto da API *MobiMod* quanto dos exemplos beneficiados com o uso da mesma.

5.1 MobiModAPI

Para o desenvolvimento da API denominada MobiMod foi utilizada a versão Android 2.2 *Froyo*.

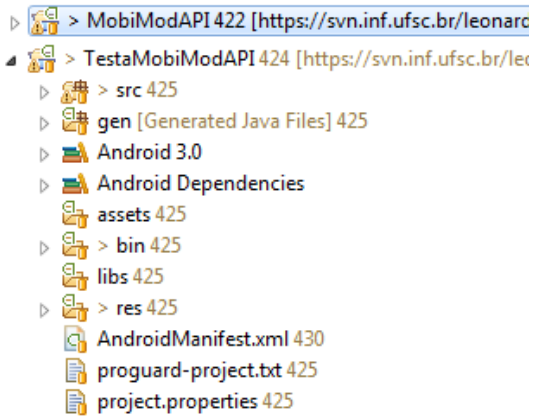

Figura 31: Project Explorer com Projetos API e Teste

5.1.1 Arquitetura

Como visto no *sub-capítulo 4.3.3 Diagramas de Classe*, a API foi desenvolvida através da abordagem orientada a objetos. Para otimização e qualidade de código foram utilizadas boas práticas, como herança de classes, abstração, atributos estáticos entre outros.

5.1.1.1 Estrutura Geral API

A sua implementação foi baseada na estrutura MVC. Separando assim classes dos tipos *modelo*, *view* e *controle* da API por *pacotes*.

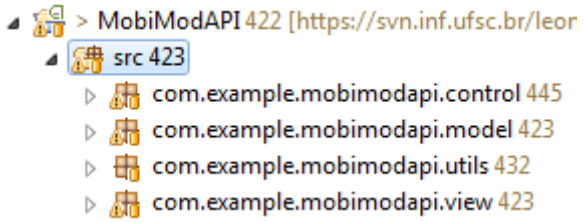

Figura 32: Project pacotes com projetos API

Além dos *pacotes* supracitados, foi adicionado o *pacote utils*, o qual detém classes encarregas por validação de campos, alertas, dentre outras utilidades específicas.

5.1.1.2 Recursos API

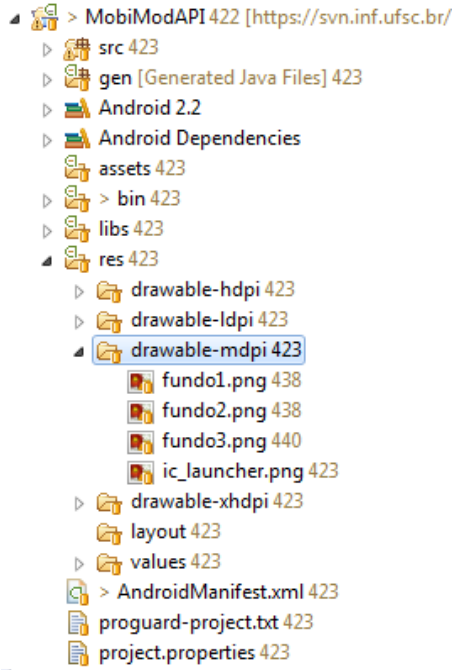

Figura 33: Imagens de Fundo

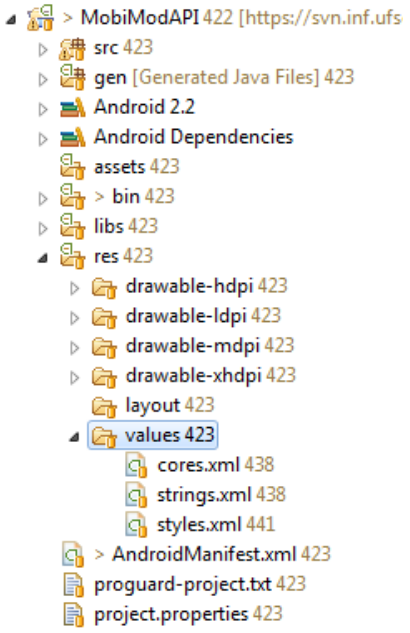

Figura 34: Cores de Fundo e Strings

Como visto na figura 33, nas pastas *drawable* foram armazenadas as imagens de fundo padrão. Estes fundos são utilizados em métodos da classe *MobimodView* que dão ao desenvolvedor alternativas de customização do seu aplicativo.

```
<resources>

    <string name="app_name">MobiModAPI</string>
    <string name="campo_email_errado">Campo \' %1$s \' deve ser corretamente preenchido.
        \n\nPadrão: endereco@email..</string>
    <string name="campo_deve_ser_preenchido">Campo \' %1$s \'
        deve ser preenchido.</string>
    <string name="imagem_nao_pode_ser_nula">Imagem \' %1$s \'
        não pode ser nula.</string>
    <string name="campo_deve_ser_maior_que">Campo \' %1$s \' deve ser maior que %2$s.</string>
    <string name="campo_deve_ser_menor_que">Campo \' %1$s \' deve ser menor que %2$s.</string>
    <string name="cad_nao_realizado">Formulário não concluído</string>
    <string name="cad_realizado">Formulário concluído</string>
    <string name="campo_obrigatorio">Campo Obrigatório</string>
</resources>
```

Figura 35 Strings.xml

Na figura 35 acima as *strings* [22] utilizadas na API foram separadas e armazenadas em um arquivo *strings.xml*, o que torna o projeto mais elegante e menos propenso a erros.

5.1.2 Principais Classes

5.1.2.1 MobiComponente e filhas

A classe pai MobiComponente e suas classes filhas (MobiEditText, MobiSpinner, MobiNumeral, MobiCamera, MobiDatePicker) são as classes utilizadas pelo desenvolvedor para especificar quais e como devem ser os componentes do formulário da figura 26.

a. **MobiEditText**: Esta classe especifica como deve ser o EditTextComponente. O desenvolvedor deve informar o *número máximo de caracteres*, a *hint* [23], tipo de campo – livre, senha, email ou sentença - de acordo com a classe *EditTextTiposEnum* e se o campo é obrigatório ou não;

b. **MobiNumeral:** Esta classe especifica como deve ser o NumeralComponente. O desenvolvedor deve informar o *valor mínimo*, *valor máximo*, a *hint, tipo de campo* – inteiro ou com ponto flutuante - de acordo com a classe *NumeralTiposEnum* e se o campo é obrigatório ou não;

c. **MobiSpinner:** Esta classe especifica como deve ser o SpinnerComponente. O desenvolvedor deve instanciar cada elemento do Spinner com a classe *ElementoSpinner* informando apenas *id* e

[22] Tipo Primitivo em Java. É uma cadeia de caracteres.
[23] Termo em inglês. Significa “Dica”.

nome(texto). Depois, instanciar o SpinnerComponente e adicionar as instancias de *ElementoSpinner* nele através do método *addElemento();*

d. **MobiDatePicker:** A classe MobiSpinner especifica como deve ser o DatePickerComponente. Desenvolvedor deve atribuir *id, nome* e informar se componente é obrigatório ou não. Por padrão, o DatePicker inicia na data atual.

e. **MobiCamera:** A classe MobiCamera especifica como deve ser a CameraComponente. Além de atribuir *id, nome,* o desenvolvedor deve definir o *id* que será utilizado para recuperar a imagem capturada pela câmera. Também deve informar se será obrigatório ou opcional o uso deste componente no formulário.

5.1.2.2 MobimodControllerHelper

A classe *MobimodControllerHelper* é a classe que representa a parte de controle da API. É nela que devem ser armazenadas as especificações de componentes. Ela é responsável por sintetizar a resposta em um objeto JSON, porém depende de uma chamada do desenvolvedor para executar esta função. As especificações supracitadas são feitas com as classes *MobiComponente*.

5.1.2.3 MobimodView

A classe *MobimodView* é responsável pela interpretação da especificação de componentes e pela adaptação em componentes visuais. Esta classe também trata personalização imagem ou cor de fundo da tela. Além disso, é responsável por acionar a validação de campos e retornar para o desenvolvedor o *Map*[24] que o mesmo enviou, porém com adição dos valores inseridos (já validados) pelo usuário.

5.1.2.4 ValidacaoUtils

A classe ValidacaoUtils é a classe responsável pela validação dos campos e componentes. Possui tratamento diferenciado para cada tipo de componente e graças à ela, o desenvolvedor fica liberado de tarefas exaustivas

[24] Estrutura de Dados

de validação. Tudo o que ele deve fazer é passar a especificação ao criar os *MobiComponentes* que o *ValidacaoUtils* se encarrega do resto.

5.1.3 Métodos Principais

a. addComponente(MobiComponente componente) – Método da classe *MobimodControllerHelper* que adiciona os componentes em uma estrutura HashMap;

b. getDadosCapturados() – Método da classe *MobimodControllerHelper* que retorna um JSONObject com as informações capturadas e já validadas;

c. montaFormularioGrafico() – Método da classe *MobimodView* que monta o formulário - a partir do Map de especificação de componentes - com auxílio do método addComponente();

d. passouValidacoes() – Método da classe *MobimodView* que avisa ao sistema se os dados inseridos ou capturados pelo usuário estão dentro dos requisitos impostos pelo desenvolvedor;

e. arrumaBackground(int cod) – Método da classe *MobimodView* que define o fundo a ser utilizado, entre as alternativas pré-definidas;

f. setFundoPersonalizado(int resource) - Método da classe MobimodView que personaliza o fundo do projeto em desenvolvimento com imagem inserida pelo desenvolvedor do aplicativo.

5.1.4 Criando um Formulário

5.1.4.1 Parte Desenvolvedor

Figura 36: Fluxo Criação Formulário

- **Instancía os MobiComponentes:** O desenvolvedor da aplicação deve instanciar todos *MobiComponentes,* que deseja utilizar no formulário em questão. Estas são estruturas que servem para especificar como devem ser os componentes e o que - que tipo de inserção feita pelo usuário - devem aceitar.

- **Adiciona MobiComponentes no Map**: Após instanciar os componentes, o desenvolvedor deve adicionar estes componentes na estrutura Map através do método *.addComponente(MobiComponente)* da Classe *MobiController*.

- **Inicia Activity:** O desenvolvedor deve iniciar uma Activity, colocando como parâmetro extra o map com as especificações de componentes

5.1.4.2 Parte API (Interface Gráfica)

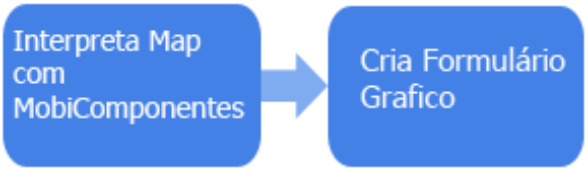

Figura 37: Fluxo Criação Formulário Gráfico

- **Interpreta Map com MobiComponentes:** A Activity é iniciada, recebe o Map de especificações dos componentes e interpreta-os.

- **Cria Formulário Gráfico**: Uma vez interpretada a especificação destes componentes, os mesmos são criadas de forma gráfica.

5.1.4.3 Parte API (Capturando dados inseridos)

Figura 38: Fluxo Captura Dados

- **Usuário Confirma Formulário**: Após inserir os dados, o usuário confirma os dados.

- **Validação dos Campos**: Baseado na especificação que foi passada pelo desenvolvedor de aplicativos, a API MobiMod percorre todos os componentes e faz a validação dos mesmos. Se tiver alguma não-conformidade, a mesma encarrega-se de mostrar um alerta ao usuário, pedindo que preencha o campo da maneira pré-estabelecida;

- **Retorna Objeto JSON com valores**: Atendendo ao *req. 7* (*Captura dados*) e *req.10* (*Envio dados à sistemas*) a activity é fechada e como retorno, o desenvolvedor recebe um Objeto JSON, contendo as informações validadas e capturadas. Se o desenvolvedor necessitar mandar tais informações para outro módulo ou sistema, basta uma requisição HTTP e enviar este objeto JSON formato em *string*.

5.1.5 Transformando Projeto em API

Segundo (DEVELOPERS, 2012):

- *Um projeto Android contém todos arquivos e recursos necessários para ser transformado em um arquivo .apk para instalação. Será necessário criar um projeto Android para cada aplicação que desejar eventualmente instalar em um aparelho.*
- *É possível designar um projeto android como uma API, que permite o compartilhamento com projetos que dependem dele. Uma vez transformada em API, não poderá ser instalado em um aparelho.*

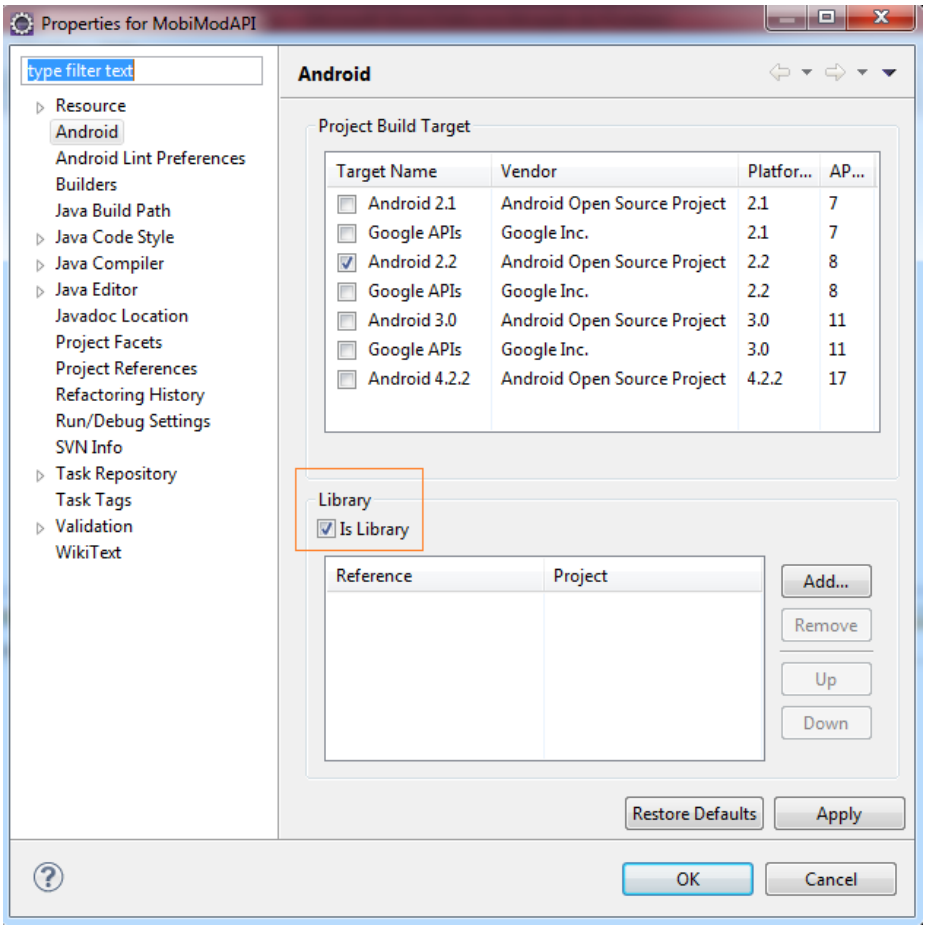

Figura 39: Transformando Projeto em API

Como demonstrado na figura 39 acima, foi necessário configurar nas propriedades do projeto MobiModAPI para então definir tal projeto como uma *Library*[25].

[25] Biblioteca em inglês.

5.2 Projetos - Experimentos e Demonstração

Este subcapítulo tem como objetivo comprovar o funcionamento da API, demonstrar como utilizar e também o ganho ao utilizá-la.

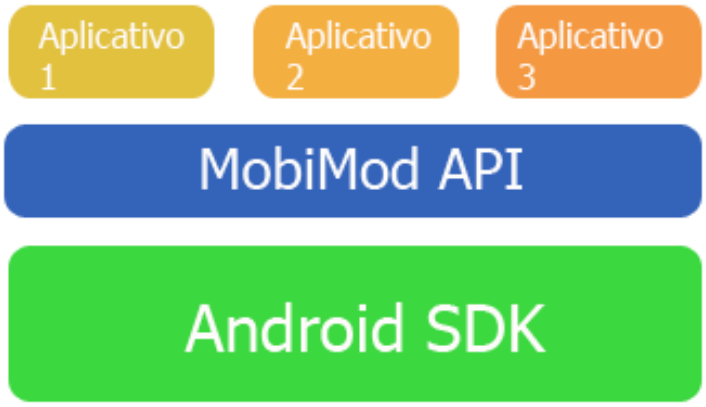

Figura 40: Visão Geral – Atuação MobiMod API em Camadas

5.2.1 Projeto CadastroFuncionario

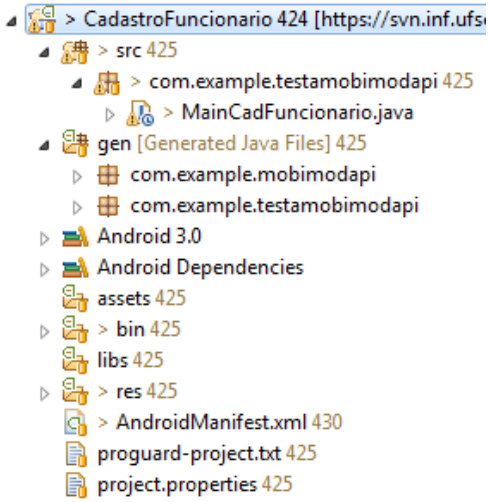

Figura 41: Projeto Explorer CadastroFuncionario

Objetivos desta demonstração

a. **Como referenciar API no projeto:** Configurar na IDE Eclipse e no arquivo AndroidManifest.xml;
b. **Mostrar validação e captura de dados:** Através de imagens, comprovar o funcionamento da validação de campos feitos pela API;

c. **Demonstrar Componentes:** EditTextComponente, DatePickerComponente, CameraComponente e SpinnerComponente.
d. **Personalização:** Mostrar a personalização do fundo do aplicativo, com o uso da API.

5.2.1.1 Referenciando MobiModAPI no Projeto

Diferente de projetos *Java* comuns, a transformação de um projeto Android em uma API, não é feita através de exportação/importação de arquivo .jar. Deve-se manter o projeto da API no workspace de desenvolvimento (Figura 31) e então referenciar esta API nos projetos dependentes dela.

5.2.1.2 Configurando referência da API no projeto

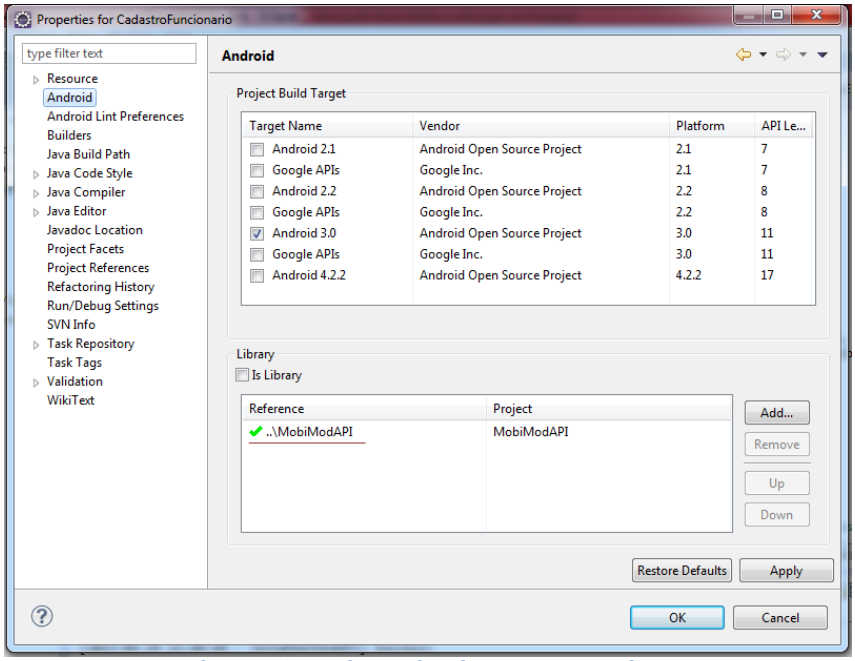

Figura 42: Referenciando a API no projeto

Como mostrado na figura 42 acima, foi necessário configurar nas propriedades do projeto *CadastroFuncionario* para referenciar o projeto MobiModAPI.

5.2.1.3 Referenciando no AndroidManifest.xml

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

    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="11" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".MainFormTeste"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity android:name="com.example.mobimodapi.view.MobimodView" />

    </application>

</manifest>
```

Figura 43: Android Manifest do Projeto CadastroFuncionario

No arquivo *AndroidManifest.xml* do projeto *CadastroFuncionario* foram adicionadas duas linhas. São elas:

- A linha <uses-permission android:name="android.permission.INTERNET"/> a qual é dá permissão ao projeto ter acesso à internet pela dispositivo aonde ele será instalado. Necessária apenas para casos onde é necessário mandar dados para sistemas externos, apesar disso, é recomendado colocar em todos projetos que dependem da API.

- A linha <activity android:name="com.example.mobimodapi.view.MobimodView"/> a qual referencia a classe MobimodView da API, responsável pela criação da interface gráfica.

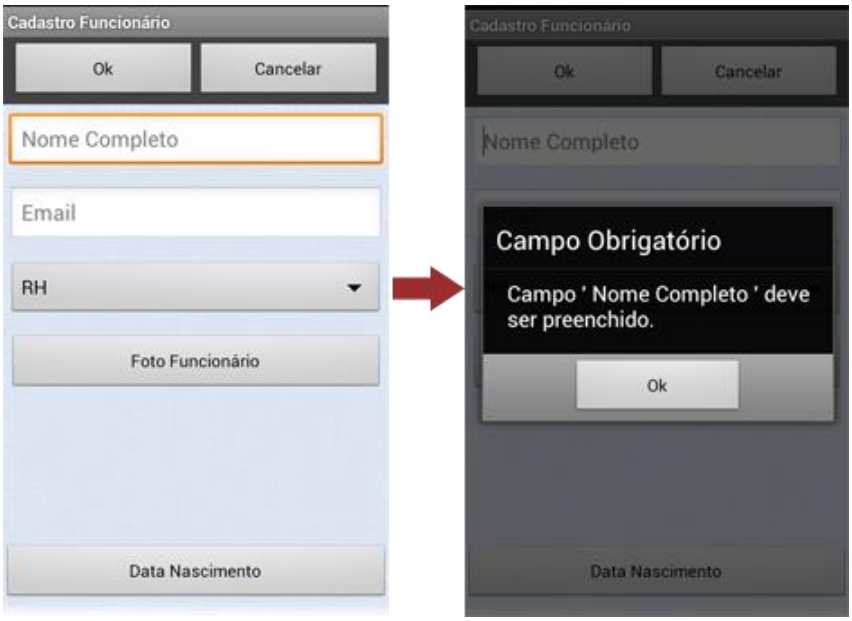

Figura 44: Validação Nome

Figura 45: Validação Data Nascimento

Nas figuras 44 e 45 acima, é demonstrado como funciona a validação. Ao clicar no botão *ok*, o usuário tentou finalizar o cadastro, porém a API impediu a ação, ao detectar que os *campos nome* e *data nascimento* não haviam sido preenchidos.

Figura 46: Validação Email

Ainda demonstrando validação, a figura 46 acima mostra que a API detectou que o campo *e-mail* estava preenchido, porém não da maneira esperada e então, não finalizou o cadastro.

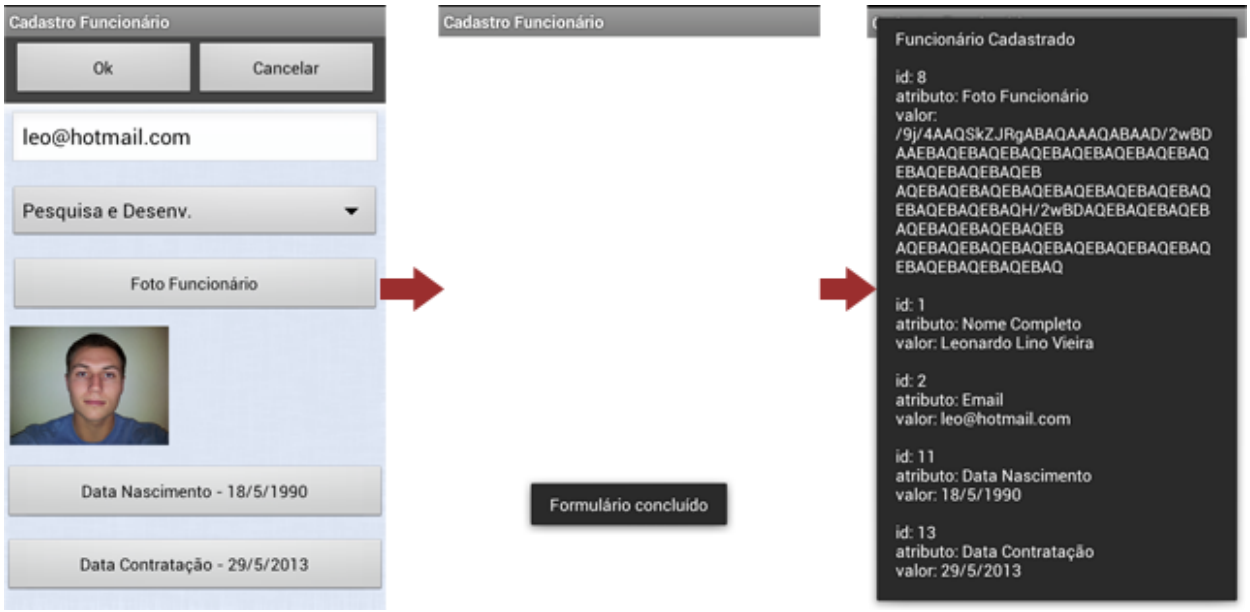

Figura 47: Formulário Concluído e Impressão Funcionário

Por fim, a figura 47 mostra o cadastro correto sendo finalizado. Na tela com cabeçalho *Funcionário Cadastrado* vemos a impressão do objeto JSON que foi capturado. Podemos ver cada item, o *id*, *nome* e *valor*. Nota-se que o valor do atributo *Foto Funcionário* armazena a imagem bitmap codificada em Base64[26].

[26] É um método para codificação de dados para transferência na internet (Codificação MIME para transferência de conteúdo).

5.2.2 Projeto CadastroEquipamento

Objetivos desta demonstração

a. **Integração:** Demonstrar código e funcionamento do API com módulos externos, a partir de requisição *HTTP* e *Servlet*;
b. **Código necessário Formulário:** Mostrar o trecho de código necessário a ser implementado pelo desenvolvedor de aplicativo para criar o formulário
c. **Demonstrar Componentes:** EditTextComponente, NumeralComponente, e SpinnerComponente.

5.2.2.1 Aplicativo CadastroEquipamento – Parte Client

```
public class CadEquipActivity extends Activity {

    public  MobimodControlHelper control;

        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            control = new MobimodControlHelper();//Cria-se a instancia de MobimodController

            MobiEditText nomeEt = new MobiEditText(1,"Nome",EditTextTiposEnum.NOME.getStatusCode(), 40, "Nome",false);
            MobiNumeral numEt = new MobiNumeral(3,"Num. Série", NumeralTiposEnum.INTEIRO.getStatusCode(), 1, 20000, "Num. Série",false);

            ElementoSpinner elemento1 = new ElementoSpinner(3,"Hardware");
            ElementoSpinner elemento2 = new ElementoSpinner(4,"Item Consumível");
            ElementoSpinner elemento3 = new ElementoSpinner(5,"Material Escritório");
            MobiSpinner spinnerCategoria = new MobiSpinner(2, "categorias");
            spinnerCategoria.addElemento(elemento1);
            spinnerCategoria.addElemento(elemento2);
            spinnerCategoria.addElemento(elemento3);
            MobiDatePicker datePicker = new MobiDatePicker(11,"Data Compra",12,true);

            control.addComponente(nomeEt); // adiciona o campo de nomeCompleto no map
            control.addComponente(spinnerCategoria); // adiciona o spinner de elementos no map de especificacao dos componentes
            control.addComponente(numEt); // adiciona o campo de numFilhos no map
            control.addComponente(datePicker); // adiciona datePicker

            Intent it = new Intent(this, MobimodView.class);
            it.putExtra(MobimodControlHelper.MAP_COMPONENTES, control.getMapComponentes());
            //Manda como parametro o map com a especificacao dos componentes

            it.putExtra(MobimodControlHelper.FUNDO_PRE_DEFINIDO, BackgroundEnum.BRANCO.getStatusCode());
            it.putExtra(MobimodControlHelper.ENVIAR_LABEL, "Enviar"); // OPCIONAL. Texto no botao de confirmar
            it.putExtra(MobimodControlHelper.CANCELAR_LABEL, "Cancelar"); // OPCIONAL. Texto no botao de cancelar
            startActivityForResult(it,MobimodControlHelper.START_FOR_RESULT);

        }
```

Figura 48: Criando e iniciando Formulário CadastroEquipamento

Na figura 48, o código que deve ser feito pelo desenvolvedor de aplicativo. O formulário é iniciado com a chamada de método startActivity(), mandando como parâmetro a *Intent* [27] criada e o atributo *MobiControlHelper.START_FOR_RESULT.* O resultado é o formulário gráfico criado, visto na figura 42 abaixo.

[27] Classe utilizada para iniciar uma *Activity*

Figura 49: Formulário gráfico criado

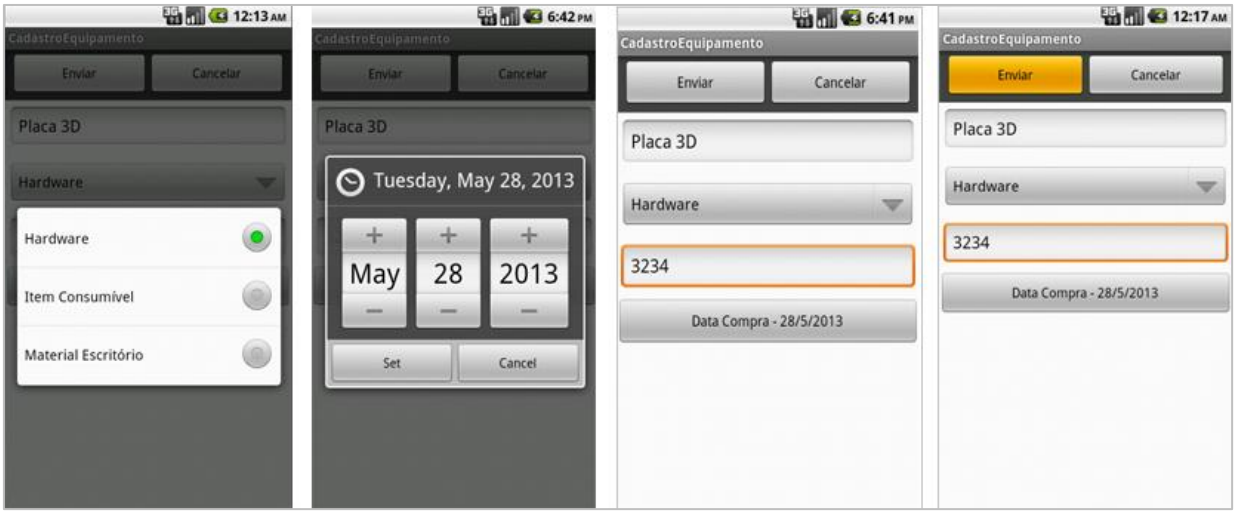

Figura 50: Formulário sendo preenchido

```
protected void onActivityResult(int codigo, int codigoRetorno, Intent it){
    super.onActivityResult(codigo, codigoRetorno, it);
    if(codigoRetorno == RESULT_OK){
        Bundle extras = it.getExtras();
        if(extras != null){
            if(extras.getSerializable("map_componentes")!=null){
                control.setMapComponentes((HashMap<Integer, MobiComponente>) extras.getSerializable("map_componentes"));
                mandaDadosImprimeRetorno();
            }
        }
    }
}
protected void mandaDadosImprimeRetorno(){
    String localIp = "10.0.2.2";
    String endereco = "http://"+localIp+":8080/CadastroEqptoWEB/CadastroEqptoServlet";
    String respostaServ = control.mandaDadosParaServidor(endereco);
    String msgRespostaTela = control.getStringFormatoApresentavel(respostaServ);


    Toast alerta = Toast.makeText(CadEquipActivity.this, "Resposta do Servidor \n\n"+msgRespostaTela, Toast.LENGTH_LONG);
    alerta.setGravity(Gravity.CENTER, 0, 0);
    alerta.show();
}
```

Figura 51: Código captura e envio de dados à servidor

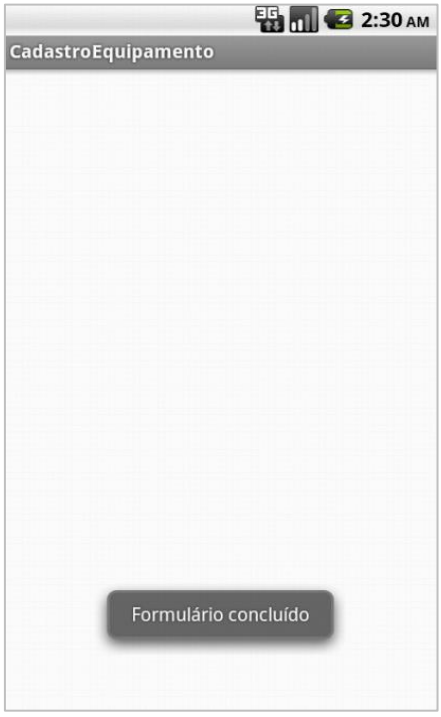

Figura 52: Formulário Concluído por Usuário

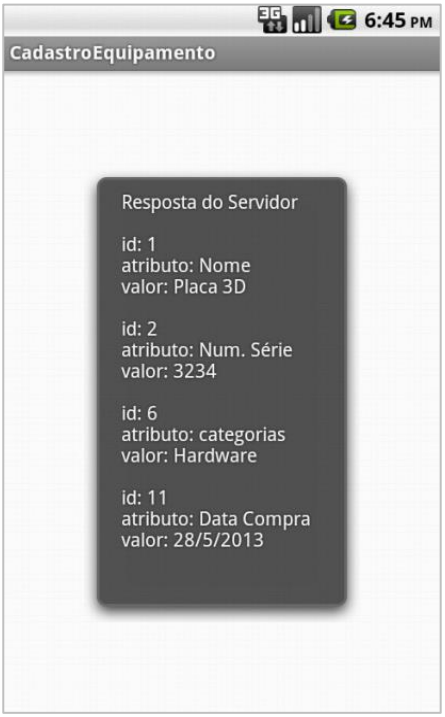

Figura 53: Mensagem de resposta do servidor

A figura 52 acima mostra a mensagem de formulário concluído, ou seja, passou da validação e o objeto JSON foi sintetizado. Na figura 53, pode-se ver que tal objeto foi enviado ao servidor e o mesmo respondeu, acusando o recebimento. Desta forma é comprovada a veracidade da informação.

5.2.2.2 Servlet CadastroEqptoServlet – Parte Servidor

```
  <servlet>
    <description></description>
    <display-name>CadastroEqptoServlet</display-name>
    <servlet-name>CadastroEqptoServlet</servlet-name>
    <servlet-class>com.example.servlet.CadastroEqptoServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>CadastroEqptoServlet</servlet-name>
    <url-pattern>/CadastroEqptoServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Figura 54: web.xml projeto CadastroEqptoWEB

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    String pDados = request.getParameter("dados_capturados");
    System.out.println(pDados);

    ServletOutputStream out = response.getOutputStream();
    out.write(pDados.getBytes());
    out.flush();
}
```

Figura 55: Método doPost() do Servlet CadastroEqptoServlet

Na figura 54, o arquivo web.xml especifica o servlet CadastroEqptoServlet que recebe os dados do formulário.Na figura 55, o trecho que mostra o método doPost() do servlet.

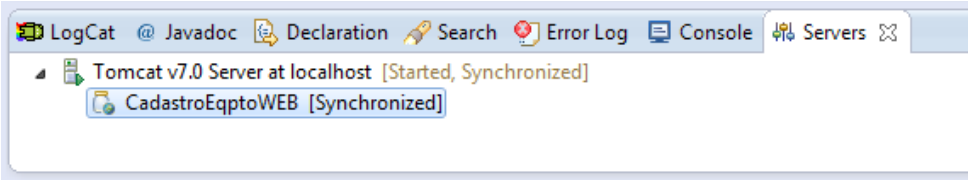

Figura 56: Servidor com projeto CadastroEqptoWEB iniciado

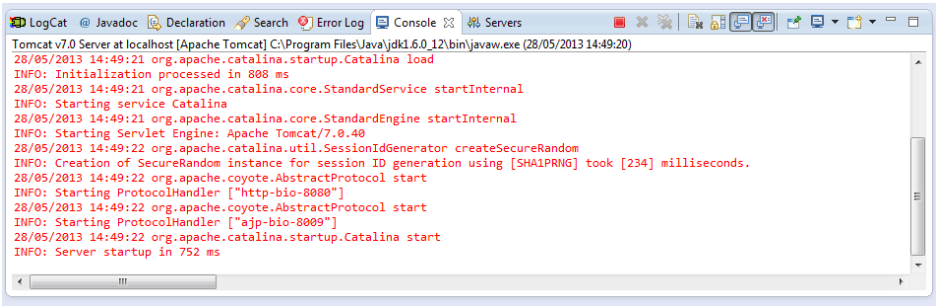

Figura 57: Console com servidor iniciado

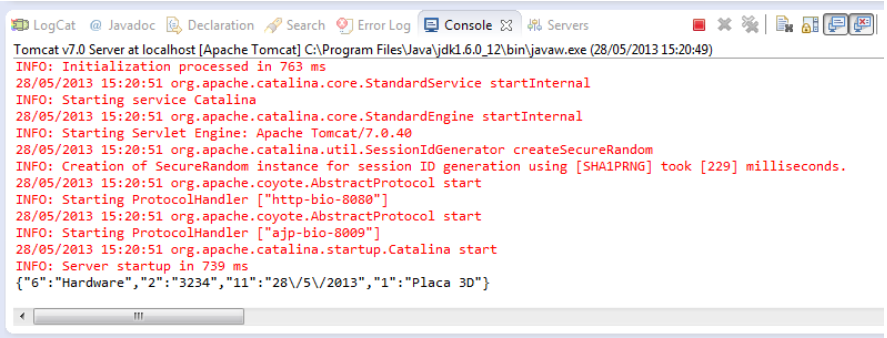

Figura 58: Console após receber dados no servlet

5.3 Comparação - Com e sem API

Para realizar uma comparação realística e mostrar as vantagens do uso da API desenvolvida, foi criada uma contra-prova. Dois projetos que montam o mesmo formulário, cadastro de equipamento. Um deles fazendo uso da MobimodAPI, já demonstrado no subcapitulo *5.2.2 Projeto Cadastro Equipamento* - e outro não – com o código anexado no apêndice. Ambos projetos-teste foram desenvolvidos pelo criador da API.

Sem fazer uso do API, é necessária a criação de um arquivo *XML* para cada classe - que estende *Activity* - criada. Cada vez que um novo componente é adicionado ao formulário, é preciso adicionar novas linhas de código neste arquivo *XML* e também na classe que estende *Activity.* Além disso, é necessário criar métodos que fazem a manipulação do componente.

Utilizando a MobimodAPI, a criação de *XML* foi totalmente descartada. É necessário criar apenas uma classe estendendo *Activity* e adicionar componentes diretamente na classe controle da API, que a mesma se encarrega da captura e validação dos dados, após o preenchimento feito pelo usuário. Por utilizar apenas bibliotecas nativas do Android, não há aumento de tempo de compilação perceptível ao usuário. Nesta comparação, foi ignorada a parte de integração com outros sistemas. Os resultados encontram-se na Tabela 1.

Cadastro Equipamento	Linhas XML para Tela	Activities criadas	Linhas Código Java	Validação Campos	Captura Dados em JSON
Com MobimodAPI	0	1	40	Sim	Sim
Sem MobimodAPI	60	2	140	Não	Não

Tabela 1: Resultados comparação

6 Conclusão e Trabalhos Futuros

O mercado mobile já se firmou como a mais nova revolução no ramo da tecnologia. Com grandes empresas apostando e investindo na área, a aceitação e necessidade encontrada na emergente clientela, cada vez mais, o mercado está altamente aquecido e com isso novas oportunidades vão surgindo. O Sistema Operacional *Android*, da Google lidera o mercado e atrai o desenvolvimento de aplicativos, APIs e frameworks para o próprio.

O *Android* fornece um kit de desenvolvimento muito completo, porém verboso, para os desenvolvedores implementarem suas aplicações. A API desenvolvida, para o SO citado, simplifica e agiliza a criação de módulo móvel que faça uso de formulários. Dessa forma, qualquer sistema pode "terceirizar" funcionalidades, aproveitando as vantagens da mobilidade.

Através de testes e comparações, foi possível constatar que o uso da API diminui em cerca de 290% a quantidade de código desenvolvido, eliminou necessidade de criação de arquivos .xml para telas e ainda deu suporte para coleta e validação de dados inseridos por usuários. Existe uma perda em flexibilidade de criação e design de telas, uma vez que utilizando a API, os componentes ficam sempre organizados de maneira linear, ordenadas verticalmente. Por isso, o uso da API é mais recomendado em situações onde o design não necessita ser altamente atrativo, como em situações corporativas, onde coleta de dados simplista e economia de tempo em desenvolvimento são mais importantes do que grande investimento em experiência de usuário.

Com esse ganho de tempo e a facilidade em que o desenvolvedor se encontra para criar formulários em *android* utilizando esta API, é possível focar-se no domínio dos problemas sem ter grande preocupação e gasto com pesquisas e desenvolvimento, principalmente em criação de telas, captura e validação de dados inseridos por usuário.

6.1 Trabalhos Futuros

Com o visível ganho que a API desenvolvida oferece, é possível pensar em novas funcionalidades que podem melhorar ainda mais esta solução. Algumas delas:

- Prover mais possibilidades de layout para formulário;
- Novos componentes de tela;
- Novas funções de personalização de telas;
- Suporte para outros tipos de telas – além de formulários;
- Integração com GPS e Google Maps.

Referências

LECHETA, R. Google Android - **Aprenda a criar aplicações para dispositivos móveis com o Android SDK 2ª Edição**. São Paulo: Novatec Editora Ltda, 2011.

DEVELOPERS, A. Android Developers, Novembro 2012. Disponível em: <http://developer.android.com/>. Acesso em: 12 Novembro 2012.

IDC, **International Data Corporation,** Dezembro 2012. Disponível em: < http://www.idc.com />. Acesso em: 12 Dezembro 2012.

SILVA, R. P. E. **Suporte ao Desenvolvimento e uso de Framework e Componentes**. Porto Alegre: Instituto de Informática da UFGS, 2000.

POCATILU, Paul. ***Developing Mobile Learning Applications for Android using Web Services***, 20 set. 2010. Disponível em: <http://www.mobilepedia.com.br/noticias/450-mil-tablets-serao-vendidos-no-brasil-ate-o-final-do-ano>. Acesso em: 20 Setembro 2012.

RAPCINSKI, Heitor. ***Utilizando subversion como controle de versão***, 29 março 2012 < *http://www.guj.com.br/content/articles/svn/SubVersion_GUJ.pdf>.* Acesso em: Setembro 2012.

SPRING, **Rest Framework**. Disponível em: < http://static.springsource.org/spring/docs/3.0.0.M3/reference/html/ch18.html>. Acesso em 22 Agosto 2012

APPLE INSIDER, **Mobile Market Share 2Q 2009**,Janeiro 2010. Disponível em: < http://appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html>. Acesso em 16 Agosto 2012.

WIKIPEDIA, **Mobile Market Share 2Q 2012,** Janeiro 2013.
Disponível em: < http://pt.wikipedia.org/wiki/Ficheiro:Android_chart.png>. Acesso em 16 Fevereiro 2013.

MY SMART PRICE, **Samsung Galaxy S2 I9100,** Janeiro 2013.
Disponível em: < http://www.mysmartprice.com/mobile/samsung-galaxy-s2-i9100-msp1170>. Acesso em 16 Fevereiro 2013.

GONÇALVES, E. **Desenvolvendo Aplicações Web com JSP, Servlets, JavaServer Faces, Hibernate, EJB 3 Persistence e AJAX**. Rio de Janeiro: Ciência Moderna LTDA, 2007.

WIKIPEDIA, **HTTP,** Janeiro 2013.
Disponível em: <http://pt.wikipedia.org/wiki/Http>. Acesso em 16 Agosto 2012.

WIKIPEDIA, **Web Service,** Janeiro 2013.
Disponível em: <http://pt.wikipedia.org/wiki/Web_service>. Acesso em 16 Agosto 2012.

ECLIPSE, Eclipse Foundation, **What is Eclipse and what is Eclipse Foundation?** Disponível em: <http://www.eclipse.org/org/#about>. Acesso em 15 Agosto 2012.

BLOCH, J. **How to design a good API and why it matters**. Disponível em: <http://www.infoq.com/presentations/effective-api-design>. Acesso em 22 Novembro 2012.

OMG, Object Management Group**, Unified Model Language** Julho 2013. Disponível em: <http://www.omg.org/spec/UML/>. Acesso em 2 Julho 2013.

AGILE MODELING, Agile Modeling **, Introduction to the diagrams of UML 2.0** Julho 2013. Disponível em: < http://www.agilemodeling.com/essays/umlDiagrams.htm >. Acesso em 2 Julho 2013.

APÊNDICE 1 - Classe MainActivity

```
package com.example.contraprovacadeqpto;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;


public class MainActivity extends Activity{
        public void onCreate(Bundle savedInstanceState){
                super.onCreate(savedInstanceState);
                Intent it = new Intent(this, TelaActivity.class);
           startActivity(it);
        }
        public void onDestroy(){
          super.onDestroy();
        }
}
```

APÊNDICE 2 - Classe TelaActivity

```
package com.example.contraprovacadeqpto;
import java.util.Calendar;
import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.Spinner;


public class TelaActivity extends Activity{
        protected LinearLayout painelEnviarCancelar;
        protected EditText nomeEt, numEt;
        protected Button okButton, cancelarButton,dataNasc;
        protected Spinner spinner;
        private static final int DATE_DIALOG_ID = 0;
         protected int mYear,mMonth,mDay;
        public void onCreate(Bundle savedInstanceState){
                super.onCreate(savedInstanceState);
                setContentView(R.layout.tela_cadastro);
                painelEnviarCancelar = (LinearLayout) findViewById(R.id.enviarCancelarPainel);
                okButton = (Button) findViewById(R.id.okButton);
                dataNasc = (Button) findViewById(R.id.dataNascButton);
                cancelarButton = (Button) findViewById(R.id.cancelarButton);
                nomeEt = (EditText) findViewById(R.id.nomeField);
                numEt = (EditText) findViewById(R.id.numSerieField);
                iniciaSpinner();

                 final Calendar c = Calendar.getInstance();
            mYear = c.get(Calendar.YEAR);
            mMonth = c.get(Calendar.MONTH);
            mDay = c.get(Calendar.DAY_OF_MONTH);



                okButton.setOnClickListener(mOkListener);
                cancelarButton.setOnClickListener(mCancelarListener);
                dataNasc.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
           showDialog(DATE_DIALOG_ID);
        }
```

```
        });
            }
            private void updateDisplay() {
                        dataNasc.setText(
                        new StringBuilder()
                                    .append(mDay).append("-")
                              .append(mMonth + 1).append("-")
                              .append(mYear).append(" "));
    }

    protected Dialog onCreateDialog(int id) {
                switch (id) {
                case DATE_DIALOG_ID:
                   return new DatePickerDialog(this,
                              mDateSetListener,
                              mYear, mMonth, mDay);
                }
                return null;
               }

    private DatePickerDialog.OnDateSetListener mDateSetListener =
          new DatePickerDialog.OnDateSetListener() {
             public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
                arrumaData(year, monthOfYear, dayOfMonth);
                updateDisplay();
             }
      };
      private void arrumaData(int year, int monthOfYear, int dayOfMonth) {
                        mYear = year;
                  mMonth = monthOfYear;
                  mDay = dayOfMonth;

                  monthOfYear+=1;
               }
            public void iniciaSpinner(){
                        spinner = (Spinner) findViewById(R.id.categoriaSpinner);
               ArrayAdapter<CharSequence> adapterProtocolos = ArrayAdapter.createFromResource(
                     this, R.array.categoria_array, android.R.layout.simple_spinner_item);
               adapterProtocolos.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
               spinner.setAdapter(adapterProtocolos);

            }
            private OnClickListener mOkListener = new OnClickListener() {
                        @Override
                        public void onClick(View arg0) {
                        }

            };
            private OnClickListener mCancelarListener = new OnClickListener() {
                        @Override
                        public void onClick(View arg0) {
                                    finish();
                        }

            };
             public void onDestroy(){
               super.onDestroy();
             }
}
```

APÊNDICE 3 – XML da Interface de Usuário tela_cadastro.xml

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

   <LinearLayout
      android:id="@+id/enviarCancelarPainel"
      android:layout_width="fill_parent"
      android:layout_height="100dp"
      android:padding="5dp"
      android:orientation="horizontal" >
      <Button
        android:id="@+id/okButton"
        android:layout_height="50dp"
        android:layout_width="100dp"
        android:text="Ok">
      </Button>
       <Button
        android:id="@+id/cancelarButton"
        android:layout_height="50dp"
        android:layout_width="100dp"
        android:layout_gravity="right"
        android:text="Cancelar">
      </Button>

   </LinearLayout>
      <EditText
                android:id="@+id/nomeField"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:hint="Nome Completo"
                android:maxLength="40"
                android:focusableInTouchMode="true"
                android:inputType="textCapWords"
                android:background="@android:drawable/editbox_background"/>

      <EditText
                android:id="@+id/numSerieField"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:hint="Num Serie"
                android:maxLength="5"
                android:inputType="numberDecimal"
                android:background="@android:drawable/editbox_background"/>

      <Spinner
         android:id="@+id/categoriaSpinner"
         android:layout_width="fill_parent"
                 android:layout_height="50dp"
                 android:prompt="@string/categoria_prompt"/>
       <Button
        android:id="@+id/dataNascButton"
        android:layout_height="50dp"
        android:layout_width="fill_parent"
        android:layout_gravity="center"
        android:text="Data Compra">
      </Button>

</LinearLayout>
```