COMPUTAÇÃO PARA DISPOSITIVOS MÓVEIS

CAPÍTULO 2 - QUAIS RECURSOS E APIS DO SISTEMA ANDROID PODEMOS UTILIZAR PARA TORNAR NOSSA INTERFACE DE USUÁRIO DINÂMICA?

Fernando Skackauskas Dias

INICIAR

Introdução

A tecnologia dos dispositivos móveis tem avançado consideravelmente nos últimos anos. Novos aparelhos com recursos inéditos, interfaces cada vez mais amigáveis e disponibilidade de novos serviços, dentre outras inovações, estão dominando o mercado. Diante deste panorama, os cientistas da computação se encontram diante de grandes desafios.

Nesse contexto, o sistema operacional Android foi desenvolvido com o objetivo de se tornar um sistema aberto, englobando diversas novas funcionalidades. Por ser um sistema aberto, significa que diferentes desenvolvedores podem atuar diretamente no sistema operacional. Portanto, podemos questionar: quais são os

recursos e aplicativos do sistema Android que podem ser utilizados? Como melhorar a interface do usuário, tornando-a mais dinâmica? Como utilizar os seus recursos de forma a desenvolver aplicativos com melhor qualidade?

O sistema operacional Android foi desenvolvido por um consórcio de empresas, sendo que, além de ser um sistema aberto, ele detém outras vantagens como, por exemplo, o fato de várias marcas de dispositivos móveis o utilizarem como SO e a possibilidade de ter acesso a vários *softwares* por meio do serviço de distribuição digital de aplicativos.

Este capítulo é focado nos recursos e padrões de programação do sistema operacional Android. Vamos entender como é feita a persistência de arquivos, como funciona o banco de dados SQLite e os provedores de conteúdo, como é o processamento XML e como são identificadas as formas de acesso à rede. Por fim, será apresentado o padrão de desenvolvimento JSON e a ferramenta Retrofit.

Acompanhe com atenção e bons estudos!

2.1 Sistemas de arquivo e acesso à rede

A grande maioria dos sistemas precisa que seus dados fiquem armazenados em algum dispositivo para posterior acesso. Esta quantidade de dados a serem armazenados dependerá da natureza e do propósito do sistema. Mas, como esse armazenamento de dados é controlado no sistema operacional de um dispositivo móvel? Como o sistema operacional Android gerencia os dados que necessitam ser armazenados?

Vamos entender, agora, sobre a persistência dos dados no sistema operacional Android e quais são os seus principais comandos.

Mas, primeiro, vamos descrever o que é o Android. Conforme explicado por Pilar (2013, p. 43) “o Android é uma plataforma móvel de código livre, baseada em Linux desenvolvida pela Google, com o objetivo de operar em dispositivos com toque na tela, como *smartphones* e *tablets*”. As principais características do Android são (PILAR, 2013, p. 44):

- utilização de *widgets*: pequenos ícones que ficam na tela inicial, onde é

possível escolher entre seus aplicativos favoritos, tendo acesso a atualizações sem sair da tela principal;

- tela de notificações: onde é possível visualizar em um só lugar as notificações de novas ligações, mensagens e e-mails;
- multitarefa: é possível executar diversos aplicativos ao mesmo tempo e alternar a visualização entre eles;
- escrita por voz: possibilidade de escrever e-mails, SMS e qualquer tipo de texto, através da voz;
- compartilhamento de imagens: possibilidade de explorar, editar e compartilhar rapidamente nas redes sociais as fotos tiradas direto do celular;
- compartilhar com o toque: compartilhamento de fotos e vídeos com outros celulares com a mesma tecnologia, apenas encostando-os. Também é possível parear dispositivos via *bluetooth* apenas encostando os aparelhos;
- Fotos panorâmicas de 360º: através do Photo Sphere é possível tirar fotos panorâmicas de todo o seu redor e realizar *upload* do resultado direto para o Google Maps, onde outros usuários poderão visualizá-la;
- múltiplos usuários: nos *tablets*, é possível criar múltiplos usuários, cada um com sua tela inicial, aplicativos e jogos.

Para que o dispositivo móvel possa executar todas as funcionalidades, é fundamental que as informações sejam armazenadas e recuperadas com eficiência. Vamos ver, a seguir, a demonstração da persistência de arquivos no sistema operacional Android.

Persistência de arquivos e preferências

Para poder disponibilizar todas as funcionalidades, é necessário que os dados de todas as aplicações sejam armazenados e recuperados, portanto, precisam ser salvos. Os dispositivos podem guardar uma quantidade limitada de informações no seu próprio armazenamento, porém, está restrito ao tamanho do espaço do endereçamento virtual. Tanenbaum (2016, p. 181) explica que “para algumas aplicações esse tamanho é adequado, mas, para outras, como reservas de passagens aéreas, bancos ou sistemas corporativos, ele é pequeno demais”.

Outro problema relativo ao armazenamento é a perda das informações quando o processo é concluído. Além destes problemas, o que ocorre é que vários processos concorrentes precisam acessar a mesma informação ao mesmo tempo. A solução encontrada é tornar a informação independente de qualquer processo.

Tanenbaum (2016, p. 181) descreve os três requisitos essenciais para o armazenamento de informação por um longo prazo:

- deve ser possível armazenar uma quantidade muito grande de informações;
- as informações devem sobreviver ao término do processo que as está utilizando;
- múltiplos processos têm que ser capazes de acessá-las ao mesmo tempo.

Portanto, as informações armazenadas em arquivos, devem ser persistentes, quer dizer, não podem ser afetadas por uma criação ou término de qualquer processo. Mas, como, no caso do Android, os dados podem ser salvos e serem persistentes?

O Android oferece várias opções de armazenamento dos dados persistentes de aplicativos. Inicialmente, a escolha dependerá das necessidades do aplicativo, se devem ser privados ou podem ser acessados por outros aplicativos. O Android oferece quatro maneiras de armazenamento. Vamos ver quais são a seguir.

VOCÊ QUER LER?

Quer conhecer uma pesquisa que auxilia o diagnóstico e a pesquisa médica, baseada em mineração de dados, utilizando interface Android? Este trabalho foi desenvolvido por Fernandes (2013). Para conhecer, acesse: < http://hdl.handle.net/10923/3199 (http://hdl.handle.net/10923/3199)>.

Quando se desenvolvem aplicativos para dispositivos móveis, muitas vezes é necessária a criação de um banco de dados, ou o armazenamento, quando há um número reduzido de dados. Com este objetivo, o sistema operacional Android oferece a classe SharePreferences. Inicialmente, a classe SharedPreference possibilita salvar e recuperar pares de valor de dados primitivos. Ressaltando que dados primitivos são dados *booleanos*, ponto flutuante, inteiros, inteiros longos e *strings.*

A maneira de se recuperar os dados são duas. A primeira é o método getSharedPreferences (*string* nome, int modo), que permite obter várias

preferências identificadas pelo nome que foi passado como parâmetro. O outro método é o getPreferences (int modo), para se recuperar dados dentro de uma mesma chave.

Outra forma possível de armazenamento no Android é o armazenamento interno de dados, que podem ser salvos diretamente na memória do dispositivo. De forma geral, outros aplicativos não têm permissão de acessar estes dados, a não ser o próprio aplicativo que os criou. O que ocorre é que quando o aplicativo que armazenou estes dados é desinstalado, os dados também são removidos. Os métodos de programação que permitem manipular estas informações são:

- **openFileOutput()**, com o nome do arquivo e o modo de funcionamento para recuperar os dados armazenados. Para escrever estes dados, se usa o método write() e o método close() para encerrar o fluxo;
- **armazenamento externo**. Quando o dispositivo é compatível com o Android, ele comporta uma memória externa para salvar arquivos, podendo ser uma mídia removível. É necessário executar na classe Environment, o método getExternalStorageState() para verificar se a mídia externa está disponível e o método getExternalFilesDir(), para acessar o arquivo.

Banco de dados SQLite e provedores de conteúdo

O Android desenvolveu um banco de dados com código aberto de estrutura relacional chamado SQLite. O SQLite é embutido em todos os dispositivos com o sistema operacional Android, sendo que o desenvolvedor de aplicativos, para esta plataforma, precisa somente definir as estruturas das tabelas e as operações que serão realizadas. Uma das suas principais características é que o SQLite não necessita que nenhum servidor SQL seja previamente instalado. Pois “SQLite é um dos mecanismos de banco de dados mais amplamente distribuídos do mundo. Cada fragmento deste aplicativo interage com um banco de dados SQLite por intermédio da classe utilitária DatabaseConnector” (DEITEL; DEITEL; WALD, 2016, p. 244). Ou seja, não é necessário fazer sua instalação e nem a configuração, visto que todo o banco de dados fica em arquivo. Na verdade, o SQLite é uma biblioteca em linguagem “C”, na qual ela mesma gerencia o banco.

Outra capacidade de armazenamento de dados do Android são os provedores de conteúdo. Estes são objetos que tem a capacidade de armazenar os dados de forma permanente, tornando-os disponíveis para que outras aplicações possam acessar. O pacote Android.provider já fornece este provedor de conteúdo. O Android fornece duas maneiras para que os seus dados se tornem públicos. Uma é a classe ContentProvider. Mas, como é possível localizar um provedor de conteúdo?

Cada tipo de dado disponibilizado pode ser encontrado por uma URI (*Uniform Resource Identifier*) diferente. Outra característica é que um mesmo provedor pode disponibilizar vários tipos de dados, sendo que cada tipo tem um URI diferente. Tem-se como exemplo:

Android.provider.contacts.Phones.CONTENT_URI

Observe que a classe é separada em quatro: a primeira parte indica que o dado é fornecido por um determinado provedor de conteúdo; logo após, vem o identificador do provedor de conteúdo; em seguida o tipo de dados; e, por fim, o identificador de um determinado registro.

Processamento XmlPullParser

O XML (do inglês *eXtensible Markup Language*) é uma linguagem de marcação, diferente das outras linguagens de programação, que corresponde a um agregado de códigos para padronizar uma determinada sequência de dados, separando o conteúdo do código. Ela foi desenvolvida originalmente para criar documentos, nos quais os dados são organizados hierarquicamente. Vamos ver, a seguir, um exemplo de um documento XML.

```
<?xml version="1.0"?>
<aviso>
<para>Fernando data="18/06/2018"</para>
<de>Juliana</de>
<cabecalho>Lembrete</cabecalho>
<corpo>É necessário fazer as correções no documento</corpo>
</aviso>
```

#PraCegoVer: Código: <aviso> <para>Fernando data= “18/06/2018”</para> <de>Juliana</de> <cabecalho>Lembrete</cabecalho> <corpo>É necessário fazer as correções no documento</corpo> </aviso>

A primeira linha do documento, **<?xml version="1.0"?>**, é uma declaração XML e é necessário que seja incluída, pois define a versão XML do documento. No exemplo, está sendo especificada a versão 1.0 da XML.

A linha **<aviso>,** define o primeiro elemento do documento, o elemento raiz.

```
<para>Fernando data="18/06/2018"</para>
<de>Juliana</de>
<cabecalho>Lembrete</cabecalho>
<corpo>É necessário fazer as correções no documento</corpo>
```

Estas quatro linhas definem os quatro elementos filhos da raiz (para, de, cabeçalho e corpo).

Por fim, última linha, **</aviso>,** define o fim do elemento raiz.

VOCÊ QUER LER?

O artigo “Sistema de recomendação baseado na localização e perfil utilizando a plataforma Android” (MACK, 2010) mostra como projetar um aplicativo no sistema operacional Android, para que os usuários recebam recomendações de estabelecimentos comerciais que fazem parte de uma determinada categoria, utilizando a posição geográfica do usuário. Para ler o trabalho completo, acesse: < http://hdl.handle.net/10183/28328 (http://hdl.handle.net/10183/28328)>.

No caso do sistema operacional Android, o XML contido nele, pode ser de um dos seguintes tipos: DOM, SAX e PullParser. O PullParser é uma interface que permite criar implementações do XML. Existem diferentes tipos de *parser,* dependendo dos

recursos. Um deles é o analisador de não-validação, conforme a especificação XML 1.0, quando FEATURE_PROCESS_DOCDECL está definido como verdadeiro. Por outro lado, quando FEATURE_PROCESS_DOCDECL é falso, o analisador se comporta como sendo uma validação não compatível com XML 1.0 sob a condição de que nenhum DOCDECL esteja presente em documentos XML.

Este modo de operação destina-se a ambientes restritos, como o J2ME ou JME. JME significa “Java Platform, Micro Edition”, oferecendo mais robustez e flexibilidade para aplicativos que são executados em dispositivos móveis e integrados, como celulares, dispositivos de mídia digital e impressoras. Utilizando-se o método next() é possível recuperar o valor de um determinado evento e compará-lo às constantes dos eventos da classe que está sendo executada. Cada elemento que foi encontrado tem um nome, valor de texto e um conjunto opcional de atributos.

Acesso à rede

O sistema operacional Android engloba uma série de funcionalidades, como o acesso à rede. Neste caso, existe a solução Android Samba Client (SMB) que permite acessar arquivos externos ao dispositivo móvel. Portanto, qualquer dispositivo móvel que utilizar o Android pode acessar, por exemplo, partições do Windows.

Formato JSON

JSON – do inglês JavaScript Object Notation – é uma estrutura que tem por objetivo representar os dados. Ela foi desenvolvida para ter um formato mais leve e simples de troca de dados. O Android fornece quatro classes diferentes para manipular dados JSON. Essas classes são: JSONArray, JSONObject, JSONStringer e JSONTokenizer. Existe uma vantagem ao se utilizar a classe JSON, que é a possibilidade de acessar as aplicações *web* no sistema operacional Android, sendo que no XML é necessário encontrar uma solução para integração, como baixar um leitor de XML. Comparando-se o XML e JSON, um arquivo XML é mais “burocrático” de ser escrito do que o JSON. Por exemplo, se formos representar uma pessoa com os atributos “nome” e “idade”, em XML, ficaria no seguinte formato:

<pessoa>

<nome>Nawarian Níckolas</nome>

<idade>20</idade>

</pessoa>

Se for representado em JSON, ficaria da seguinte maneira:

{

"nome": "Nawarian Níckolas",

"idade": 20

}

Portanto, o XML possui uma capacidade descritiva muito alta e, consequentemente, pode alcançar níveis de complexidade muito altos.

Retrofit para acesso a serviços REST

Os aplicativos de dispositivos móveis precisam se conectar com outros aparelhos para receber e enviar dados. O HTTP – sigla de *HyperText Transfer Protocol (*que significa Protocolo de Transferência de Hipertexto) – é um protocolo que efetua a transferência de dados entre computadores em rede. E como é realizada esta transferência no sistema operacional Android?

O sistema operacional Android oferece a biblioteca retrofit, que permite ao desenvolvedor realizar qualquer tipo de comunicação, via HTTP, de forma bastante amigável. Tendo sido desenvolvida em Java, com o objetivo de criar uma solicitação HTTP, o modelo REST (REpresentational State Transfer) representa a possibilidade para a criação de *web services*, cujas principais diferenças em relação ao modelo tradicional estão na utilização semântica dos métodos HTTP (GET, POST, PUT e DELETE) (SUBONIS, 2017).

Portanto, esta biblioteca do Android processa a resposta HTTP de uma API REST. O REST Client é a biblioteca Retrofit que é usada no lado do cliente (Android) para fazer uma solicitação HTTP para a API REST, o que seria o nosso caso.

Uma API REST define um conjunto de funções, nas quais os desenvolvedores podem executar solicitações e receber respostas, via protocolo HTTP, como GET e POST. Também pode-se dizer que uma API RESTful é uma interface de programa de aplicativo (API, em inglês: *Application Programming Interface*) que usa solicitações HTTP para dados GET, PUT, POST e DELETE.

Desenvolvimento de aplicativo utilizando SQLite

Vamos demonstrar, a seguir, a criação de um aplicativo CRUD (criação, consulta, alteração e exclusão) para o controle de livros de uma biblioteca, utilizando o banco de dados interno ao Android, o SQLite.

Os programadores Deitel, Deitel e Wald (2016) definem o SQLite como um banco de dados no modelo relacional, com código de acesso livre. O sistema operacional Android permite criar quantos bancos de dados forem necessários, sendo acessados e visualizados somente pelo próprio aplicativo.

É necessário utilizar uma API de acesso ao banco de dados SQLite, que já é um componente do pacote SDK. Assim, duas classes serão utilizadas para a criação do banco de dados, via aplicação (DEITEL; DEITEL; WALD, 2016):

- SQLiteDatabase: nesta classe, estão os métodos de manipulação dos dados no banco;
- SQLiteOpenHelper: esta é a classe responsável pela criação do banco e por seu versionamento.

Vamos ver como fica na figura a seguir.

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by Fernando Skackauskas Dias - 2018.
 */

public class CriaBanco extends SQLiteOpenHelper {
    @Override
    public void onCreate(SQLiteDatabase db) {

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    }
}
```

Figura 1 - Criação de uma classe para a criação do banco de dados de um aplicativo desenvolvido em Android. Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: Print da tela com o código:

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
/**
* Created by Fernando Skackauskas Dias – 2018.
*/
public class CriaBanco extends SQLiteOpenHelper {
@Override
Public void onCreate(SQLiteDatabase db){
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

}
}

O código apresentado acima foi desenvolvido na ferramenta Android Studio, na

qual foi feita a prototipação dos métodos. É necessário implementar dois métodos para que o aplicativo funcione corretamente:

- **onCreate()**: este é o método implementado no momento em que a aplicação cria o banco de dados pela primeira vez. Ele deve conter todas as diretrizes de criação e inserção de dados inicial do banco;
- **onUpgrade()**: este é o método que atualiza o banco de dados com as alterações sofridas pelas informações estruturais.

O objeto principal e db da classe SQLite, o método onUpgrade(), tem como objetivo atualizar as informações. Ele recebe dois parâmetros: a versão antiga da tabela e uma nova versão, para onde deverá ser feito o *upgrade*. Assim será criada uma tabela de cadastro com os campos: id, título, autor e editora. Na figura a seguir, é mostrado o código com o método onUpgrade(), com o comando SQL que deleta a tabela, caso ela exista, para depois, invocar o método onCreate() e recriar a tabela com as alterações feitas.

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by Fernando Skackauskas Dias - 2018.
 */
public class CriaBanco extends SQLiteOpenHelper {
    private static final String NOME_BANCO = "banco.db";
    private static final String TABELA = "livros";
    private static final String ID = "_id";
    private static final String TITULO = "titulo";
    private static final String AUTOR = "autor";
    private static final String EDITORA = "editora";
    private static final int VERSAO = 1;

    @Override
    public void onCreate(SQLiteDatabase db) {
       String sql = "CREATE TABLE"+TABELA+"("
+ ID + " integer primary key autoincrement,"
+ TITULO + " text,"
+ AUTOR + " text,"
+ EDITORA + " text"
+")";
      db.execSQL(sql);
    }

      public CriaBanco(Context context){
          super(context, NOME_BANCO,null,VERSAO);
      }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
          db.execSQL("DROP TABLE IF EXISTS" + TABELA);
          onCreate(db);
    }
}
```

Figura 2 - Criação de uma classe para desenvolver a estrutura com os campos do banco de dados de um aplicativo em Android. Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: Print da tela com o código:

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
/**
* Created by Fernando Skackauskas Dias – 2018.
*/
public class CriaBanco extends SQLiteOpenHelper {
private static final String NOME_BANCO = “banco.db”;

```
private static final String TABELA = “livros”;
private static final String ID = “id”;
private static final String TITULO = “titulo”;
private static final String AUTOR = “autor”;
private static final String EDITORA = “editora”;
private static final int VERSAO = 1;

@Override
public void onCreate(SQLiteDatabase db){
String sql = “CREATE TABLE”+TABELA+“{“
+ ID + “integer primary key autoincrement,”
+ TITULO + “text,”
+ AUTOR + “text,”
+ EDITORA + “text,”
+ “}”;
db.execSQL(sql);
}
Public CriaBanco(Context context){
super(context, NOEM_BANCO,null,VERSAO);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
db.execSQL(“DROP TABLE IF EXISTS” + TABELA);
onCreate(db);
}
}
```

Para criar uma *Activity* é necessário estender a classe *Activity*, ou uma de suas implementações como AppCompatActivity ou ListActivity. Isso tem como objetivo fazer com que a classe herde todas as características das *Activities*, senão ela seria somente uma classe do Java. Temos um exemplo a seguir.

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

#PraCegoVer: Print da tela com o código:

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

Na figura a seguir, podemos ver a tela do Android Studio, na qual é possível criar a interface de entrada de dados.

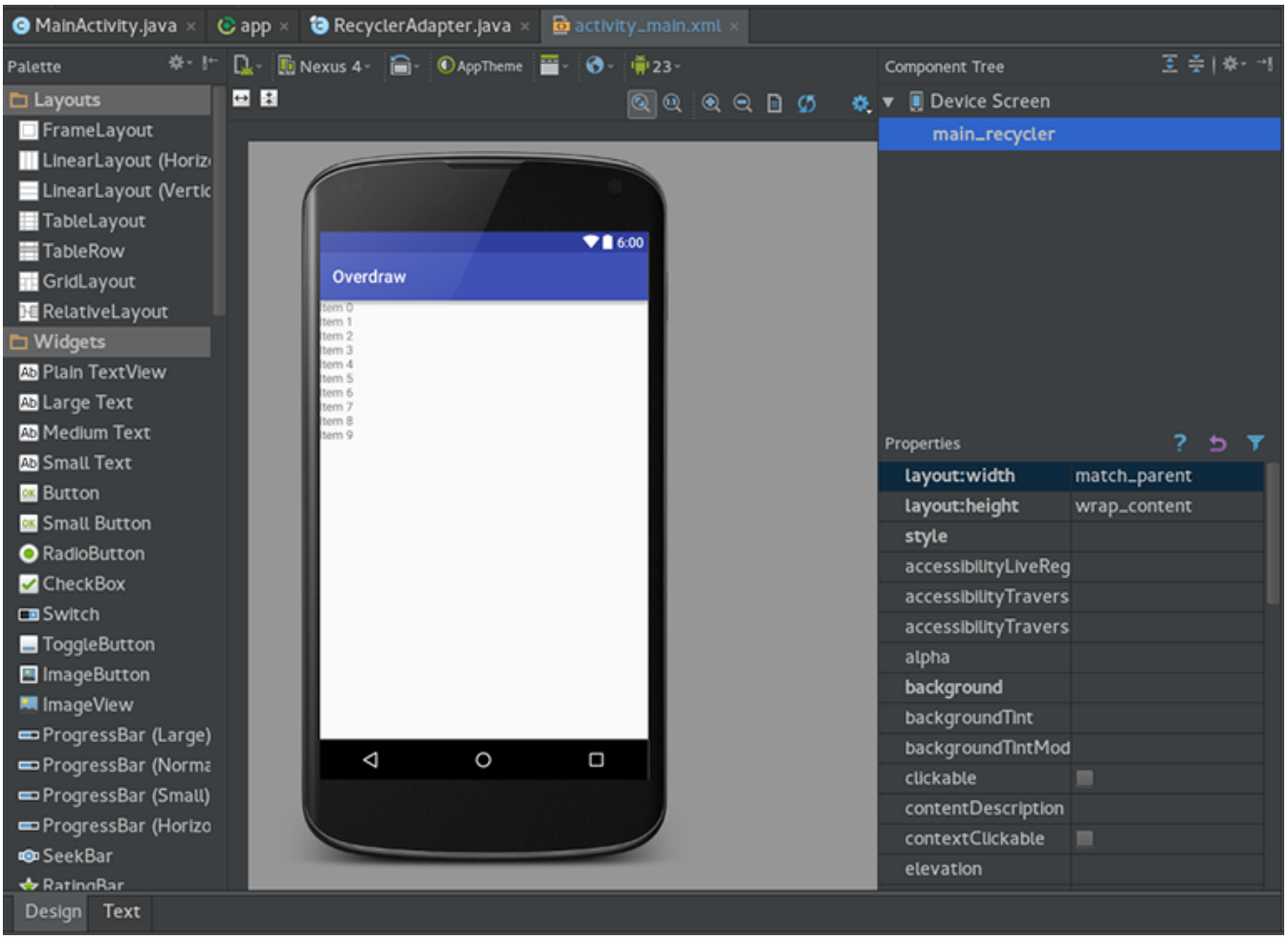


Figura 3 - Tela do Android Studio, onde é possível elaborar a interface do usuário descrevendo os campos a serem exibidos no aplicativo. Fonte: Elaborada pelo autor, 2018, adaptada da tela do Preview do Android Studio.

#PraCegoVer: Desenho de um celular, além dos campos: Layout, Widgets. Na tela do celular tem uma faixa azul no topo da tela escrito Overdraw.

Agora, o arquivo XML do *layout* no Android Studio, no qual, o usuário insere as informações.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".Inicial"
    android:id="@+id/Inser">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Titulo:"
        android:id="@+id/textView"
        android:layout_marginTop="47dp"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:layout_below="@+id/textView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Autor:"
        android:id="@+id/textView2"
        android:layout_below="@+id/editText"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="51dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText2"
        android:layout_below="@+id/textView2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignRight="@+id/editText"
        android:layout_alignEnd="@+id/editText" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:text="Editora:"
        android:id="@+id/textView3"
        android:layout_below="@+id/editText2"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="64dp" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText3"
```

Figura 4 - Código XML do layout do Insert de um aplicativo desenvolvido em Android, utilizando o banco de dados SQLite, adaptada da tela do Preview do Android Studio, 2018

```
        android:layout_below="@+id/textView3"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignRight="@+id/editText2"
        android:layout_alignEnd="@+id/editText2" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Cadastrar"
        android:id="@+id/button"
        android:layout_below="@+id/editText3"
        android:layout_alignParentStart="true"
        android:layout_marginTop="47dp"
        android:layout_alignRight="@+id/editText3"
        android:layout_alignEnd="@+id/editText3" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:id="@+id/textView4"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

#PraCegover: Print da tela com o código:

<Relativelayout xmlns:android=“http://schemas.android.com/apk/res/android”
xmlns:tools=“http://schemas.android.com/tools”
android:layout_width=“match_parent”
android:layout_height=”match_parent”
android:paddingLeft=“@dimen/activity_horizontal_margin”
android:paddingRight=“@dimen/activity_horizontal_margin”
android:paddingTop=“@dimen/activity_vertical_margin”
android:paddingBottom=“@dimen/activity_vertical_margin”
tools:context=“Inicial”
android:id=“@+id/Inser”/

<TextView
android:layout_width=“wrap_content”
android:layout_height=“wrap_content”
android:textAppearence=“?android:attr/textAppearanceMedium”
android:text=“Titulo:”
android:id=“@+id/TextView”
android:layout_marginTop=“47dp”
android:layout_alignParentTop=“true”
android:layout_alignParentLeft=“true”
android:layout_alignParentStart=“true” />

<EditView
android:layout_width=“wrap_content”
android:layout_height=“wrap_content”
android:id=“@+id/editText”
android:layout_below=“@+id/textView”
android:layout_alignParentLeft=“true”
android:layout_alignParentRight=“true”
android:layout_alignParentEnd=“true” />

```
<TextView
android:layout_width=“wrap_content”
android:layout_height=“wrap_content”
android:textAppearence=“?android:attr/textAppearanceMedium”
android:text=“Autor:”
android:id=“@+id/TextView2”
android:layout_below=“@+id/editText”
android:layout_alignParentLeft=“true”
android:layout_alignParentStart=“true”
android:layout_marginTop=“51dp” />

<EditView
android:layout_width=“wrap_content”
android:layout_height=“wrap_content”
android:id=“@+id/editText2”
android:layout_below=“@+id/textView2”
android:layout_alignParentLeft=“true”
android:layout_alignParentStart=“true”
android:layout_alignRight=“@+id/editText”
android:layout_alignEnd=“@+id/editText” />

<TextView
android:layout_width=“wrap_content”
android:layout_height=“wrap_content”
android:textAppearence=“?android:attr/textAppearanceMedium”
android:text=“Editora:”
android:id=“@+id/TextView3”
android:layout_below=“@+id/editText2”
android:layout_alignParentLeft=“true”
android:layout_alignParentStart=“true”
android:layout_marginTop=“64dp” />

<EditView
android:layout_width=“wrap_content”
android:layout_height=“wrap_content”
```

```
android:id="@+id/editText3"
android:layout_below="@+id/textView3"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_alignRight="@+id/editText2"
android:layout_alignEnd="@+id/editText2" />

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Cadastrar"
android:id="@+id/button"
android:layout_below="@+id/editText3"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_marginTop="72dp"
android:layout_alignRight="@+id/editText3"
android:layout_alignEnd="@+id/editText3" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearence="?android:attr/textAppearanceMedium"
android:id="@+id/TextView4"
android:layout_alignParentBottom="true"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true" />

</RelativeLayout>
```

É possível ver na figura a seguir, como fica representada a interface do aplicativo em um dispositivo móvel.

Figura 5 - Layout do cadastro de livro de um aplicativo desenvolvido em Android, utilizando o banco de dados nativo SQLite e a ferramenta Android Studio. Fonte: Elaborada pelo autor, adaptada de Fenton one, Shutterstock, 2018.

#PraCegoVer: Desenho de um celular com os campos Título, Autor e Editora para serem preeenchidos e embaixo o botão cadastrar.

O código ainda não apresenta nenhuma funcionalidade, pois não foi programado

internamente ao *Activity* que fará o controle. Existem duas classes contidas no código: a classe *Activity*, cujo objetivo é controlar a interface, e a classe BancoController, cujo objetivo é manter o controle das operações no banco de dados.

A classe BancoController faz o controle das manipulações no banco de dados, e é necessária para a execução do aplicativo. O atributo banco é repassado na instanciação do construtor das classes. De forma semelhante, o resultado do método getWritableDatabase() é passado ao atributo db. Esta ação faz a comunicação ao sistema operacional Android para a leitura e manipulação dos dados. Para a inserção dos dados, o método insert() recebe a tabela de dados ao final, encerrando a conexão.

Na figura a seguir, são demonstradas as classes BancoController e InsereDado.

```
public class BancoController {

    private SQLiteDatabase db;
    private CriaBanco banco;

    public BancoController(Context context){
        banco = new CriaBanco(context);
    }

    public String insereDado(String titulo, String autor, String editora){
        ContentValues valores;
        long resultado;

       db = banco.getWritableDatabase();
       valores = new ContentValues();
       valores.put(CriaBanco.TITULO, titulo);
       valores.put(CriaBanco.AUTOR, autor);
       valores.put(CriaBanco.EDITORA, editora);

       resultado = db.insert(CriaBanco.TABELA, null, valores);
       db.close();

       if (resultado ==-1)
           return "Erro ao inserir registro";
       else
           return "Registro Inserido com sucesso";

    }
}

public class ClassInsereDado extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inicial);

        Button botao = (Button)findViewById(R.id.button);

        botao.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                BancoController crud = new BancoController(getBaseContext());
                EditText titulo = (EditText)findViewById(R.id.editText);
                EditText autor = (EditText)findViewById((R.id.editText2));
                EditText editora = (EditText)findViewById(R.id.editText3);
                String tituloString = titulo.getText().toString();
                String autorString = autor.getText().toString();
                String editoraString = editora.getText().toString();
                String resultado;

                resultado = crud.insereDado(tituloString,autorString,editoraString);

                Toast.makeText(getApplicationContext(), resultado, Toast.LENGTH_LONG).show();
            }
        });
    }
}
```

Figura 6 - Classes de inserção de dados em Android, utilizando o banco de dados nativo SQLite e a ferramenta Android Studio. Fonte: Elaborado pelo autor, 2018.

#PraCegoVer: Print da tela com o código:

pubic class BancoController{

```
private SQLiteDatabase db;
private CriaBanco banco;

public BancoController(Context context){
banco = new CriaBanco(context);
}

Public String insereDado(String titulo, String autor, String editora) {
ContentValue valores;
Long resultado;

db = banco.getWritableDatabase();
valores = new ContentValue();
valores.put(CriaBanco.TITULO, titulo);
valores.put(CriaBanco.AUTOR, autor);
valores.put(CriaBanco.EDITORA, editora);

resultado = db.insert(CriaBanco. TABELA, null, valores);
db.close();

if (resulado ==-1)
return “Erro ao inserir registro”;
else
return “Registro Inserido com sucesso”;
}
}

public class ClassInsereDado extends Activity{

@Override
protected void onCreate (Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_inicial);

Buttom botao = (Button)findViewById(R.id.button);
```

```
botao.setonClickListener(new View.onClickListenner() {
@Override
public void onClick(View v) {
BancoController crud = new BancoController(getBaseContext());
EditText titulo = (EditText)findViewById(R.id.editText);
EditText autor = (EditText)findViewById(R.id.editText2);
EditText editora = (EditText)findViewById(R.id.editText3);
String tituloString = titulo.getText().toString();
String autorString = autor.getText().toString();
String editoraString = editora.getText().toString();
String resultado;

resultado = crud.insereDado(tituloString,autorString,editoraString);
Toast.makeText(getApplicationContext(),resultado, Toast.LENGTH_LONG).show();
}
}};
}
}
```

Para fazer a consulta aos dados, precisamos criar a interface, que apresenta um ListView, que é um arquivo de XML de *layout* para sua estilização, como é mostrado na figura a seguir.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="br.com.home.bancodedados.Consulta">


    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listView"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

Figura 7 - Código da Interface do usuário de consulta desenvolvido com o banco de dados SQLite e a ferramenta Android Studio. Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: Print da tela com o código:

<RelativeLayout xmlns?android=http://schemas.android.com/apk/res/android
xmlns:tools=http://schemas.android.com/tools
android:layout_width=”match_parent”
android:layout_height=”match_parent”
android:paddingLeft=”@dimen/activity_horizontal_margin”
android:paddingRight=”@dimen/activity_horizontal_margin”
android:paddingTop=”@dimen/activity_vertical_margin”
android:paddingBottom=”@dimen/activity_vertical_margin”
tools:context=”br.com.home.bancodedados.Consulta”>

<ListView
android:layout_width=“wrap_content”
android:layout_height=“wrap_content”
android:id=“@+id/listView”
android:layout_alignParentLeft=“true”
android:layout_alignParentStart=“true” />
</RelativeLayout>

O resultado da criação do *layout* pode ser conferido na figura a seguir**.**

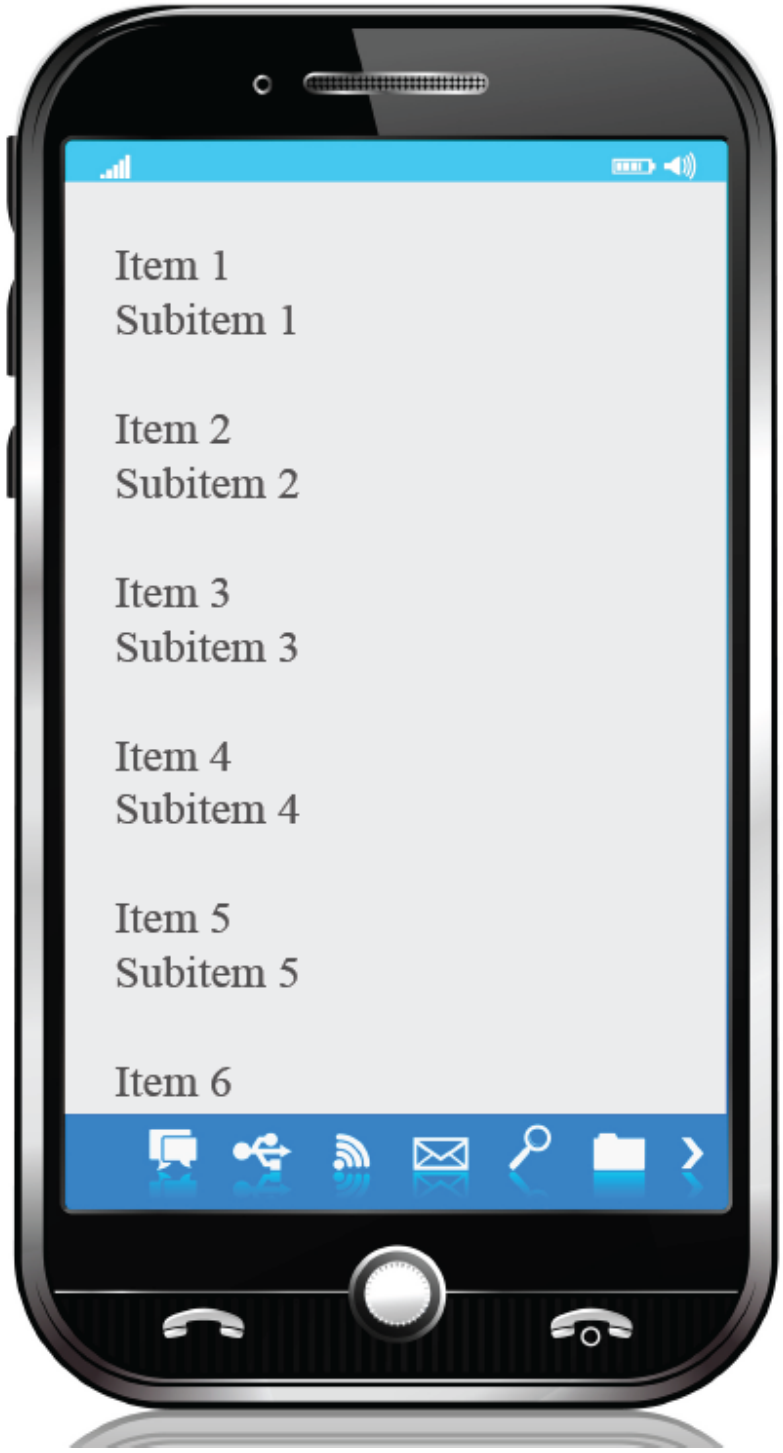


Figura 8 - Layout de consulta de livro de um aplicativo desenvolvido em Android, utilizando o banco de dados nativo SQLite e a ferramenta Android Studio. Fonte: Elaborada pelo autor, adaptada de Fenton one, Shutterstock, 2018.

#PraCegoVer: Desenho de um celular com os campos Item 1 Subitem 1, embaixo Item 2 Subitem 2, embaixo Item 3 Subitem 3, Item 4 Subitem 4, Item 5 Subitem 5, Item 6.

Neste exemplo, foi possível entender o desenvolvimento de um aplicativo Android, com as funcionalidades de inclusão e consulta, utilizando o banco de dados nativo SQLite e a ferramenta Android Studio.

2.2 Processamento assíncrono

m determinadas situações, pode ser necessário que mais de um processo seja executado simultaneamente. Isto se chama processo assíncrono. Para isso, os processos são realizados simultaneamente, sem a necessidade que um termine para outro ser executado. Como esta função é executada em dispositivos móveis? Como o Android trata os processamentos assíncronos? A seguir, serão apresentados os processamentos assíncronos do sistema operacional Android.

Processamento com *Handlers*, *AsyncTasks* e *Loaders*

Nos dispositivos móveis, é possível realizar processos que são executados ao mesmo tempo, de duas formas: pelo paralelismo ou pelo pseudoparalelismo. Um processo é realizado em paralelo, se houver mais de um núcleo no processador, sendo que cada processo é realizado em um núcleo. Caso seja realizado mais de um processo em um núcleo, ocorrerá o pseudoparalelismo. Este procedimento escalona e processa parte de cada tarefa por vez. O *time-slice* tem a função de retirar parte de cada processo e executar a parte de outro colocando no final de uma fila de processos que estão prontos para serem executados. No Android são utilizados os *threads*, que permitem a criação de linhas de execução a serem processadas em paralelo.

VOCÊ SABIA?

Como os *smartphones* usam uma arquitetura de *software* semelhante à dos PCs, eles tem as mesmas vulnerabilidades, mas a diferença é que seus recursos são limitados e impedem a integração de soluções avançadas de monitoramento de segurança. No artigo "*Secloud: a cloud-based comprehensive and lightweight security solution for smartphones*", o pesquisador Zonous e seus colaboradores (2013) propõem a Secloud, uma solução de segurança baseada em nuvem, para dispositivos *smartphone,* demonstrando o uso dos recursos do sistema operacional Android:

<https://www.sciencedirect.com/science/article/pii/S016740481300031X (https://www.sciencedirect.com/science/article/pii/S016740481300031X)>.

A atualização da interface gráfica, a partir das informações processadas, precisa de *handler*s e, para isso, a classe AsyncTask foi criada. Ela permite que esses processos rodem em segundo plano, a fim de atualizar os dados na Interface. Deitel, Deitel e Wald (2016, p. 245), descrevem o processo da seguinte forma:

> Quando precisarmos dos resultados de uma operação de banco de dados na *thread* da interface gráfica, vamos usar uma subclasse de AsyncTask (pacote Android.os) para efetuar a operação em uma *thread* e receber os resultados na *thread* da interface. Os detalhes da criação e manipulação de *thread*s são tratados pela classe AsyncTask, assim como a comunicação dos resultados de AsyncTask para a *thread* da interface gráfica do usuário.

Assim, vemos que a classe AsyncTask tem a função de encapsular a criação das funções *thread*s e *handler*. A criação da classe é feita pela ação AsyncTask<Parâmetro, Progresso, Resultado>, na qual os parâmetros passados são o tipo de dados, o Progresso e o retorno da *thread* (Resultado). A classe AsyncTask compreende outros métodos sobrescritos que são o onPreExecute (antes de se iniciar a *thread*), o método doInBackground (processamento em paralelo e *background,* para processamentos pesados), o método onPostExecute (retorno do doInBackground) e onProgressUpdate (percentual do *download* para a interface) (DEITEL; DEITEL; WALD, 2016).

Serviços no Android

Simões e Pereira (2014) explicam que serviço é uma parte de um aplicativo, com operação mais pesada e que não interage com a interface. Um serviço é iniciado por um determinado componente do aplicativo, sendo executado em segundo plano, mesmo havendo alteração entre os aplicativos pelo usuário (DEITEL; DEITEL; WALD, 2016). Conforme descrito por Simões e Pereira (2014), pode ter duas formas:

- **iniciado**. A iniciação de um serviço é realizada pelo método startService(), ficando em execução em segundo plano, mesmo que o componente que o iniciou seja cancelado. Quando a operação for concluída, o serviço deverá ser interrompido;
- **vinculado**. Um serviço é "vinculado" quando um componente do aplicativo chama bindService() para vinculá-lo. Um serviço vinculado oferece uma interface servidor-cliente que permite que os componentes interajam com a interface, enviem solicitações, obtenham resultados, mesmo em processos com Comunicação Interprocessual (IPC). Um serviço vinculado permanece em execução somente enquanto outro componente do aplicativo estiver vinculado a ele. Vários componentes podem ser vinculados ao serviço de uma só vez, mas quando todos desfizerem o vínculo, o serviço será destruído.

Assim, um serviço iniciado é o que permanece em execução indefinidamente em segundo plano, mas pode também permitir a vinculação, sendo que este serviço permanece em execução, enquanto outro componente estiver associado a ele.

VOCÊ SABIA?

O modelo de programação assíncrona é comumente usado em sistemas móveis e ambientes *web* 2.0. Porém, usam algoritmos que são de magnitude de desempenho e espaço ineficiente, em comparação com os detectores de dados convencionais. Pensando nisso, o pesquisador Hsiao e seus colaboradores (2017) resolveram identificar e abordar dois problemas importantes no raciocínio sobre a causalidade entre eventos assíncronos. Eles propõem uma nova classe chamada AsyncClock, que resolve esse problema, explicitamente rastreando eventos causalmente precedentes.

Se o aplicativo for iniciado ou vinculado, outros componentes do aplicativo poderão usar o serviço, da mesma forma que qualquer componente poderá usar uma atividade — iniciando com uma *Intent*. Portanto, é possível declarar o serviço como privado, no arquivo do *manifest*, e bloquear o acesso de outros aplicativos.

*Download d*e imagens com o Google Glide

O Google Glide é um aplicativo que permite que usuários troquem mensagens de vídeo em tempo real, bastante útil quando incorporado ao uso corporativo ou, mesmo, pessoal.

VOCÊ O CONHECE?

Sundar Pichai é o atual CEO da Google, que adquiriu o sistema operacional Android Inc, em agosto de 2005. Em 2004, ele liderou na Google, o gerenciamento de produtos como o Google Chrome e Chrome OS e, com o passar do tempo, começou a liderar outros projetos, como o sistema operacional Android. Leia mais no artigo escrito por Griffith (2015), disponível em <https://www.pcmag.com/article2/0,2817,2489388,00.asp (https://www.pcmag.com/article2/0,2817,2489388,00.asp)>.

Conforme exposto por Goovaerts (2016), o Google Glide permitirá que usuários façam e recebam chamadas por Bluetooth e escutem mensagens de áudio e vídeo, por meio de aplicativos. O Glide é uma biblioteca do Image Loader para Android, desenvolvida e recomendada pela Google. Ele foi usado em muitos projetos de código aberto da Google, incluindo o aplicativo oficial Google I / O 2014. Ele fornece suporte a GIF animado e lida com o carregamento / armazenamento de imagens. Glide é uma biblioteca de carregamento de imagens rápida e eficiente para Android, focada em rolagem suave. O Glide oferece uma API fácil de usar, um *pipeline* de decodificação de recursos e extensíveis e um conjunto automático de recursos. Glide suporta busca, decodificação e exibição de fotos de vídeo, imagens e GIFs animados. O Glide inclui uma API flexível, que permite aos desenvolvedores conectar-se a praticamente qualquer pilha de rede. Por padrão, o Glide usa uma pilha baseada em HttpUrlConnection, mas também inclui bibliotecas de utilitários no projeto Volley da Google ou na biblioteca OkHttp da Square. O Glide leva em consideração dois aspectos principais do desempenho do carregamento de imagens no Android:

- a velocidade na qual as imagens podem ser decodificadas;

- o montante de resíduos ocorridos durante a decodificação de imagens.

Para que os usuários tenham uma ótima experiência com um aplicativo, as imagens não devem aparecer apenas rapidamente, mas devem também fazê-lo sem causar muita interferência e engasgos com a entrada e saída do *thread* principal, ou com o excesso de coleta de lixo.

O Glide realiza várias etapas para garantir que o carregamento de imagens seja o mais rápido e suave possível no Android:

- a redução de escala e o armazenamento em cache inteligentes e automáticos minimizam os tempos de sobrecarga e decodificação do armazenamento.
- a reutilização agressiva de recursos, como matrizes de *bytes* e *bitmaps*, minimiza as dispendiosas coletas de lixo e a fragmentação de *heap*.
- a integração profunda do ciclo de vida garante que apenas as solicitações para fragmentos e atividades ativos sejam priorizadas, e que os aplicativos liberem recursos, quando necessário, para evitar serem eliminados, quando estão em segundo plano.

O foco principal do Glide é tornar a rolagem, de qualquer tipo de lista de imagens, o mais suave e rápida possível, mas o Glide também é eficaz em quase todos os casos em que você precisa buscar, redimensionar e exibir uma imagem remota.

2.3 Receptores e notificações

A forma como as informações são recebidas e notificadas nos diversos sistemas operacionais são diferenciadas. Como o sistema operacional Android efetua as notificações para a interface do usuário?

Vamos acompanhar agora como o Android trabalha com os receptores e notificações no seu sistema operacional.

JobScheduler

O componente JobScheduler tem como objetivo planejar os processos, para tornar o processamento mais eficiente, inclusive reduzindo o consumo de energia. Neste sentido, são criadas classes lógicas para o gerenciamento dos sinais. É criada, também, a classe SynAdapater para executar tarefas em *background*, tornando mais ágeis os processos em execução. O JobScheduler não consegue ter o controle dos serviços escalonados, sendo que o espaço de intervalo de execução é definido pelos métodos setPeriodic(), setBackoffCriteria() e setOverrideDeadline().

Para Meike (2016, p. 155), "a maior motivação para a criação do JobScheduler é um controle mais refinado do agendamento das tarefas assíncronas [...] a arquitetura do JobSchedule é um modelo que está entre as classes sync-adapter e AlarmManager". Quando se processa o JobScheduler, são necessárias três etapas: tornar a tarefa acessível pelo JobScheduler, agendar a tarefa e implementar o agendamento.

BroadcastReceiver

Um BroadcastReceiver é um componente que responde aos eventos de transmissão em todo o sistema. Muitas transmissões são originadas do próprio sistema, como uma transmissão anunciando que a tela foi desligada ou a bateria está fraca. Os aplicativos também podem iniciar transmissões para permitir que outros aplicativos saibam que alguns dados foram baixados para o dispositivo e estão disponíveis para serem usados. Embora os receptores de transmissão não exibam, necessariamente, uma interface de usuário, eles podem criar uma notificação de barra de *status* para alertar o usuário de algum evento de transmissão. O mais comum porém, é que um receptor de *broadcast* seja apenas um canal para outros componentes e tenha o objetivo de fazer uma quantidade mínima de trabalho. Um receptor de *broadcast* é implementado como uma subclasse de BroadcastReceiver e cada *broadcast* é entregue como um objeto *Intent*.

Notificações no Android

No sistema operacional Android, uma notificação significa uma mensagem enviada ao usuário, de uma forma normal. Estas notificações são realizadas por meio de um ícone. O armazenamento destas notificações é gerenciado pelo próprio sistema operacional Android, sendo que o usuário do sistema pode ter acesso a este repositório. As classes que são utilizadas são o objeto NotificationCompat.Builder(), para notificar as ações, o método NotificationCompat.Builder.build(), para criar a notificação, e o método NotificationManager.notifiy(), para emitir a notificação.

Interface avançada do usuário

A interface com o usuário de um dispositivo móvel tem se desenvolvido de maneira rápida e com recursos cada vez mais avançados. No sistema operacional Android diversas novas funcionalidades foram criadas para atender de forma mais eficiente às mais variadas demandas dos seus usuários.

Drag and drop*, *drawables*, *parcelables*, *styles*, *themes* e *widgets

Com a estrutura de trabalho de arrastar e soltar (*drag and drop*) do Android é possível permitir que usuários movam dados de uma visão para outra no *layout* atual, usando um gesto gráfico de arrastar e soltar. A estrutura de trabalho contém uma classe de eventos de arrastar, e métodos e classes auxiliares.

Embora a estrutura de trabalho seja projetada principalmente para movimentação de dados, é possível usá-la para outras ações de interface com o usuário. Por exemplo, pode-se criar um aplicativo que mistura cores, quando o usuário arrasta um ícone colorido sobre outro ícone.

É possível criar um objeto de evento de arrastar ("*listening*") de uma classe que programe a classe View.OnDragListener. O objeto do evento de arrastar para uma visão é definido com o método setOnDragListener(). Cada objeto também tem um método de retorno de chamada onDragEvent(). Ambos estão descritos em mais detalhes.

Estilos e temas no Android permitem separar os detalhes do *design* de seu aplicativo da estrutura e do comportamento da interface do usuário, semelhante ao CSS (*Cascading Style Sheets*), no *design* da *web*. Um estilo é uma coleção de atributos que especificam a aparência de uma única visão. Um estilo pode

especificar atributos como cor da fonte, tamanho da fonte, cor do plano de fundo e muito mais.

Um tema é um tipo de estilo aplicado a uma hierarquia inteira de aplicativos, atividades ou visualizações, não apenas uma visualização individual. Quando se aplica seu estilo como tema, todas as visualizações no aplicativo se aplicam a cada atributo de estilo que ele suporta. Os temas também podem aplicar estilos a elementos que não são de visualização, como a barra de status e o plano de fundo da janela. Estilos e temas são declarados em um arquivo de recurso de estilo chamado *styles.xml*.

Quando é necessário exibir imagens estáticas em um aplicativo Android, poderá ser usada a classe Drawable e suas subclasses para desenhar formas e imagens. A Drawable é uma abstração geral para algo que pode ser desenhado. As várias subclasses ajudam com cenários específicos de imagem e pode estendê-las para definir seus próprios objetos, que se comportam de maneiras únicas. Existem duas maneiras de definir e instanciar um Drawable, além de se usar os construtores de classe:

- acrescentar um recurso de imagem (um arquivo de *bitmap*) salvo em seu projeto;
- acrescentar um recurso XML que define as propriedades *drawable*.

No sistema operacional Android é possível adicionar gráficos ao aplicativo referenciando um arquivo de imagem dos recursos do seu projeto. Os tipos de arquivos suportados são PNG, JPG e GIF. Ícones de aplicativos, logotipos e outros elementos gráficos, como os usados em jogos, são adequados para essa técnica. Para usar um recurso de imagem, é necessário adicionar o arquivo ao diretório *drawable* do projeto. Uma vez no projeto, pode-se referenciar o recurso de imagem do código ou seu *layout* XML.

VOCÊ QUER VER?

Na palestra de Ygor Barbosa (*Mobile Engineer* da Nubank Brasil) e Thales Machado (*Software Engineer* da Nubank Brasil) (2017), você pode conhecer a experiência dos dois desenvolvedores com a arquitetura

dos aplicativos Android, na Conferência Android Dev de 2017. Para assistir acesse <https://www.youtube.com/watch?v=Vgbe52XJkNU (https://www.youtube.com/watch?v=Vgbe52XJkNU)>.

Parcelables e os Bundles são objetos que devem ser usados nos limites do processo, como nas transações IPC/Binder, entre as atividades, com o objetivo de armazenar o estado transitório nas alterações de configuração. O Bundle é um armazenamento de chave/valor para objetos especializados, usado para passar dados para componentes do Android. Se for preciso passar os objetos de cliente via Bundle, é necessário implementar a Parcelable interface.

A Parcelable é a implementação do Android do Java Serializable que assume a estrutura de processamento. Dessa forma, um Parcelable pode ser processado de forma relativamente rápida, comparada com a serialização Java padrão. Para permitir que o objeto seja analisado para outro componente, é preciso implementar a Android.os.Parcelableinterface. Ele também fornece um método final estático chamado CREATOR, que deve implementar a Parcelable.Creatorinterface. “O objeto Parcelable é um objeto que implementa uma API específica que requer dois métodos: o describeContent() e o writeParcel()” (MEIKE, 2016, p. 116).

O sistema operacional Android oferece as funcionalidades de *widgets*, que são pequenos aplicativos dispostos na tela inicial, que tornam o acesso mais rápido às informações como previsão do tempo, hora e agenda, sem que seja necessário abrir o aplicativo correspondente..

Composição e customização de *views*

Um conjunto de serviços para desenvolvimento, ou SDK (*Software Development Kits*), faz parte do sistema operacional Android para a construção dos componentes da interface. Estes componentes podem ser *widgets* ou *layouts*. As classes View e ViewGroup são responsáveis por construir os componentes. Em meio a estes componentes, estão os já conhecidos em outras plataformas como Button, TextView, EditText, ListView, CheckBox, RadioButton e ImageView.

Resource selectors

Os dispositivos Android vêm em uma variedade de configurações diferentes de tamanho, densidade de *pixels* da tela, configurações de idioma, etc. O Android suporta a seleção automática de recursos que se ajustam à configuração do dispositivo. Para isso, o sistema fornece recursos diferentes em subpastas predefinidas no diretório. O nome do sufixo de uma subpasta define a configuração do dispositivo para a qual ela é válida, por exemplo, orientação, resolução ou idiomas.

Temos, como exemplo, as telas de dispositivos Android, que são diferentes em termos de resolução e em termos de densidade dos *pixels* na tela. No caso de diferentes densidades de tela, o mesmo comprimento de um componente de interface do usuário consiste em uma quantidade diferente de *pixels*. Por exemplo, se o desenvolvedor especificar os *pixels* diretamente, poderá obter um determinado resultado em um dispositivo com um número relativamente pequeno de *pixels*. Por esta razão, é necessário fornecer recursos gráficos, por exemplo, os ícones da barra de ação e *laucher* em diferentes resoluções. Isso pode ser feito usando qualificadores de recursos com base na densidade de pixel. É possível usar a densidade como qualificador de recurso. O Android suporta as seguintes seleções: a linha de base é de 160 pontos por polegada (ppp). Se, por exemplo, o dispositivo Android tiver 320 dpi, o *drawable* do *drawable-hdpi* será selecionado.

Uso de animações

As animações podem adicionar dicas visuais que avisam os usuários sobre o que está acontecendo no aplicativo. Eles são especialmente úteis quando a interface do usuário muda de estado, como quando novos carregamentos de conteúdo ou novas ações ficam disponíveis. As animações também adicionam uma aparência elegante ao aplicativo, o que proporciona maior qualidade.

O *Property Animation* é um *framework* disponível no Android e foi desenvolvido com o objetivo de fornecer mais robustez e controle, ao desenvolver animações. O *Property Animation* permite que qualquer objeto seja animado, seja este visível na interface ou não. O funcionamento do *Property Animation* parte do princípio de que é possível animar qualquer atributo de um objeto por um período de tempo, como também definir as características importantes:

- **duração**: o tempo total de execução da animação que, se não alterado, permanece com o tempo *default* de 300ms;
- **interpolação do tempo**: como os valores da animação serão calculados em função do tempo.

O Android dispõe de várias interpolações que não passam de fórmulas, que podem ser copiadas e modificadas, para satisfazer o objeto do aplicativo.

CASO

Uma empresa de gestão de projetos percebeu que, um dos maiores problemas que enfrentava, era devido aos atrasos no acompanhamento das tarefas dos projetos. Os gestores de projetos tinham problemas de acompanhar o desenvolvimento e entrega de todas as tarefas. Para contornar a situação, a empresa desenvolveu um aplicativo em Android com a API 21, conhecida como o Android Lollipop. O Google forneceu um novo componente conhecido como JobScheduler API, para lidar com esse mesmo cenário. A JobScheduler API realiza uma operação para o aplicativo quando um conjunto de condições predefinidas é atendido. Além disso, a JobScheduler é capaz de agrupar várias tarefas para serem executadas juntas. Isso permite que o aplicativo execute e informe a tarefa determinada, para que os gestores de projetos possam resolver os problemas de acompanhamento.

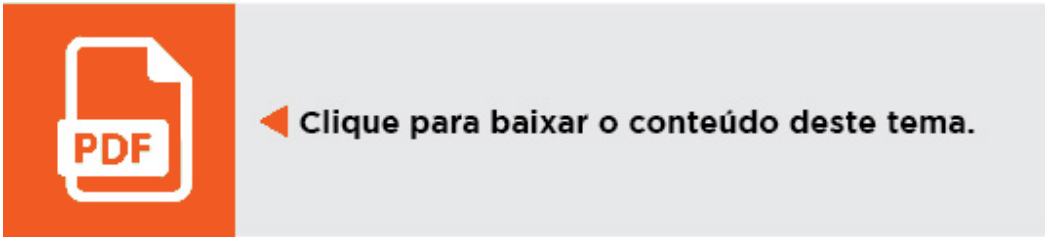
O Android inclui diferentes APIs (API é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de *software,* ou plataforma, baseado na *web*) de animação, dependendo do tipo de animação que se deseja. Portanto, essa página fornece uma visão geral das diferentes maneiras de adicionar movimento à interface do usuário. Quando se deseja animar um gráfico de *bitmap*, como um ícone ou uma ilustração, deve usar as APIs de animação desenhados. Geralmente, essas animações são definidas estaticamente com um recurso *drawable*, mas também pode definir o comportamento da animação no tempo de execução.

Síntese

Compreendemos, neste capítulo, como utilizar e implantar os recursos e APIs do sistema Android, de forma a tornar a interface de usuário mais dinâmica. Foi visto como o sistema operacional Android oferece classes que permitem o acesso à rede, ao armazenamento de dados e como fazer a persistência de dados. Ao implantar um aplicativo em um dispositivo móvel, verificamos a importância dos processamentos assíncronos e as notificações, agregando à interface, melhores funcionalidades para o usuário.

Neste capítulo, você teve a oportunidade de:

- compreender e aplicar as principais classes e métodos de armazenamento e manipulação de dados por meio do SQlite, o processamento XmlPullParser, o formato JSON e a ferramenta Retrofit;
- demonstrar e executar processamentos assíncronos utilizando *handler*s e AsuncTaksas, e as manipulações de imagens e serviços no sistema operacional Android utilizando o Google Glide;
- identificar o serviço de agendamento e notificações por meio do JobSchedulere BroadcastReceiver;
- prender a utilizar os recursos de interface do Android como o *drag and drop*, *drawables*, *pacelables*, *styles*, *themes* e *widgets*, com a possibilidade de criar animações e o *resource selectors*.



Bibliografia

ALMEIDA, L.; FERNANDES, I.; COSTA, C. SISCI - Sistema para Controle de Irrigação através de Dispositivos Celulares. **HOLOS**, 2012, Vol.28(1), pp.147-156. Disponível em <http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/681 (http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/681)>. Acesso em: 21/06/2018.

ANDRADE, A. W.; AGRA, R.; MALHEIROS, V.. **Estudos de caso de aplicativos móveis**

no governo brasileiro. Documento oficial do SERPRO - Serviço Federal de Processamento de Dados. SGAN 601 Modulo V - 70836-900 – Brasília – DF – Brasil, 2013. Disponível em: <https://sol.sbc.org.br/index.php/sbsi/article/download/5740/5637/ (https://sol.sbc.org.br/index.php/sbsi/article/download/5740/5637/)>. Acesso em: 06/06/2018.

BARBOZA, Y.; MACHADO, T. **Arquitetura de um app Android - Thales Machado e Ygor Barboza**. Canal iMasters, YouTube, publicado em 25 de out de 2017. Disponível em: <https://www.youtube.com/watch?v=Vgbe52XJkNU (https://www.youtube.com/watch?v=Vgbe52XJkNU)>. Acesso em: 21/06/2018.

CARVALHO, A. A.; ARAUJO, I.; FONSECA, A. Das Preferências de Jogo à Criação do Mobile Game Konnecting: um estudo no ensino superior. **RISTI** [online]. 2015, n.16, pp.30-45. Disponível em <http://dx.doi.org/10.17013/risti.16.30-45 (http://dx.doi.org/10.17013/risti.16.30-45)>. Acesso em: 21/06/2018.

COSTA, C; *et al*. Monitoramento de Plataformas de Poços de Petróleo através de Dispositivos Móveis. **HOLOS**, 2012, Vol.28(3), pp. 60-74. Disponível em <http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/614 (http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/614)>. Acesso em: 21/06/2018.

DEITEL, P. J.; DEITEL, H.; WALD, A. **Android 6 para Programadores**: uma abordagem baseada em aplicativos. 3. ed. Porto Alegre: Bookman, 2016.

FERNANDES, C. B. **Solução de auxílio ao diagnóstico e à pesquisa médica baseada em mineração de dados utilizando interface Android.** Dissertação de mestrado em Engenharia Elétrica da Pontifícia Universidade Católica do Rio Grande do Sul, 2013. Disponível em <http://hdl.handle.net/10923/3199 (http://hdl.handle.net/10923/3199)>. Acesso em: 21/06/2018.

FONSECA FILHO, C. **História da Computação**: o Caminho do Pensamento e da Tecnologia. Porto Alegre: EDIPUCRS, 2007. Disponível em <http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf (http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf)>. Acesso em: 21/06/2018.

GOOVAERTS, D. Android Wear Update to Support Speakers, Gestures, LTE. **Wireless Week**, Feb 5, 2016.

GRIFFITH, E. **15 Things to Know About Google CEO Sundar Pichai**. PCmag.com, Aug 11, 2015.

HSIAO, C.; *et al*. AsyncClock: Scalable inference of asynchronous event causality ASPLOS 2017. **22nd International Conference on Architectural Support for Programming Languages and Operating Systems.** p. 193-205, 2017. Disponível em <https://dl.acm.org/doi/abs/10.1145/3037697.3037712 (https://dl.acm.org/doi/abs/10.1145/3037697.3037712)>. Acesso em: 21/06/2018.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações Móveis**: arquitetura, projeto e desenvolvimento. Pearson Education do Brasil: São Paulo, 2005.

MACK, R. S. **Sistema de recomendação baseado na localização e perfil utilizando a plataforma** Android. Universidade Federal do Rio Grande do Sul. Departamento de Ciência da Computação, 2010. Disponível em <http://hdl.handle.net/10183/28328 (http://hdl.handle.net/10183/28328)>. Acesso em: 21/06/2018.

MEIKE, G. B. **Android Concurrency**. New York: Addison-Wesley, 2016.

SOARES, R.; PEREIRA, M.; MARTINS, J. A. Recolha, preservação e contextualização de objectos digitais para dispositivos móveis com Android. **RISTI** [online], n. 9, pp. 75-89, 2012. Disponível em: <http://dx.doi.org/10.4304/risti.9.75-89 (http://dx.doi.org/10.4304/risti.9.75-89)>. Acesso em: 21/06/2018.

PILAR, C. P. **Avaliação das Arquiteturas de Desenvolvimento para Dispositivos Móveis**. Universidade de Caxias do Sul. Departamento de Sistemas de Informação. 2013.

SIMÕES, D. D.; PEREIRA, J. C. Sistemas operacionais móveis - Android x ios. **XVI Semana de Informática de Paranavaí**. 2014.

SUBONIS, T. **Reactive Android Programming**. Pack Publishing. Birmingham, 2017

TANENBAUM, A. S.; BOS, H. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Person Education do Brasil, 2016.

ZONOUS, S. et al. A cloud-based comprehensive and lightweight security solution for *smartphones*. **Computers & Security**. Volume 37, Pages 215-227. September, 2013. Disponível em <https://www.sciencedirect.com/science/article/pii/S016740481300031X (https://www.sciencedirect.com/science/article/pii/S016740481300031X)>. Acesso em: 21/06/2018.