

CAPÍTULO III

Utilizando Google Geocoding API em aplicações Android

Aprenda a converter Geocódigos em endereços e endereços em geocódigos com Google Geocoding API

Capítulo utiliza estudo de caso desenvolvido na IDE:

Uma das preocupações mais antigas da humanidade é saber “onde estou” ou então onde alguém está a nível regional ou até mesmo global. Uma das formas mais tradicionais de identificar a própria localização é com a utilização da bússola, esse equipamento composta basicamente por uma agulha magnetizada sobre uma superfície plana indica sempre a direção do Norte-Sul.

No período conhecido como era do descobrimento, momento em que os europeus descobriram novas terras, a bússola possibilitou que os viajantes, seja de navio ou a cavalo, se localizassem de forma eficiente. Esse instrumento possibilitou que este mesmo percurso pudesse ser refeito posteriormente.

Uma técnica contemporânea para o geolocalização é a utilização do GPS (*Geographic Position System*), este recurso largamente difundido é encontrado em rastreadores de automóveis, aparelhos de orientação de direção e rotas e mais atualmente na maioria dos smartphones e tablets. Com a popularização do desta tecnologia e a possibilidade do desenvolvimento de aplicativos com GPS para smartphone possibilitou inúmeras possibilidades de soluções em geolocalização.

Porém, ao utilizar um GPS, as coordenadas retornadas por ele são a longitude, a latitude e a altitude. A última informação é retornada em metros, já a longitude e latitude são retornadas em graus, minutos e segundos. Nos sistemas computacionais, estas informações se dão no formato decimal, tornando-as mais fáceis de serem tratados.

Mesmo assim, ao recuperar uma determinada posição e armazenando-a em um banco de dados, ao recuperá-las aqueles aglomerados de números não dizem muita coisa por si só. Muitas vezes, para um geocódigo, como são chamados as coordenadas retornadas pelo GPS, fazerem sentidos estas precisam serem transportadas para um mapa, para só então o usuário ter uma ideia do significado daquele geocódigo.

Outra alternativa para familizarizar os usuários de sistemas que usam GPS com os geocódigos é transformar estes em notações textuais, permitindo que o usuário não veja uma longitude e/ou latitude, e sim, um endereço formado por nome de rua, número, bairro, cidade e estado.

Com o objetivo de facilitar o uso de GPS em aplicações Android, este capítulo apresenta algumas APIs para tratamento de geocódigos: a Geocoding API e Google Maps Coordinate API.

### Google Geocoding API

A API Google Geocodig é uma forma de utilizar um geocodificador através de um serviço HTTP. A geocodificação consistem em transformar um coordenadas um endereço, como por exemplo, “Via do Conhecimento, km1, Pato Branco, Paraná” em -26.196223 e -52.689523. Também é possível realizar a operação inversa, transformando coordenadas em endereços.

Este serviço é utilizado normalmente para geocodificar endereços previamente conhecidos para que torne possível a localização do endereço em um mapa. Esta é uma rotina custosa e demorada, e utiliza muitos recursos, por isso o ideal é que sempre que possível o desenvolvedor persista em banco de dados ou em cache os geocódigos de endereços estáticos. Isto irá tonar seus aplicativos mais rápidos, pois irá melhorar o tempo de resposta da sua aplicação, pois as consultas seriam realizadas a esta API apenas quando necessário.

Este serviço da Google está limitado a 2.500 consultas ao dia, quem possui serviço o pacote de serviços empresarial da Google pode realizar até 100.000 consultas por dia. A limitação no número de consultas existe para evitar abuso na utilização da Google Geocoding API. É uma determinação da empresa que disponibilizar este serviço que este seja utilizado apenas em conjunto com uma mapa do Google.

Para utilizar esta API basta consumir o seguinte endereço web via URL.

**http**://maps.googleapis.com/maps/api/geocode/*output*?*parameters*

Onde o parâmetro “output” deve definir o formato da resposta da solicitação, sendo possível informar “json” ou “xml”. A recomendação da própria Google é a utilização do formato JSON (Notação de Objetos JavaScript).

Existe a possibilidade de utilizar o protocolo HTTP ou HTTPS. O segundo é recomendado quando os parâmetros informados são confidenciais, e o protocolo HTTPS provê a proteção das informações informadas por parâmetro. Independentemente do protocolo escolhido o formato dos parâmetros de envio ou o conteúdo da resposta será o mesmo.

Os *parametes* devem seguir os padrões de URLs, ou seja, cada parâmetro deve ser separado pelo caractere “&” (“e” comercial). Alguns parâmetros são opcionais, outros são obrigatórios, como veremos abaixo.

**Formato JSON**

O formato JSON é uma notação leve de troca de dados, é considerada fácil de ler e escrever, para o computador é fácil de gerar e interpretar, sendo a opção recomendada pelo Google para o uso da leitura de geocode, já que o tamanho da mensagem de resposta em bytes é menor do que a utilização em XML.

O resultado de uma requisição JSON é bastante grande e completa, assim para exemplificar, a **Listagem 3.1** apresenta apenas parte de uma mensagem retornada pelo Google Geocoding API. Observem que os principais dados, em especial que se refere ao endereço se encontra no corpo da mensagem.

**Listagem 3.1.** Código JSON em resposta a uma requisição para transformar geocódigo em endereço pela Google Geocoding API.

01. {

02. "results" : [

03. {

04. "address\_components" : [

05. {

06. "long\_name" : "333-343",

07. "short\_name" : "333-343",

08. "types" : [ "street\_number" ]

09. },

10. {

11. "long\_name" : "Avenida Elisa Rosa Cola Padoan",

12. "short\_name" : "Av. Elisa Rosa Cola Padoan",

13. "types" : [ "route" ]

14. },

15. {

16. "long\_name" : "Fraron",

17. "short\_name" : "Fraron",

18. "types" : [ "sublocality\_level\_1", "sublocality", "political" ]

19. },

20. {

21. "long\_name" : "Pato Branco",

22. "short\_name" : "Pato Branco",

23. "types" : [ "locality", "political" ]

24. },

25. {

26. "long\_name" : "Pato Branco",

27. "short\_name" : "Pato Branco",

28. "types" : [ "administrative\_area\_level\_2", "political" ]

29. },

30. {

31. "long\_name" : "Paraná",

32. "short\_name" : "PR",

33. "types" : [ "administrative\_area\_level\_1", "political" ]

34. },

35. {

36. "long\_name" : "Brasil",

37. "short\_name" : "BR",

38. "types" : [ "country", "political" ]

39. },

40. {

41. "long\_name" : "85508",

42. "short\_name" : "85508",

43. "types" : [ "postal\_code\_prefix", "postal\_code" ]

44. }

45. ],

46. "formatted\_address" : "Av. Elisa Rosa Cola Padoan, 333-343 - Fraron, Pato Branco - PR, Brasil",

47. "geometry" : {

48. "bounds" : {

49. "northeast" : {

50. "lat" : -26.1965843,

51. "lng" : -52.6890572

52. },

53. "southwest" : {

54. "lat" : -26.1966978,

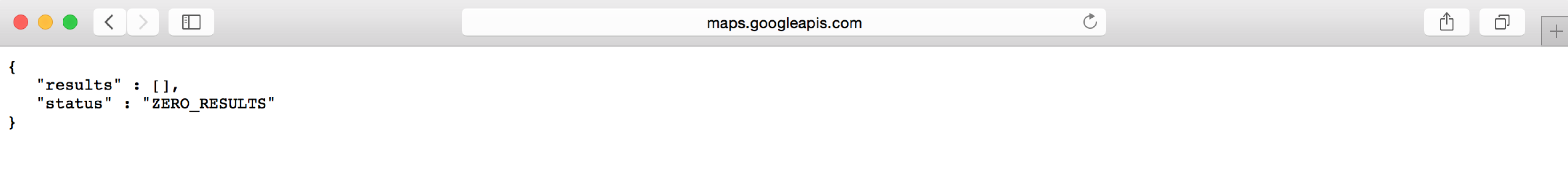
55. "lng" : -52.6892667

56. }

57. }

A resposta JSON possui dois elementos básicos, o primeiro elemento é o *results* que contém uma matriz de informações de endereços geocodificados e informações de geometria. O segundo elemento é o *status* que contém metadados sobre a solicitação, no exemplo acima retornou OK, o que significa que não houve erros, o endereço foi analisado e ao menos um resultado foi retornado.

Existem outros resultados possíveis, ZERO\_RESULTS indica que nenhum resultado foi retornado, normalmente acontece porque o endereço não foi encontrado. Outro exemplo é o OVER\_QUERY\_LIMIT que indica que foi ultrapassada a cota diária de consultas. Um exemplo de erro ZERO\_RESULTS é apresentado na **Figura 3.1**.

****

**Figura 3.1.** Tela de um browser após uma pesquisa sem resultado.

**Formato XML**

O XML é uma linguagem de marcação largamente utilizada nas tecnologias computacionais e comunicação com serviços web. É recomendado para documentos com dados com organizações hierárquicas, sendo uma das duas consultas possíveis nesta API. O resultado da mesma consulta acima feito com retorno XML é apresentado na **Listagem 3.2.**

**Listagem 3.2.** Código XML em resposta a uma requisição para transformar geocódigo em endereço pela Google Geocoding API.

01. <GeocodeResponse>

02. <status>OK</status>

03. <result>

04. <type>street\_address</type>

05. <formatted\_address>

06. Av. Elisa Rosa Cola Padoan, 333-343 - Fraron, Pato Branco - PR, Brazil

07. </formatted\_address>

08. <address\_component>

09. <long\_name>333-343</long\_name>

10. <short\_name>333-343</short\_name>

11. <type>street\_number</type>

12. </address\_component>

13. <address\_component>

14. <long\_name>Avenida Elisa Rosa Cola Padoan</long\_name>

15. <short\_name>Av. Elisa Rosa Cola Padoan</short\_name>

16. <type>route</type>

17. </address\_component>

18. <address\_component>

19. <long\_name>Fraron</long\_name>

20. <short\_name>Fraron</short\_name>

21. <type>sublocality\_level\_1</type>

22. <type>sublocality</type>

23. <type>political</type>

24. </address\_component>

25. <address\_component>

26. <long\_name>Pato Branco</long\_name>

27. <short\_name>Pato Branco</short\_name>

28. <type>locality</type>

29. <type>political</type>

30. </address\_component>

31. <address\_component>

32. <long\_name>Pato Branco</long\_name>

33. <short\_name>Pato Branco</short\_name>

34. <type>administrative\_area\_level\_2</type>

35. <type>political</type>

36. </address\_component>

37. <address\_component>

38. <long\_name>Paraná</long\_name>

39. <short\_name>PR</short\_name>

40. <type>administrative\_area\_level\_1</type>

41. <type>political</type>

42. </address\_component>

43. <address\_component>

44. <long\_name>Brazil</long\_name>

45. <short\_name>BR</short\_name>

46. <type>country</type>

47. <type>political</type>

48. </address\_component>

49. <address\_component>

50. <long\_name>85508</long\_name>

51. <short\_name>85508</short\_name>

52. <type>postal\_code\_prefix</type>

53. <type>postal\_code</type>

54. </address\_component>

55. <geometry>

56. <location>

57. <lat>-26.1966585</lat>

58. <lng>-52.6892299</lng>

59. </location>

60. <location\_type>RANGE\_INTERPOLATED</location\_type>

61. <viewport>

62. <southwest>

63. <lat>-26.1979900</lat>

64. <lng>-52.6905109</lng>

65. </southwest>

66. <northeast>

67. <lat>-26.1952921</lat>

68. <lng>-52.6878130</lng>

69. </northeast>

70. </viewport>

A resposta XML possui um elemento raiz chamado *GecodeResponse* e dois elementos principais de nível superior, os nomes dos elementos e o conteúdo no sentido de informações, e não de formato, são os mesmos do padrão JSON, o *result* e o *status*.

Para acesso ao serviço, um conjunto de parâmetros deve ser informados. Para uma melhor organização do texto, primeiramente serão apresentados os parâmetros obrigatórios para consulta, na sequência os parâmetros não obrigatórios e por fim, os Parâmetros para uso em Serviço Empresarial.

**Parâmetros obrigatórios para consulta**

Alguns parâmetros são obrigatórios no momento da consulta, sendo eles:

* address,latlng, components
* sensor

**Address, latlng, components**

O parâmetro *address* deve ser utilizado para indicar o endereço desejado para codificar.

http://maps.googleapis.com/maps/api/geocode/json?address=Via+do+conhecimento,km1,+Pato+Branco,PR&sensor=false

Já para pesquisas utilizando a longitude e a latitude, o parâmetro latlng deve ser utilizado.

http://maps.googleapis.com/maps/api/geocode/xml?latlng=-26.196223,-52.689523&sensor=false

Adicionando o parâmetro *componets* é possível filtrar os resultados por uma região específica, como o componente é o mesmo para o mundo todo, é interessante filtrar ao menos o país onde deseja-se realizar a pesquisa.

Esta restrição é especificada através do filtro *components*. Um filtro consiste em uma lista contendo “componente:valor” separado com uma barra vertical, o caractere “|”. Desta forma apenas os valores que estiverem dentro dos filtros especificados serão retornados pelo WebService.

http://maps.google.com/maps/api/geocode/json?address=Via+do+conhecimento,km1,+Pato+Branco,PR &**components=country:BR**&sensor=false

No exemplo acima foi adicionado no parâmetro o componente *country* que indica a restrição de país, existem mais componentes possíveis, sendo eles:

* route: Nome longo ou abreviado de um determinado trajeto.
* locality: corresponde aos tipos locality e sublocality.
* administrative\_area: corresponde a todos os níveis administrative\_area.
* postal\_code: corresponde a postal\_code e postal\_code\_prefix.
* country: É o nome de um determinado país ou então a abreviação de duas letras definida no padrão ISSO 3166-1.

**Sensor**

Este parâmetro indica se a solicitação vem ou não com um dispositivo com sensor de localização, o valor informado deve ser simplesmente true ou false. Na dúvida da existência do sensor no seu device, utilize o parâmetro false.

**Parâmetros não obrigatórios para consulta**

Existem alguns parâmetros facultativos que possibilitam a utilização de recursos diferenciados dentro da API, tais como:

* language
* region
* bounds

**Language**

Determina o idioma que os resultados devem ser retornados. Caso o parâmetro não for informado, a API tentará retornar o idioma nativo referente ao local da qual a solicitação foi enviada.

**Region**

Este parâmetro favorecem resultados de um domínio específico. Caso existam resultados relevantes fora do local informado, estes podem ser incluídos neste parâmetro.

http://maps.google.com/maps/api/geocode/json?components=administrative\_area:parana|country:Bra zil&sensor=false&region=BR

**Bounds**

É possível também fazer com que o serviço favoreça resultados que se encontrem em uma determinada janela de visualização. Da mesma forma que o parâmetro *region*, o *bounds* apenas favorece os resultados, ele não impede que outros resultados relevantes sejam retornados.

Este parâmetro define coordenadas de latitude e longitude dos cantos sudoeste e nordeste da janela delimitadora, com a barra vertical “|” separando os valores, conforme exemplificado abaixo.

http://maps.google.com/maps/api/geocode/json?components=administrative\_area:parana|country:Bra zil&sensor=false&bounds=-26.7172983,-54.6192979|-22.5166644,-48.0235303

**Parâmetros para uso em Serviço Empresarial**

Para os usuários que possuírem serviço da API para empresas devem ser incluídos mais dois parâmetros, o *client* e *signature* válidos em sua solicitação de geolocalização. O parâmetro *client* é o ID do cliente e o *signature* é uma assinatura exclusiva composta por uma chave criptográfica válida.

**Desenvolvimento de um estudo de caso**

Para o presente aplicativo, utilizaremos um aplicativo que recupera as posições de latitude e longitude do device Android, utilizando o GPS deste. Estas informações serão apresentadas na tela do device, em dois componentes TextViews. Nesta tela, também haverá um componente Button, que ao ser pressionado acessará o serviço de Geocoding API, convertendo de geocódigos para dado textual, apresentando este na tela para o usuário.

Vamos iniciar um novo projeto Android, este chamado de AutoLocalizacao. Este projeto terá uma única tela, assim, teremos o arquivo activity\_principal.xml e PrincipalActivity.java. O código fonte referente a tela principal é apresentado na **Listagem 3.3.**

**Listagem 3.3.** activity\_principal.xml – Tela principal do aplicativo.

01. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

02. xmlns:tools="http://schemas.android.com/tools"

03. android:layout\_width="match\_parent"

04. android:layout\_height="match\_parent"

05. android:orientation="vertical"

06. tools:context=".PrincipalActivity">

07.

08. <TextView

09. android:layout\_width="wrap\_content"

10. android:layout\_height="wrap\_content"

11. android:text="Latitude:" />

12.

13. <TextView

14. android:id="@+id/tvLatitude"

15. android:layout\_width="wrap\_content"

16. android:layout\_height="wrap\_content"

17. android:textAppearance="?android:attr/textAppearanceLarge"

18. android:text="0.0" />

19.

20. <TextView

21. android:layout\_width="wrap\_content"

22. android:layout\_height="wrap\_content"

23. android:text="Longitude"/>

24.

25.

26. <TextView

27. android:id="@+id/tvLongitude"

28. android:layout\_width="wrap\_content"

29. android:layout\_height="wrap\_content"

30. android:textAppearance="?android:attr/textAppearanceLarge"

31. android:text="0.0" />

32.

33. <TextView

34. android:layout\_width="wrap\_content"

35. android:layout\_height="wrap\_content"

36. android:text="Descrição:" />

37.

38. <TextView

39. android:id="@+id/tvDescricao"

40. android:layout\_width="wrap\_content"

41. android:layout\_height="wrap\_content"

42. android:textAppearance="?android:attr/textAppearanceLarge"

43. android:text="" />

44.

45. <Button

46. android:id="@+id/btBuscarDescricao"

47. android:layout\_width="wrap\_content"

48. android:layout\_height="wrap\_content"

49. android:text="Buscar Descrição Local"

50. android:onClick="btBuscarDescricaoOnClick" />

51.

52. </LinearLayout>

Esta tela é formada por um gerenciador de layout LinearLayout (linha 01), este organizado no formato vertical (linha 5). Os componentes na tela são dispostos aos pares, sendo dois componentes para apresentar o rótulo e a informação referente a latitude (linhas 09 e 13), longitude (linhas 20 e 26), e o campo descrição que apresentará os dados retornados do Geocoding API (linhas 33 e 38). Por fim, o botão que fará acessa a esta API da Google (linha 45).

Observem que todos os componentes visuais de entrada, processamento e saída de dados possuem um nome, este informado na tag android:id. Também, o botão btBuscarDescricao possui um método atribuído a este btBuscarDescricaoOnClick(), o qual deve ser codificado na classe Java.

O passo seguinte é a codificação desta classe, conforme **Listagem 3.4**.

**Listagem 3.4.** PrincipalActivity.java – Classe Java para processamento da tela principal.

01. package br.com.livro.localizacao;

02.

03. import android.content.Context;

04. import android.location.Location;

05. import android.location.LocationListener;

06. import android.location.LocationManager;

07. import android.os.AsyncTask;

08. import android.app.Activity;

09. import android.os.Bundle;

10. import android.view.View;

11. import android.widget.TextView;

12.

13. import org.apache.http.client.HttpClient;

14. import org.apache.http.client.methods.HttpGet;

15. import org.apache.http.impl.client.DefaultHttpClient;

16.

17. import java.io.BufferedReader;

18. import java.io.InputStream;

19. import java.io.InputStreamReader;

20. import java.net.URLEncoder;

21.

22.

23. public class PrincipalActivity extends Activity implements LocationListener {

24.

25. private TextView tvLongitude;

26. private TextView tvLatitude;

27. private TextView tvDescricao;

28.

29. @Override

30. protected void onCreate(Bundle savedInstanceState) {

31. super.onCreate(savedInstanceState);

32. setContentView(R.layout.activity\_principal);

33.

34. tvLongitude = (TextView) findViewById( R.id.tvLongitude );

35. tvLatitude = (TextView) findViewById( R.id.tvLatitude );

36. tvDescricao = (TextView) findViewById( R.id.tvDescricao );

37.

38. LocationManager lm = (LocationManager)

39. this.getSystemService( Context.LOCATION\_SERVICE );

40. lm.requestLocationUpdates( LocationManager.GPS\_PROVIDER, 2000, 0, this );

41.

42. }

43.

44. @Override

45. public void onLocationChanged(Location location) {

46. double latitude = location.getLatitude();

47. double longitude = location.getLongitude();

48.

49. tvLatitude.setText( String.valueOf( latitude ) );

50. tvLongitude.setText( String.valueOf( longitude ) );

51. }

52.

53. @Override

54. public void onStatusChanged(String provider, int status, Bundle extras) {

55.

56. }

57.

58. @Override

59. public void onProviderEnabled(String provider) {

60.

61. }

62.

63. @Override

64. public void onProviderDisabled(String provider) {

65.

66. }

67.

68. public void btBuscarDescricaoOnClick( View v ) {

69. new Conexao().execute( tvLatitude.getText().toString(),

70. tvLongitude.getText().toString() );

71. }

72.

73. class Conexao extends AsyncTask<String, String, String> {

74.

75. @Override

76. protected String doInBackground(String... params) {

77. try {

78. String url = "http://maps.googleapis.com/maps/api/geocode/xml?latlng="

79. + params[0] + "," + params[1]

80. + "&sensor=false";

81. HttpClient httpclient = new DefaultHttpClient();

82. HttpGet request = new HttpGet(url);

83. InputStream in = httpclient.execute(request).getEntity()

84. .getContent();

85.

86. BufferedReader br = null;

87. StringBuilder sb = new StringBuilder();

88.

89. br = new BufferedReader(new InputStreamReader(in));

90.

91. String line = br.readLine();

92.

93. while (line != null) {

94. sb.append(line);

95. line = br.readLine();

96.

97. }

98.

99. return sb.toString();

100.

101. } catch ( Exception e ) {

102. return "Erro: " + e.getMessage();

103. }

104. }

105.

106. @Override

107. protected void onPostExecute(String msg) {

108. tvDescricao.setText( msg );

109. }

110.

111. }

112. }

A classe principal utiliza as primeiras linhas para declaração do pacote e importação das classes utilizadas ao longo do programa (linha 01 a 20). Na linha 23 é declarada a classe principal, assim como implementada a interface LocationListener, utilizada para o uso do GPS.

Da linha 25 a 27 são declarados os componentes visuais do aplicativo, sendo estes recuperados nas linhas 34 a 36. Nas linhas 38 e 39 é recuperado o serviço de GPS, assim como criado um listener para “escutar” mudanças de posição do device Android a cada dois segundos.

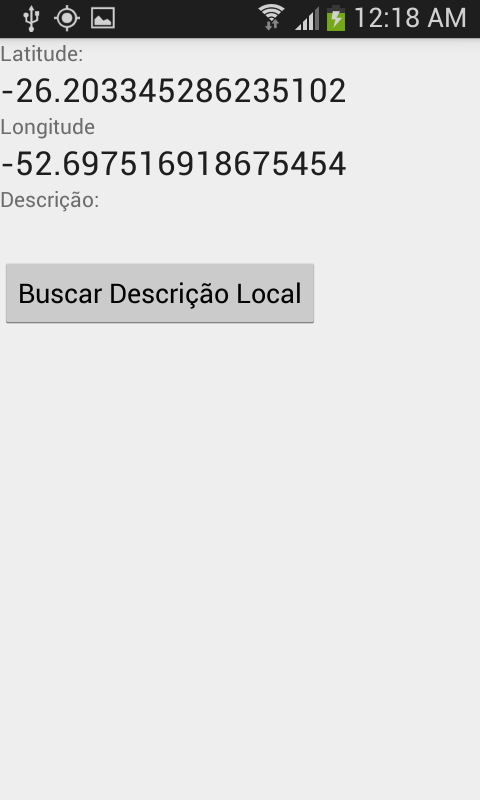
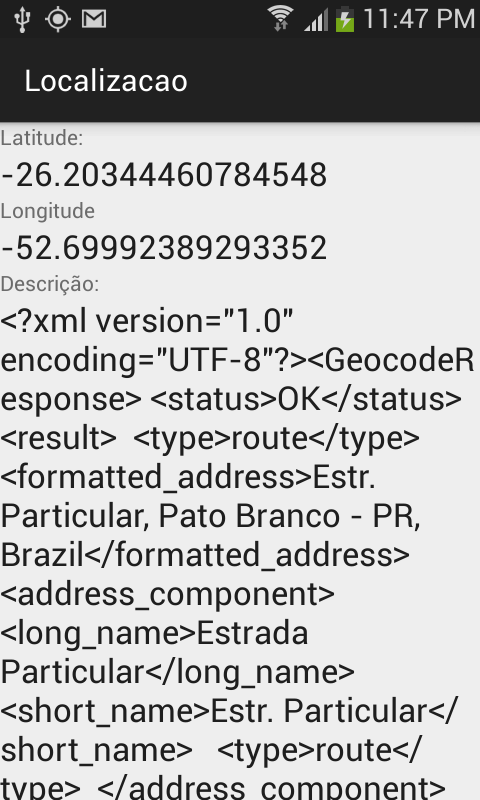
O método onLocationChanged(linha 45) é chamado a cada nova posição do GPS, recuperando a latitude e a longitude (linhas 46 e 47) e apresentado estes dados na tela do device (linhas 49 e 50).

O método btBuscarDescricaoOnClick (linha 68) trata o clique do botão, o qual inicia uma AsyncTask passando por parâmetros os campos latitude e longitude (linha 69 e 70), estes recuperados a partir do GPS.

Na classe interna Conexao (linha 73), no método doInBackground(linha 76), é definida uma url para utilizar a API de Geocoding. Para acesso, é utilizado um objeto HttpClient (linha 81), assim como utilizado o protocolo GET para o acesso (linha 82). Para recuperar o retorno, um objeto InputStream é instanciado (linha 83). Como o retorno é um InputStream, temos que converter este para String, para auxiliar, é utilizado um BufferedReader e um StringBuilder (linhas 86 e 87), o qual recebe o InputStreamReader (linha 89), e cada caractere é adicionado ao objeto sb (linhas 91 a 97). Ao término, o resultado é retornado (linha 99).

A apresentação do resultado na tela acontece no método onPostExecute (linha 107), o qual recupera a resposta processada no método doInBackground() e adiciona este na tela (linha 108).

O resultado deste processamento é apresentado na **Figura 3.2.**

**Figura 3.2.** Resultado do aplicativo em execução.

Verifiquem que o resultado veio no formato XML, devido ao parâmetro utilizado na chamada do serviço. Desta foram, o resultado na integra, sem formatação, é apresentado na tela.

Poderíamos utilizar APIs específicas para a formatação da resposta, mas como nosso objetivo é simplesmente recuperar o conteúdo da tag formatted\_addres, o mesmo pode ser recuperado a partir do método substring, conforme apresentado na **Listagem 3.5**.

**Listagem 3.5.** Método doInBackground() – Processamento de uma informação específica.

01. protected String doInBackground(String... params) {

02. try {

03. String url = "http://maps.googleapis.com/maps/api/geocode/xml?latlng="

04. + params[0] + "," + params[1]

05. + "&sensor=false";

06. HttpClient httpclient = new DefaultHttpClient();

07. HttpGet request = new HttpGet(url);

08. InputStream in = httpclient.execute(request).getEntity()

09. .getContent();

10.

11. BufferedReader br = null;

12. StringBuilder sb = new StringBuilder();

13.

14. br = new BufferedReader(new InputStreamReader(in));

15.

16. String line = br.readLine();

17.

18. while (line != null) {

19. sb.append(line);

20. line = br.readLine();

21.

22. }

23.

**24. String resposta = sb.toString();**

**25.**

**26. return resposta.substring(**

**27. resposta.indexOf( "<formatted\_address>" ) + 19,**

**28. resposta.indexOf( "</formatted\_address>" ) );**

**29.**

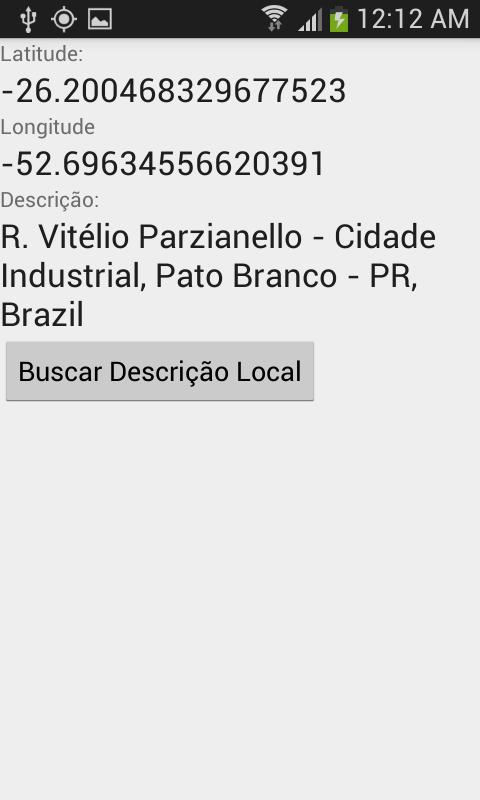
30. } catch ( Exception e ) {

31. return "Erro: " + e.getMessage();

32. }

33. }

Este código tem mudança apenas na parte em negrito, onde o resultado da consulta, um texto XML, é processado, recuperando deste apenas o conteúdo da tag <formatted\_address>, o qual possui o dado textual da posição recuperada, e este é apresentado na tela, no componente tvDescricao. Após a execução, o resultado do programa é apresentada na **Figura 3.3**.



**Figura 3.3.** Resultado do aplicativo em execução com a resposta formatada.

**Concluindo...**

O presente capítulo apresentou o uso do Google Geocoding API, o qual permite conversões de geocódigos (latitude e longitude) em endereço e vice-versa.

O uso desta API traz inúmeras vantagens ao desenvolvedor, em especial, para apresentação de dados geográficos na tela. Muitas vezes, informações numéricas de latitude e longitude não dizem muita coisa para o usuário, desta forma, para a apresentação nada melhor do que apresentar uma informação textual referente ao lugar.

Entretanto, para armazenamento ou transferência de dados ainda é melhor trabalhar com os dados numéricos referentes a longitude e latitude, uma vez que os dados textuais podem não ser muito preciso, e costumam ocupar muito espaço se comparado aos dados numéricos.