



MANUAL DE TECNICO PROYECTO CRIPTOMONEDAS

Contenido

Introducción	2
API	3
Android Manifest	7
Build.gradle(:app).....	8
Strings	9
Colors	9
Conexión a API	10
Monedas	11
Activity main	13
Main Activity	15

Introducción

El propósito de la aplicación Criptomonedas es lograr la monitorización de las criptomonedas que se encuentran en el mercado, esto es posible gracias a que el proyecto está conectado a una API llamada Coingecko. La idea de este proyecto es que pueda tener cambios en un futuro, por ende, la intención de este manual es explicar cómo funciona la aplicación en su backend.

API

La API es coingecko, esta API nos permite configurar las url para extraer datos que nosotros deseemos o que esté en nuestras necesidades del proyecto.

La api coingecko se puede obtener de: <https://www.coingecko.com/en/api/documentation>.

Para obtener los datos que se encuentran en la aplicación, a la fecha de realización de este documento, fue necesario ir al apartado coins y seleccionar la casilla de /coins/markets.

coins		^
GET	/coins/list	List all supported coins id, name and symbol (no pagination required) ✓
GET	/coins/markets	List all supported coins price, market cap, volume, and market related data ✓
GET	/coins/{id}	Get current data (name, price, market, ... including exchange tickers) for a coin ✓
GET	/coins/{id}/tickers	Get coin tickers (paginated to 100 items) ✓
GET	/coins/{id}/history	Get historical data (name, price, market, stats) at a given date for a coin ✓
GET	/coins/{id}/market_chart	Get historical market data include price, market cap, and 24h volume (granularity auto) ✓
GET	/coins/{id}/market_chart/range	Get historical market data include price, market cap, and 24h volume within a range of timestamp (granularity auto) ✓
GET	/coins/{id}/ohlcv	Get coin's OHLC ✓

En el apartado coins/market se le da click en el botón Try Out

GET

/coins/markets

List all supported coins price, market cap, volume, and market related data

^

Use this to obtain all the coins market data (price, market cap, volume)

Parameters

Try it out

Name	Description
vs_currency * required string (query)	The target currency of market data (usd, eur, jpy, etc.) <input type="text" value="vs_currency"/>
ids string (query)	The ids of the coin, comma separated cryptocurrency symbols (base). refers to /coins/list . When left empty, returns numbers the coins observing the params limit and start <input type="text" value="ids"/>
category string (query)	filter by coin category. Refer to /coin/categories/list <input type="text" value="category"/>
order string (query)	valid values: market_cap_desc, gecko_desc, gecko_asc, market_cap_asc, market_cap_desc, volume_asc, volume_desc, id_asc, id_desc sort results by field. Default value : market_cap_desc <input type="text" value="market_cap_desc"/>
per_page integer (query)	valid values: 1..250 Total results per page Default value : 100 <input type="text" value="100"/>
page integer (query)	Page through results Default value : 1 <input type="text" value="1"/>
sparkline boolean (query)	Include sparkline 7 days data (eg. true, false) Default value : false <input type="text" value="false"/>
price_change_percentage string (query)	Include price change percentage in 1h, 24h, 7d, 14d, 30d, 200d, 1y (eg. 1h,24h,7d ' comma-separated, invalid values will be discarded) <input type="text" value="price_change_percentage"/>

Responses

Response content type

application/json

En `vs_currency` se pone el tipo de moneda física que se desee que sean mostrados los precios de las criptomonedas (dólares, euros, etc.). En nuestro caso, se pone `mxn` para pesos mexicanos. En caso de desear cambiar el tipo de moneda, solo se escribe la que se desea.

vs_currency * required	The target currency of market data (usd, eur, jpy, etc.)
string (query)	<input type="text" value="vs_currency"/>

En `per_page` es la cantidad de criptomonedas que deseamos mostrar, Coingecko cuenta con 100 criptomonedas, por ende, si se desean las 100, ponemos el número 100 que se encuentra por default.

per_page	valid values: 1..250
integer (query)	Total results per page
	<input type="text" value="100"/>

Los datos restantes son opcionales. Cuando ingresamos los datos, ejecutamos la URL en el botón azul "Execute".

sparkline

boolean

(query)

Include sparkline 7 days data (eg. true, false)

false

price_change_percentage

string

(query)

Include price change percentage in 1h, 24h, 7d, 14d, 30d, 200d, 1y (eg. '1h,24h,7d' comma-separated, invalid values will be discarded)

price_change_percentage

Execute

Responses

Response content type

application/json

Code	Description
200	List all coins with market data

Al haber ejecutado la URL se nos regresará, lo que se debe tomar en cuenta es el request URL (url en letras blancas).

Responses

Response content type

application/json

Curl


```
curl -X 'GET' \
'https://api.coingecko.com/api/v3/coins/markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false' \
-H 'accept: application/json'
```

Request URL

```
https://api.coingecko.com/api/v3/coins/markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false
```

En la parte de abajo veremos cómo es el Json que nos arrojó la API, aquí podemos ver los datos que se nos arrojarán en pantalla.

Server response

Code	Details
200	<div>Response body</div> <pre>[{ "id": "bitcoin", "symbol": "btc", "name": "Bitcoin", "image": "https://assets.coingecko.com/coins/images/1/large/bitcoin.png?1547033579", "current_price": 819109, "market_cap": 15600978644274, "market_cap_rank": 1, "fully_diluted_valuation": 17220602555540, "total_volume": 627042404056, "high_24h": 825565, "low_24h": 791416, "price_change_24h": 27680, "price_change_percentage_24h": 3.49748, "market_cap_change_24h": 569156181294, "market_cap_change_percentage_24h": 3.78634, "circulating_supply": 19024918, "total_supply": 21000000, "max_supply": 21000000, "ath": 1489247, "ath_change_percentage": -41.81087, "ath_date": "2021-11-10T17:30:22.767Z",</pre> <div> Download</div>

El proyecto ha sido desarrollado en Android Studio y es compatible con dispositivos Android 5.1 en adelante.

Android Manifest

Los cambios realizados en el Android Manifest fueron el añadido de los permisos de acceso a internet a la app, sin ellos, el proyecto no funcionará. Los permisos se encuentran en las líneas 5 y 6.

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


Build.gradle(:app)

Aquí se añadió retrofit y retrofit converter, retrofit nos permite conectar la api y retrofit converter nos permite navegar a través de los archivos JSON. Se añadieron en el apartado dependencies en la línea 39 y 40.

```
30 dependencies {
31
32     implementation 'androidx.appcompat:appcompat:1.4.1'
33     implementation 'com.google.android.material:material:1.5.0'
34     implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
35     testImplementation 'junit:junit:4.13.2'
36     androidTestImplementation 'androidx.test.ext:junit:1.1.3'
37     androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
38     //IMPLEMENTACION DE RETROFIT
39     implementation "com.squareup.retrofit2:retrofit:2.9.0"
40     implementation "com.squareup.retrofit2:converter-gson:2.9.0"
41     //CARDVIEW Y RECYCLERVIEW
42     implementation 'androidx.recyclerview:recyclerview:1.2.0'
43     implementation 'androidx.cardview:cardview:1.0.0'
44 }
```

Strings

Solo se cambió el nombre de la app y se creó un String Array de criptomonedas, este no se le asigna una función aún, pero tenía como intención, mostrar solo las monedas que se encuentren dentro de este string.

```
<resources>
  <string name="app_name">CRIPTOMONEDAS</string>

  <string-array name="cripto">
    <item>Bitcoin</item>
    <item>Ethereum</item>
    <item>NEAR Protocol</item>
    <item>Dogecoin</item>
    <item>Shiba Inu</item>
  </string-array>
</resources>
```

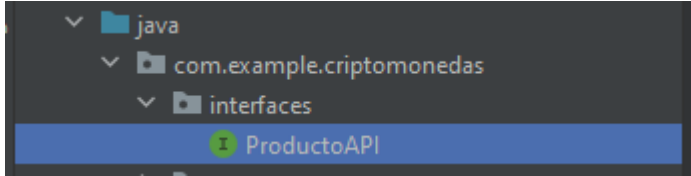
Colors

Solo se cambiaron los colores de la barra.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3    <color name="purple_200">#FFBB86FC</color>
4
5    <color name="purple_500">#FF000000</color>
6    <color name="purple_700">#75706F</color>
7    <color name="teal_200">#FF03DAC5</color>
8    <color name="teal_700">#FF018786</color>
9    <color name="black">#FF000000</color>
10   <color name="white">#FFFFFFFF</color>
11 </resources>
```

Conexión a API

Se creó una interfaz llamada ProductoAPI donde se realizará la búsqueda de la API.



Dentro de la interfaz encontraremos la importación de retrofit y retrofitGET, es importante no borrarlos debido a que nos permiten conectarnos a la API online.

```
1 package com.example.criptomonedas.interfaces;
2
3 import com.example.criptomonedas.models.Monedas;
4
5 import java.util.List;
6
7 import retrofit2.Call;
8 import retrofit2.http.GET;
9
10 public interface ProductoAPI {
11
12
13     @GET("markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false&price_change_percentage=1h")
14     Call<List<Monedas>> getCripto();
15 }
16
```

En el apartado GET se puso entre paréntesis el apartado de donde se extraen los datos de la API, **solo colocaremos la parte final de la URL** que generamos en Coingecko.

```
@GET("markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false&price_change_percentage=1h")
Call<List<Monedas>> getCripto();
}
```

Por ejemplo: nuestra URL completa es https://api.coingecko.com/api/v3/coins/markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false. Solo tomaremos la parte en rojo debido a que lo demás será la URL base y eso debe ponerse en otro apartado. Si no lo hacemos, el proyecto hará crash al momento de ejecutarlo.

Debido a que el JSON de Coingecko es en lista, creamos una lista que extrae datos de una clase llamada Monedas.

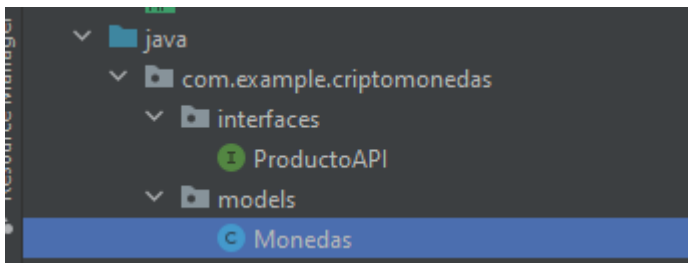
Si deseas reducir la cantidad de criptomonedas en pantalla, cambia el número que está en rojo en esta URL. Recuerda que tiene como limite de 100. Lo mismo ocurre con la moneda internacional solo cámbiala por el id de la moneda deseada.

https://api.coingecko.com/api/v3/coins/markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false

```
@GET("markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false")  
Call<List<Monedas>> getCripto();
```

Monedas

En la carpeta Models hay una clase llamada Monedas



Dentro de la clase están los datos que queremos extraer de la API, **los datos que queremos extraer deben llamarse exactamente igual que como se encuentran en el JSON**. Para saber cómo se llaman los datos podemos verlos en el resultado que nos dio Coingecko al ejecutar la URL o podemos tomar la URL completa y pegarla en nuestro navegador, se recomienda tener una extensión que acomode los JSON para una mejor visualización.

```

1 // 20230420143330
2 // https://api.coingecko.com/api/v3/coins/markets?vs_currency=us&order=market_cap_desc&per_page=100&page=1&sparkline=false.
3
4 {
5   {
6     "id": "bitcoin",
7     "symbol": "btc",
8     "name": "Bitcoin",
9     "image": "https://assets.coingecko.com/coins/images/1/large/bitcoin.png?1547033571",
10    "current_price": 810068,
11    "market_cap": 1551956158303,
12    "market_cap_rank": 1,
13    "fully_diluted_valuation": 17130472395873,
14    "total_volume": 632750915196,
15    "high_24h": 825505,
16    "low_24h": 797139,
17    "price_change_24h": 19179.55,
18    "price_change_percentage_24h": 2.40070,
19    "market_cap_change_24h": 314023687908,
20    "market_cap_change_percentage_24h": 2.05322,
21    "circulating_supply": 19024968.0,
22    "total_supply": 21000000.0,
23    "max_supply": 21000000.0,
24    "ath": 140247,
25    "ath_change_percentage": -42.16001,
26    "ath_date": "2021-11-10T17:30:22.767Z",
27    "atl": 89.32,
28    "atl_change_percentage": 94755.21634,
29    "atl_date": "2013-07-05T00:00:00.000Z",
30    "roi": null,
31    "last_updated": "2023-04-20T20:32:18.306Z"
32  },
33  {
34    "id": "ethereum",
35    "symbol": "eth",
36    "name": "Ethereum",
37    "image": "https://assets.coingecko.com/coins/images/279/large/ethereum.png?1595368880",
38    "current_price": 60308,
39    "market_cap": 725788508399,
40    "market_cap_rank": 2,
41    "fully_diluted_valuation": null,
42    "total_volume": 37027168990,
43    "high_24h": 60950,
44    "low_24h": 58416,
45    "price_change_24h": 1758.54,
46    "price_change_percentage_24h": 3.00331,
47    "market_cap_change_24h": 18064166752,
48    "market_cap_change_percentage_24h": 2.66838,
49    "circulating_supply": 120566629.249,
50    "total_supply": null,
51    "max_supply": null,
52    "ath": 4864.76,
53    "ath_change_percentage": -98.7456,
54    "ath_date": "2021-12-03T06:30:00.000Z",
55    "atl": 0.41815,
56    "atl_change_percentage": 14417.4273,
57    "atl_date": "2015-07-30T00:00:00.000Z",
58    "roi": {
59      "times": 14417.4273,
60      "currency": "USD",
61      "from": 2015-07-30T00:00:00.000Z
62    },
63    "last_updated": "2023-04-20T20:32:18.306Z"
64  }
65 }

```

Los datos que se toman de momento son name, current_price y symbol. Los métodos getters y setters deben mantenerse para poder interactuar con los datos. En caso de querer añadir más datos de las criptomonedas, lo único que se debe hacer es crear los strings con los nombres de los datos y crear sus getters y setters.

```

1 package com.example.criptomonedas.models;
2
3 public class Monedas {
4
5     private String name;
6     private String current_price;
7     private String market_cap;
8     private String market_cap_rank;
9     private String symbol;
10
11     public String getMarket_cap_rank() { return market_cap_rank; }
12
13     public void setMarket_cap_rank(String market_cap_rank) {
14         this.market_cap_rank = market_cap_rank;
15     }
16
17     public String getMarket_cap() { return market_cap; }
18
19     public void setMarket_cap(String market_cap) { this.market_cap = market_cap; }
20
21     public String getCurrent_price() { return current_price; }
22
23     public void setCurrent_price(String current_price) { this.current_price = current_price; }
24
25     public String getName() { return name; }
26
27     public void setName(String name) { this.name = name; }
28
29     public String getSymbol() { return symbol; }
30
31     public void setSymbol(String symbol) { this.symbol = symbol; }
32 }

```

Activity main

El activity main solo tiene el constraint layout, dentro de él un Linear Layout con un TextView, este solo tiene formato, no realiza ninguna tarea.

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/black"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="5dp"
        android:background="@color/black"
        android:orientation="vertical"
        tools:layout_editor_absoluteX="5dp"
        tools:layout_editor_absoluteY="5dp">

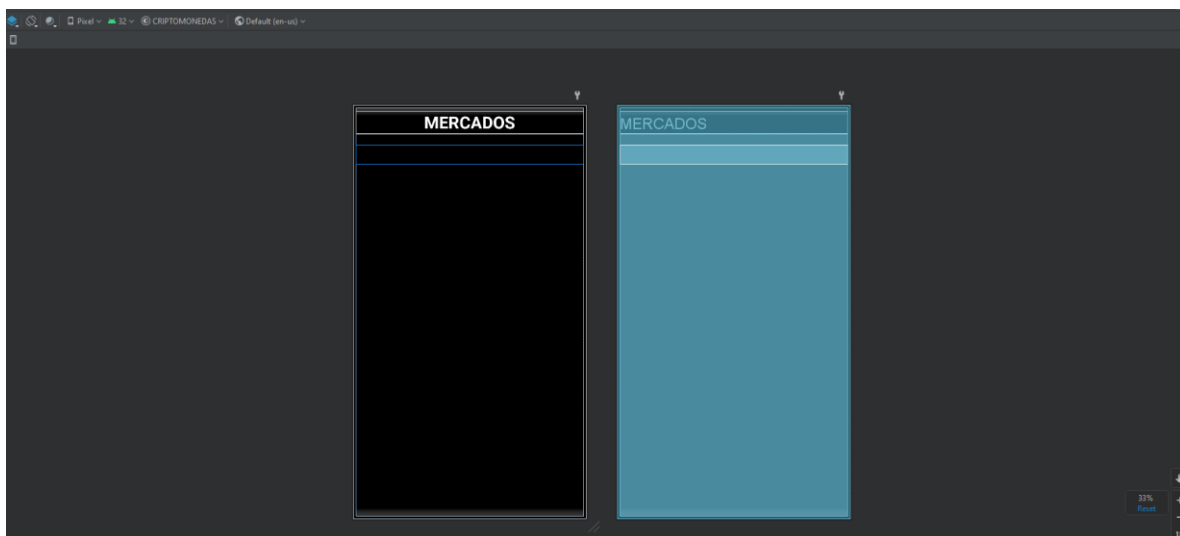
        <TextView
            android:id="@+id/tvTitulo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="5dp"
            android:gravity="center"
            android:textColor="@android:color/white"
            android:text="MERCADOS"
            android:textSize="30sp"
            android:textStyle="bold"
            android:background="@color/black"/>
```

Debajo hay un NestedScrollView, este se añadió debido a que es necesario desplazar la pantalla para visualizar las criptomonedas. Dentro del ScrollView hay un TextView que no se ve en el apartado de diseño, pero en este se mostrarán las criptomonedas. Su ID es **tvCurrentBitcoin**.

```

<androidx.core.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@android:color/black">
    <TextView
        android:id="@+id/tvCurrentBitcoin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text=""
        android:textColor="@android:color/darker_gray"
        android:textSize="25sp"
        android:textStyle="normal" />
    </androidx.core.widget.NestedScrollView>
</LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>

```



Main Activity

Aquí se tienen todas las importaciones de todas las herramientas que son necesarias para que la app funcione correctamente. Es importante no borrar ninguna.

```
package com.example.criptomonedas;

import androidx.appcompat.app.AppCompatActivity;

import android.content.res.Resources;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

import com.example.criptomonedas.interfaces.ProductoAPI;
import com.example.criptomonedas.models.Monedas;

import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
```


Se asignó la variable al Textview y aquí se tiene el método **setMonedas** para obtener todas las monedas

```
21 public class MainActivity extends AppCompatActivity {
22
23
24     TextView Currentbtc;
25
26     @Override
27     protected void onCreate(Bundle savedInstanceState) {
28         super.onCreate(savedInstanceState);
29         setContentView(R.layout.activity_main);
30         Currentbtc = findViewById(R.id.tvCurrentBitcoin);
31
32
33         setMonedas();
34
35     }
36
```

En `setMonedas` se implementa la url base del api, entonces, si tenemos https://api.coingecko.com/api/v3/coins/markets?vs_currency=mxn&order=market_cap_desc&per_page=100&page=1&sparkline=false aquí se pone solamente la parte en rojo de la url, ya que esa es la base de donde se buscarán los datos y se vincula la interfaz **ProductoAPI** con este apartado.

También se llama **getCripto** con la lista de monedas.

```
7     public void setMonedas() {
8         Retrofit retrofit = new Retrofit.Builder()
9             .baseUrl("https://api.coingecko.com/api/v3/coins/")
10            .addConverterFactory(GsonConverterFactory.create())
11            .build();
12
13        ProductoAPI productoAPI = retrofit.create(ProductoAPI.class);
14
15        Call<List<Monedas>> call = productoAPI.getCripto();
16    }
17
```

Se manda a llamar la lista para obtener los objetos de la Clase Monedas, en el public void se crea el onResponse y en caso de un fallo en el sistema el response.isSuccessful se mandará el código de error y se imprimirá en pantalla para permitir saber la causa del error (response.code()).

```
48      call.enqueue(new Callback<List<Monedas>>() {
49          @Override
50          public void onResponse(Call<List<Monedas>> call, Response<List<Monedas>> response) {
51
52              if(!response.isSuccessful()){
53                  Currentbtc.setText("CODIGO " +response.code());
54                  return;
55              }
```

Se creó un for donde se realizará la obtención de los datos, se vincula la clase monedas y después listMonedas, dentro del bucle se tiene un string. El string obtendrá los datos que se desean imprimir, al terminar de imprimir una criptomoneda, volverá al proceso hasta imprimir todas las monedas que se encuentren en la URL.

```
57      List<Monedas> listMonedas = response.body();
58
59      for(Monedas moneda: listMonedas){
60
61
62
63          String content = "";
64          content += moneda.getSymbol() + "\n";
65          content += "          $" + moneda.getCurrent_price() + "mxn" + "\n";
66          content += moneda.getName() + "\n" + "\n" + "\n" + "\n";
67          Currentbtc.append(content);
68
69      }
```

En caso de un fallo de conexión a internet se imprime un Toast

```

    @Override
    public void onFailure(Call<List<Monedas>> call, Throwable t) {
        Currentbtc.setText(t.getMessage());
        Toast.makeText( context MainActivity.this, text: "ERROR EN SISTEMA", Toast.LENGTH_SHORT).show();
    }
};
```