

código.py

GALÁXIA 7

Apis no Python



Introdução

Olá, seja bem-vindo à Galáxia 7 de APIs no Python! Aprenderemos sobre as APIs, que é a forma que o Python utiliza como comunicação entre sites, banco de dados, softwares, etc. Aprenderemos também sobre as principais APIs do mercado financeiro e como você poderá utilizá-las para melhorar suas análises.



Mundo 1

1.1. O que é uma API?

Antes de entender o que é uma API, você precisa ter a noção que ela é o pilar central dos serviços da internet. É por meio de APIs que estes serviços conseguem se comunicar.

API nada mais é do que um mediador entre a interface e a programação. Imagine que você tenha um banco de dados com informações que precisam ser disponibilizadas para terceiros: é por meio de APIs que você conseguirá compartilhar informações de forma automática, simples e segura.

Bote na sua cabeça que sempre que você precisar acessar um sistema para coletar informações, seja pela B3, Facebook, IBGE ou até mesmo a base de dados do nosso curso, terá que ser por meio de APIs. E como nada é fácil nesta vida, cada API tem seu método de consulta, sua própria documentação, suas regras, etc. mas, no fundo, todas funcionam da mesma maneira, e é isto que vou te mostrar nessa Galáxia.

Um exemplo prático de como as APIs são utilizadas no dia-a-dia, é o login. Quando você deseja fazer um Login em alguma plataforma você precisa seu usuário e senha, e esta plataforma retornará uma resposta, como "senha inválida", para você. O sistema da empresa fará toda verificação para permitir, ou não, sua entrada. Você não precisa saber como este sistema funciona, mas precisa ter noção que sempre que você envia uma informação, são retornadas respostas mesmo que você não as veja.

Existem plataformas que disponibilizam APIs prontas que fazem toda integração com sua loja virtual, por exemplo. Quem disponibiliza a API pode definir: quais, como e quantas informações podem chegar para o servidor.

1.2. Como acessar uma API?

Existem várias formas de acessar uma API e existem vários tipos de APIs também. Neste caso, vou me abster apenas das API's REST que são as que se comunicam com a internet.

As APIs REST são acessadas por endpoints que fazem parte das URLs, conhecidas como Links. Quando você acessa um link, você está acessando a parte final de um sistema. Então imagine que você quer acessar dados da inflação do banco central:

URL =

ENDPOINT =

Se desejo acessar o índice de preços:

URL =

ENDPOINT =

URLs parecidas, mas os endpoints diferem e definem qual parte do site você quer acessar. Para trabalhar com APIs REST é a mesma forma. Aliás, os sites são divididos por APIs... Mas isso é papo para mais tarde.

1.3. Como os dados chegam?

Existem dois principais formatos utilizados, os quais são o Json e XML, que possuem a seguinte formatação:

XML

vs.

JSON

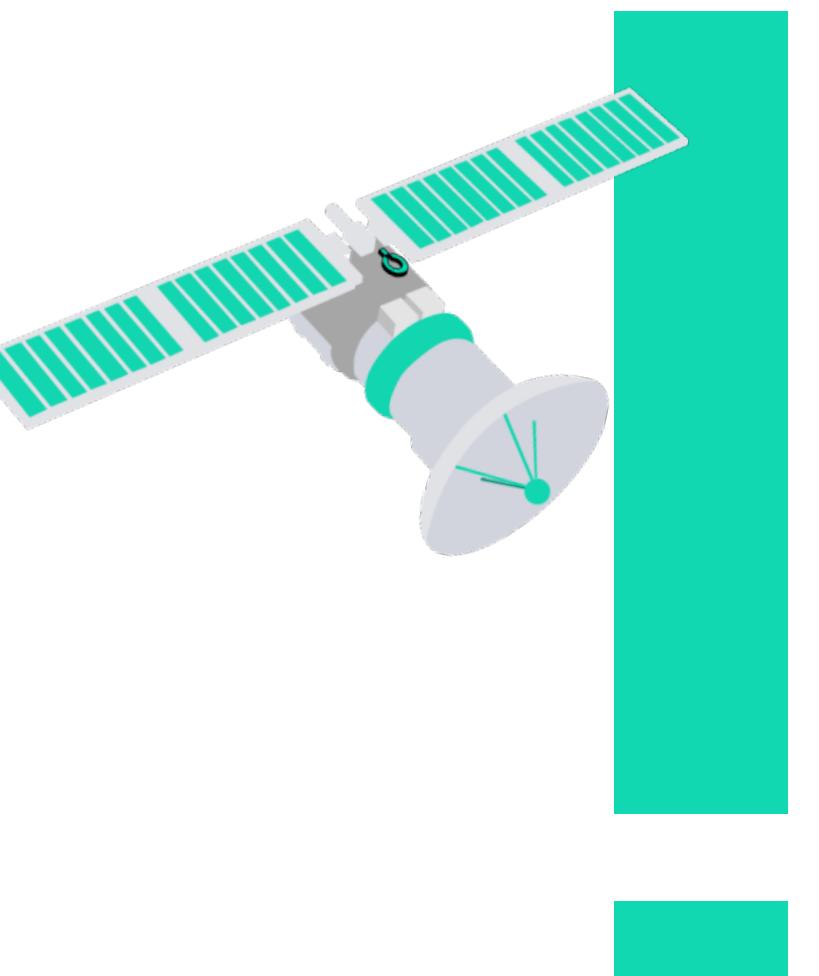
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <endereco>
3   <cep>31270901</cep>
4   <city>Belo Horizonte</city>
5   <neighborhood>Pampulha</neighborhood>
6   <service>correios</service>
7   <state>MG</state>
8   <street>Av. Presidente Antônio Carlos, 6627</street>
9 </endereco>
```

```
1 {
2   "endereco": {
3     "cep": "31270901",
4     "city": "Belo Horizonte",
5     "neighborhood": "Pampulha",
6     "service": "correios",
7     "state": "MG",
8     "street": "Av. Presidente Antônio Carlos, 6627"
9   }
10 }
```

O formato Json é mais comum de ser utilizado e tem sua estrutura igual ao famoso Dicionário dentro do Python.

1.3.1. Como funciona a segurança?

Como dito anteriormente, a instituição que configurou a API é quem escolhe quem pode, ou não, acessá-la. E eles fazem isso por meio de chaves de autenticação. Essa chave é sua identificação e deve ser guardada de uma forma segura que ninguém mais tenha acesso a ela.



Mundo 2

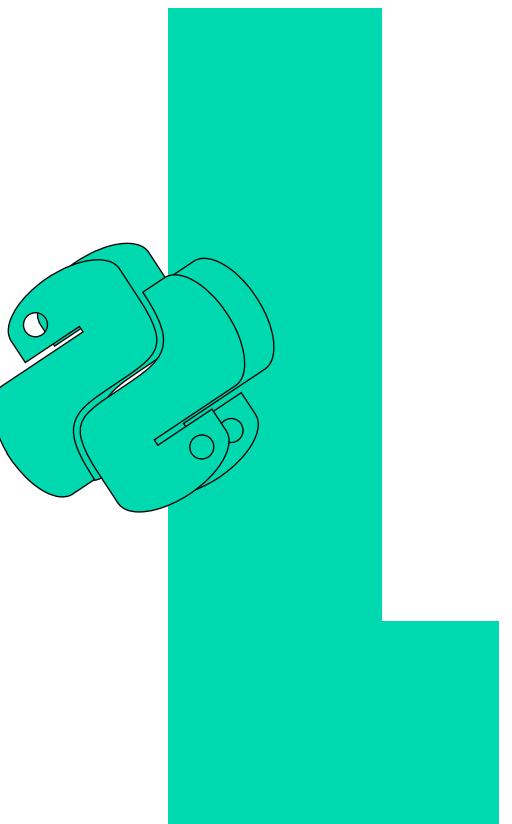
2.1. Entendendo requisições

Como dito anteriormente, a internet funciona por meio de APIs compostas em sua grande maioria por APIs REST.

Na composição de uma API REST, existem os protocolos que informam para API como esta funcionará, entre os protocolos podem se destacar os seguintes:

GET:

É por meio deste protocolo que as informações são obtidas. Como dito anteriormente, as partes que compõem um site são divididas em APIs, cada janela de um site é um endpoint diferente. Sempre que você acessa um site, você está fazendo uma requisição para o protocolo GET. Você enviou requisições GET toda sua vida sem saber disso.



POST:

Este protocolo é o inverso do GET, e assim como o GET, você enviou requisições POST a sua vida inteira sem saber. Sempre que você precisar inserir alguma informação, seja uma senha ou um formulário, o método POST estará por trás. O mais comum é o Login.

PUT:

Este protocolo é utilizado para atualizar informações já existentes em um banco de dados.

DELETE:

Este protocolo é utilizado para deletar itens ou informações de um software.

Mundo 3

3.1. Resposta de um servidor

Sempre que você envia uma requisição, seja de GET, POST, PUT ou DELETE o servidor envia de volta um código. Através desse código você poderá ter uma noção se sua requisição foi aceita ou não e, por que não.

Em resumo, estas são as respostas possíveis que um servidor pode te retornar:

Respostas de informação (100-199):

Esses códigos são usados para fornecer informações sobre o processamento da solicitação. Alguns exemplos incluem:

- 100 Continue: Indica que o servidor recebeu a parte inicial da solicitação e aguarda que o cliente envie o restante.

- 101 Switching Protocols: Indica que o servidor está mudando para um protocolo diferente conforme solicitado pelo cliente.

- 102 Processing: Indica que o servidor está processando a solicitação, mas ainda não concluiu o processamento.

Respostas de sucesso (200-299):

Esses códigos indicam que a solicitação foi recebida e processada com sucesso pelo servidor. Alguns exemplos incluem:

- 200 OK: Indica que a solicitação foi bem-sucedida e o servidor retornou os dados solicitados.
- 201 Created: Indica que a solicitação foi bem-sucedida e resultou na criação de um novo recurso.
- 202 Accepted: Indica que a solicitação foi aceita para processamento, mas ainda não foi concluída.

- 204 No Content: Indica que a solicitação foi bem-sucedida, mas não há conteúdo para retornar.

Redirecionamentos (300-399):

Esses códigos indicam que a solicitação requer ações adicionais para ser concluída. Alguns exemplos incluem:

- 300 Multiple Choices: Indica que a solicitação tem várias opções disponíveis e o cliente deve escolher uma delas.
- 301 Moved Permanently: Indica que o recurso solicitado foi permanentemente movido para uma nova localização.
- 302 Found: Indica que o recurso solicitado foi temporariamente alterado para uma nova localização.
- 304 Not Modified: Indica que a versão em cache do recurso ainda é válida e não precisa ser baixada novamente.



Erros do cliente (400-499):

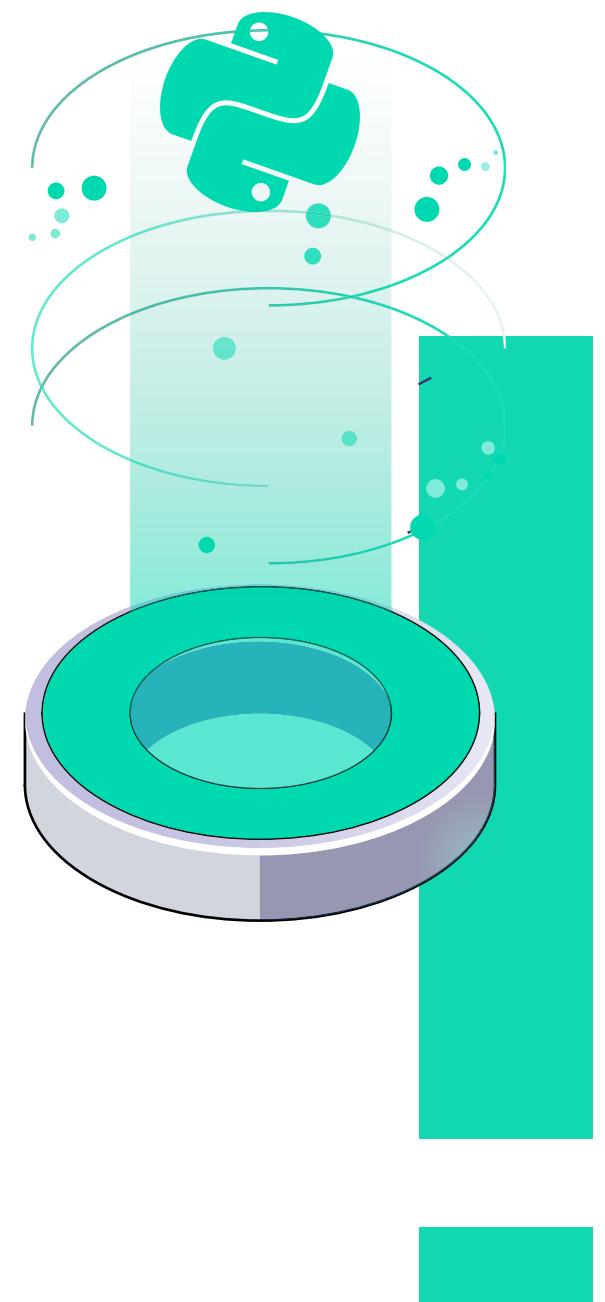
Esses códigos indicam que ocorreu um erro na solicitação feita pelo cliente. Alguns exemplos incluem:

- 400 Bad Request: Indica que a solicitação do cliente é inválida ou malformada.
- 401 Unauthorized: Indica que o cliente não está autorizado a acessar o recurso solicitado.
- 403 Forbidden: Indica que o servidor entende a solicitação, mas se recusa a atendê-la.
- 404 Not Found: Indica que o recurso solicitado não foi encontrado no servidor.

Erros do servidor (500-599):

Esses códigos indicam que ocorreu um erro no servidor durante o processamento da solicitação. Alguns exemplos incluem:

- 500 Internal Server Error: Indica um erro interno no servidor que impede o processamento da solicitação.
- 501 Not Implemented: Indica que o servidor não suporta a funcionalidade necessária para atender à solicitação.
- 502 Bad Gateway: Indica que o servidor atuando como gateway ou proxy recebeu uma resposta inválida do servidor upstream.
- 503 Service Unavailable: Indica que o servidor não está disponível no momento devido à sobrecarga ou manutenção.



3.2. nMétodo GET - Fazendo requisições, na prática

Neste caso estamos efetuando uma requisição GET ao servidor do Google. Observe que o prefixo "https://" é obrigatório.

```
import requests

info_google = requests.get("https://www.google.com.br/")

print(info_google)
```

```
>>> <Response [200]>
```

Como já dito anteriormente, o Python é uma linguagem de programação orientada a objetos. Neste caso, o "info_google" virou um objeto, e como sabemos, os objetos possuem atributos próprios.

3.2.1. Atributos do objeto obtido pelo request

Se você digitar um “.” depois do objeto e apertar os botões Ctrl + Barra-de-espaco você consegue ver todos os atributos que este objeto possui.

```
import requests
info_google = requests.get("https://www.google.com.br/")
print(info_google)
    □ raise_for_status
    □ raw
    □ reason
    □ request
    □ status_code
    □ text
    □ url
    □ _content
    □ _annotations_
    □ _attrs_
    □ _bool_
    □ _class_
```

Destes atributos, os que se destacam nesta biblioteca são:

Text: retorna o html presente dentro daquela requisição

```
{
var h=this||self;function l(){return void 0!==window.google&&void 0!==window.google.kOPI&&0
==window.google.kOPI?window.google.kOPI:null} ;var m,n=[];function p(a){for(var b;a&&(!a.getAttribute
||!(b=a.getAttribute("eid")));)a=a.parentNode;return b||m}function q(a){for(var b=null;a&&
(!a.getAttribute||!(b=a.getAttribute("leid")));)a=a.parentNode;return b}function r(a){/^http:/i.test(a)&&"https:"
==window.location.protocol&&(google.ml&&google.ml(Error("a"),!1,{src:a,gLMM:1}),a="");return a}
```

Json: retorna o json presente dentro daquela requisição

No caso do Google não funcionará, pois o Google não disponibiliza um Json, porém é bom que você saiba que esse método existe e é muito utilizado. Este método retornará um dicionário

Headers: retorna o cabeçalho presente dentro daquela requisição

```
{"Date": "Tue, 16 May 2023 13:00:36 GMT", "Expires": "-1", "Cache-Control": "private, max-age=0", "Content-Type": "text/html; charset=ISO-8859-1", "Content-Security-Policy-Report-Only": "object-src 'none';base-uri 'self';script-src 'nonce-dv8sHVPjvb0zuucsWycSpA' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: http;;report-uri https://csp.withgoogle.com/csp/gws/other-hp"}
```

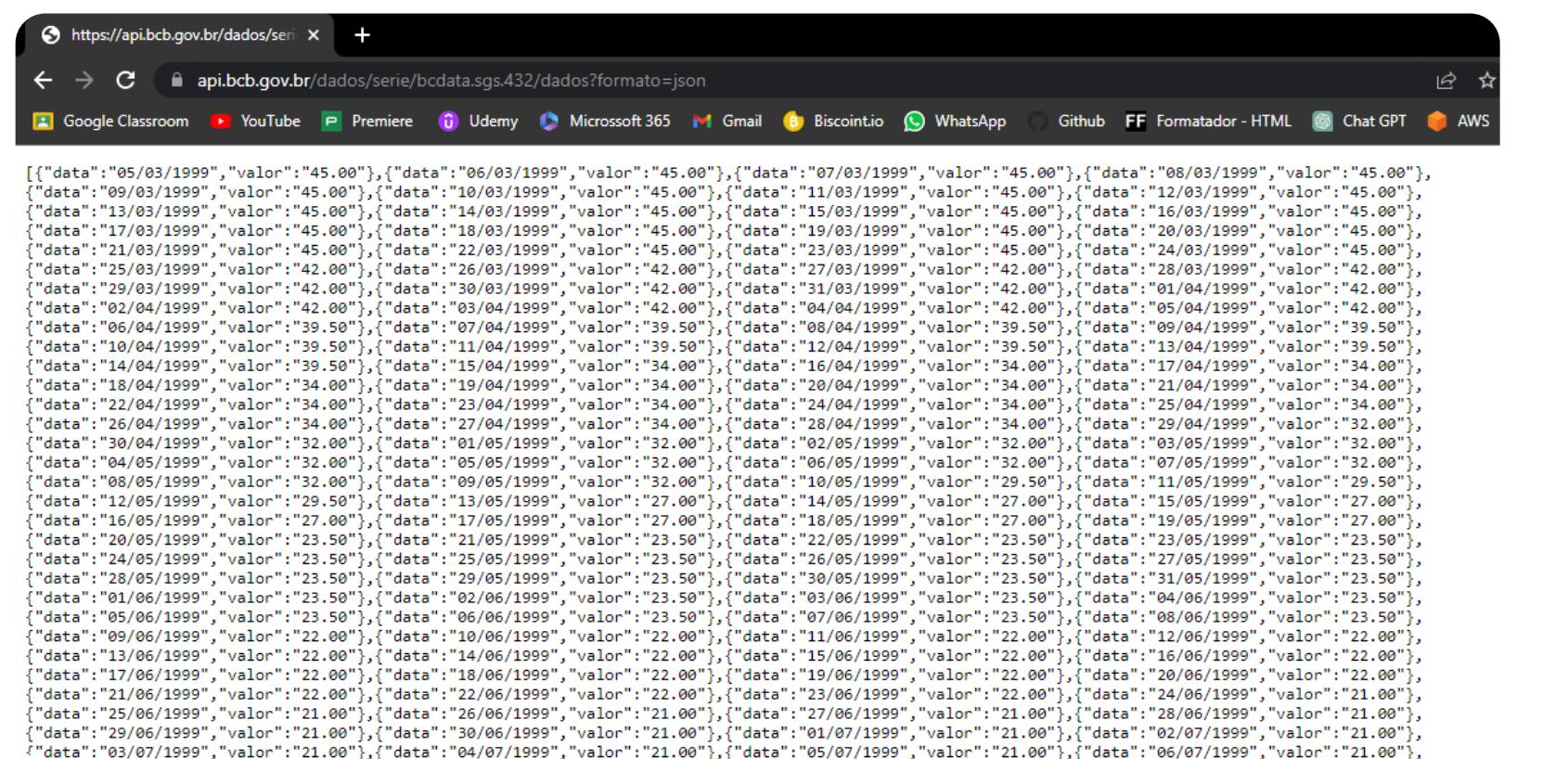
3.3. Fazendo requisição para o Banco Central

Faremos requisição para o banco central através deste site:

<https://api.bcb.gov.br/dados/serie/bcdata.sgs.432/dados?formato=json>

Se você clicar neste site será redirecionado para a seguinte página

abaixo:



É importante que você note que, por estarmos trabalhando com todos HTTPS, que é o método utilizado pela internet, a gente consegue acessar os sites e obter as informações. Previsível né? Já que o Python não cria nada, apenas retira as informações que já estão no site.

3.3.1. Fazendo requisição no Python

```
import requests
import pandas as pd

codigo = 432
url_banco_central = f"https://api.bcb.gov.br/dados/serie/bcdata.sgs.{codigo}/dados?formato=json"

response = requests.get(url_banco_central)
json_selic = response.json()

print(json_selic)
df_selic = pd.DataFrame(json_selic)
print(df_selic)
```

```
>> Print json_selic
```

```
[{'data': '05/03/1999', 'valor': '45.00'},  
 {'data': '06/03/1999', 'valor': '45.00'},  
 {'data': '07/03/1999', 'valor': '45.00'},  
 {'data': '08/03/1999', 'valor': '45.00'},  
 {'data': '09/03/1999', 'valor': '45.00'},  
 {'data': '10/03/1999', 'valor': '45.00'},  
 {'data': '11/03/1999', 'valor': '45.00'},  
 {'data': '12/03/1999', 'valor': '45.00'},  
 {'data': '13/03/1999', 'valor': '45.00'},  
 {'data': '14/03/1999', 'valor': '45.00'},  
 {'data': '15/03/1999', 'valor': '45.00'},  
 {'data': '16/03/1999', 'valor': '45.00'},  
 {'data': '17/03/1999', 'valor': '45.00'},  
 {'data': '18/03/1999', 'valor': '45.00'},  
 {'data': '19/03/1999', 'valor': '45.00'}]
```

```
>> Print df_selic
```

```
data  valor  
0    05/03/1999  45.00  
1    06/03/1999  45.00  
2    07/03/1999  45.00  
3    08/03/1999  45.00  
4    09/03/1999  45.00  
...   ...       ...  
8870  17/06/2023 13.75  
8871  18/06/2023 13.75  
8872  19/06/2023 13.75  
8873  20/06/2023 13.75  
8874  21/06/2023 13.75  
[8875 rows x 2 columns]
```



3.4. Biblioteca Json

A biblioteca json em Python é usada para lidar com dados JSON (JavaScript Object Notation). Ela fornece métodos para serializar (converter objetos Python em JSON) e desserializar (converter de JSON para objetos Python) dados.

- **json.dumps():** Este método é usado para serializar (codificar) um objeto Python em um JSON. Ele recebe um objeto Python como entrada e retorna um JSON correspondente

Exemplo:

```
import json

json_str = '{"Wege": "20", "Vale": 30}'
json_convvt = json.loads(json_str)
print(type(json_convvt))
json_convvt2 = json.dumps(json_convvt)
print(type(json_convvt2))
```



Resposta:

```
>> <class 'dict'>
>> <class 'str'>
```

- **json.loads():** Este método é usado para desserializar (decodificar) um JSON em um objeto Python. Ele recebe um JSON como entrada e retorna o objeto Python correspondente

Exemplo:

```
import json

json_str = '{"Wege": "20", "Vale": 30}'
print(type(json_str))
json_convvt = json.loads(json_str)
print(type(json_convvt))
```

Resposta:

```
>> <class 'str'>
>> <class 'dict'>
```

Mundo 4

4.1. API do Banco Central

Neste mundo aprenderemos sobre a API do Banco Central, e alguns detalhes importantes que podem ser reutilizados em outras API's. Cada API possui sua própria documentação e seu jeito de trabalhar. Por isso é importante que você leia cada uma para poder entender melhor. Segue abaixo a API do banco central:

https://dadosabertos.bcb.gov.br/dataset?res_format=API

4.2. Conceitos importantes

Como dito anteriormente, cada API tem suas próprias propriedades, contudo algumas propriedades se repetem em outras APIs.

4.2.1. Paginação

A paginação existe por um simples motivo: dividir a abundante quantidade de dados, com o intuito de otimizar. Esta prática é muito comum em APIs que disponibilizam dados históricos.

Em resumo, essa quantidade de dados é dividida em partes menores que darão mais escalabilidade e facilidade na hora do acesso das informações. Claro que isso significa, também, fazer mais de um request para conseguir ter os dados completos.

Existem algumas maneiras de se acessarem essas diferentes páginas, as duas mais comuns são:

Colocar no endpoint o número da página acessada: <https://www.exemplo.com.br/wfkajbwiaejn/page?=2>

2. Colocar dentro do request um campo parecido com:

```
{"next_page": " https://www.exemplo.com.br/wfkajbwiaejn }
```

4.3. Construtor de query

Não é comum, mas algumas APIs disponibilizam um construtor de query. Este construtor é utilizado para criar endpoints, utilizados para obter as informações que você deseja, cada API tem seu próprio padrão para endpoints.

Máximo: Define a quantidade de itens, sendo no máximo 10000

Filtro: Filtrar as opções desejadas

Ordenação: Como os dados vão ser ordenados, pode ser por data, valor, ascendente, menor pro maior, ou descendente, maior pro menor.

Campos: Define os campos que entram dentro da sua query:

The screenshot shows a query builder interface with the following settings:

- Primeiro: 1
- Máximo: 100
- Filtro: Exemplo: Nome eq 'João'
- Ordenação: Exemplo: Nome asc, Idade desc
- Saída: json
- Campos: Data, Quantidade, Valor (R\$), Denominação, Espécie
- URL de pesquisa: https://olinda.bcb.gov.br/olinda/service/mecir_dinheiro_em_circulacao/versao/v1/odata/informacoes_diarias?Stop=100&\$format=json

Buttons at the bottom: Copiar URL, Baixar json, Executar.

Se você preencher as informações e clicar em “executar”, as informações são disponibilizadas logo abaixo:

The results table displays the following data:

Data	Quantidade	Valor (R\$)	Denominação	Espécie
1994-10-03	692701959	6927019.59	0.01	Moedas
1994-10-03	462277579	23113878.95	0.05	Moedas
1994-10-03	404559065	40455906.5	0.10	Moedas
1994-10-03	1492870	373217.5	0.25	Moedas
1994-10-03	278901842	139450921	0.50	Moedas
1994-10-03	267853898	267853898	1.00	Cédulas
1994-10-03	181609358	181609358	1.00	Moedas
1994-10-03	252922174	1264610870	5.00	Cédulas
1994-10-03	273630983	2736309830	10.00	Cédulas
1994-10-03	28945486	1447274300	50.00	Cédulas
1994-10-03	2841789	284178900	100.00	Cédulas

São essas informações que serão disponibilizadas a partir do request que será feito dentro do Python.

Repare como os endpoints podem ser modificados, cada parte do endpoint abaixo tem uma função específica dentro da sua requisição:

Url base: [https://olinda.bcb.gov.br/olinda/servico/mecir_dinheiro_em_circulacao/versao/v1/odata/\[codigo_recurso\]?\\$format=json&\[Outros Parâmetros\]](https://olinda.bcb.gov.br/olinda/servico/mecir_dinheiro_em_circulacao/versao/v1/odata/[codigo_recurso]?$format=json&[Outros Parâmetros])

Url após utilizar o construtor: [https://olinda.bcb.gov.br/olinda/servico/mecir_dinheiro_em_circulacao/versao/v1/odata/informacoes_diarias?\\$top=100&\\$format=json&\\$select=Data,Quantidade,Valor,Denominacao,Especie](https://olinda.bcb.gov.br/olinda/servico/mecir_dinheiro_em_circulacao/versao/v1/odata/informacoes_diarias?$top=100&$format=json&$select=Data,Quantidade,Valor,Denominacao,Especie)

Requisição:

Esta requisição é apenas para a primeira página de dados:

```
import requests
import pandas as pd

url_base = "https://olinda.bcb.gov.br/olinda/servico/mecir_dinheiro_em_circulacao/versao/v1/odata/informacoes_diarias?$top=100&$format=json&$select=Data,Quantidade,Valor,Denominacao,Especie"
response = requests.get(url_base)

json_moedas = response.json()
df_moedas = pd.DataFrame(json_moedas["value"])

print(df_moedas)
```

Resposta:

Data	Quantidade	Valor	Denominacao	Especie
0	1994-10-03	692701959	6.927020e+06	0.01 Moedas
1	1994-10-03	462277579	2.311388e+07	0.05 Moedas
2	1994-10-03	404559065	4.045591e+07	0.10 Moedas
3	1994-10-03	1492870	3.732175e+05	0.25 Moedas
4	1994-10-03	278901842	1.394509e+08	0.50 Moedas
..
95	1994-10-14	266887039	1.334435e+09	5.00 Cédulas
96	1994-10-14	295033117	2.950331e+09	10.00 Cédulas
97	1994-10-14	30173106	1.508655e+09	50.00 Cédulas
98	1994-10-14	3220680	3.220680e+08	100.00 Cédulas
99	1994-10-17	721819389	7.218194e+06	0.01 Moedas

[100 rows x 5 columns]

Url após a modificação:<https://bit.ly/46oVstC>

O parâmetro “top” define a quantidade de dados a ser retornado enquanto o parâmetro “skip” define a quantidade de linhas a serem puladas, ou seja, se a gente quer a linha 10000 é só colocar o “skip=10000”.

Após entender os parâmetros, é só jogar tudo em uma estrutura de repetição e estruturar da melhor forma:

Obs: Deixar o pd.concat() fora da estrutura de repetição é uma boa prática de programação, já que esse método exige bastante.

Requisição:

É desta forma que você faz quando necessita da base histórica completa. Como dito anteriormente, você precisa consultar a documentação da API para poder entender como é feito a paginação, no caso da API do banco central é utilizado intervalos:

```
import requests
import pandas as pd

pular = 0

lista_df_moedas = []

while True:

    url_base = f"https://olinda.bcb.gov.br/olinda/servico/mecir_dinheiro_em_circulacao/versao/v1/odata/informacoes_diarias?$top=10000&$skip={pular}&$format=json&$select=Data,Quantidade,Valor,Denominacao,Especie"
    response = requests.get(url_base)

    pular = pular + 1000

    json_moedas = response.json()
    df_moedas = pd.DataFrame(json_moedas["value"])
    lista_df_moedas.append(df_moedas)

    if len(json_moedas["Value"]) < 1:
        break

base_completa = pd.concat(lista_df_moedas)
print(base_completa)
```



Resposta:

Data	Quantidade	Valor	Denominacao	Especie
0	1994-10-03	692701959	6.927020e+06	0.01 Moedas
1	1994-10-03	462277579	2.311388e+07	0.05 Moedas
2	1994-10-03	404559065	4.045591e+07	0.10 Moedas
3	1994-10-03	1492870	3.732175e+05	0.25 Moedas
4	1994-10-03	278901842	1.394509e+08	0.50 Moedas
..
95	1994-10-14	266887039	1.334435e+09	5.00 Cédulas
96	1994-10-14	295033117	2.950331e+09	10.00 Cédulas
97	1994-10-14	30173106	1.508655e+09	50.00 Cédulas
98	1994-10-14	3220680	3.220680e+08	100.00 Cédulas
99	1994-10-17	721819389	7.218194e+06	0.01 Moedas

[94466 rows x 5 columns]

Mundo 5

5.1. API do Google Maps

Neste mundo aprenderemos sobre a API do Google Maps, esta é uma das APIs mais difíceis do curso, mas não se preocupe, pois estou aqui para descomplicar isso para você. Visite a documentação sempre que estiver com alguma dúvida:

<https://github.com/googlemaps/google-maps-services-python>

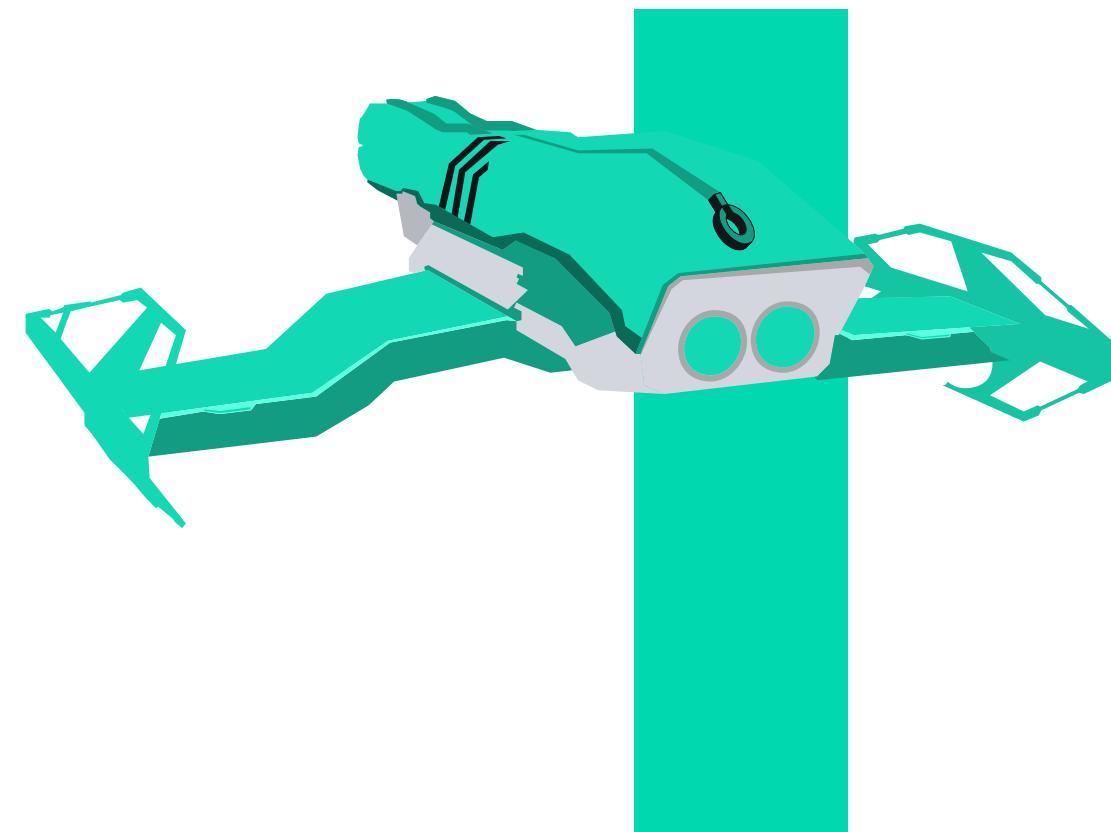
5.2. Gerenciamento de dados sensíveis

1. Segurança: Se um programa Python for comprometido ou acessado por uma pessoa não autorizada, qualquer dado sensível armazenado nele estará em risco. Isso inclui informações como senhas, chaves de criptografia, números de cartão de crédito ou qualquer outro dado pessoal identificável. É importante proteger esses dados armazenando-os de forma segura e evitando deixá-los no código do programa.

2. Controle de acesso: Quando você armazena dados sensíveis diretamente no código Python, qualquer pessoa que tenha acesso ao código também terá acesso a esses dados. Isso pode incluir desenvolvedores, administradores de sistemas ou qualquer pessoa que possa visualizar ou modificar o código. Armazenar dados sensíveis separadamente e aplicar medidas adequadas de controle de acesso ajudará a reduzir a exposição desses dados.

3. Manutenção e escalabilidade: Ao armazenar dados sensíveis diretamente no código Python, você pode enfrentar desafios quando precisar atualizar, modificar ou escalar o programa. A cada vez que você precisar fazer uma alteração no código, terá que lidar com a manipulação cuidadosa dos dados sensíveis. Isso pode ser demorado e aumentar o risco de erro humano.

4. Boas práticas de desenvolvimento: É considerada uma boa prática separar dados sensíveis do código do programa. Em vez de deixá-los no código, é recomendável armazená-los em um local seguro, como um banco de dados protegido por senha, um arquivo de configuração criptografado ou um serviço de gerenciamento de segredos. Isso torna o código mais limpo, modular e reutilizável, além de melhorar a segurança geral do sistema.



5.3. Se conectando ao Google

5.3.1. Baixando os pacotes

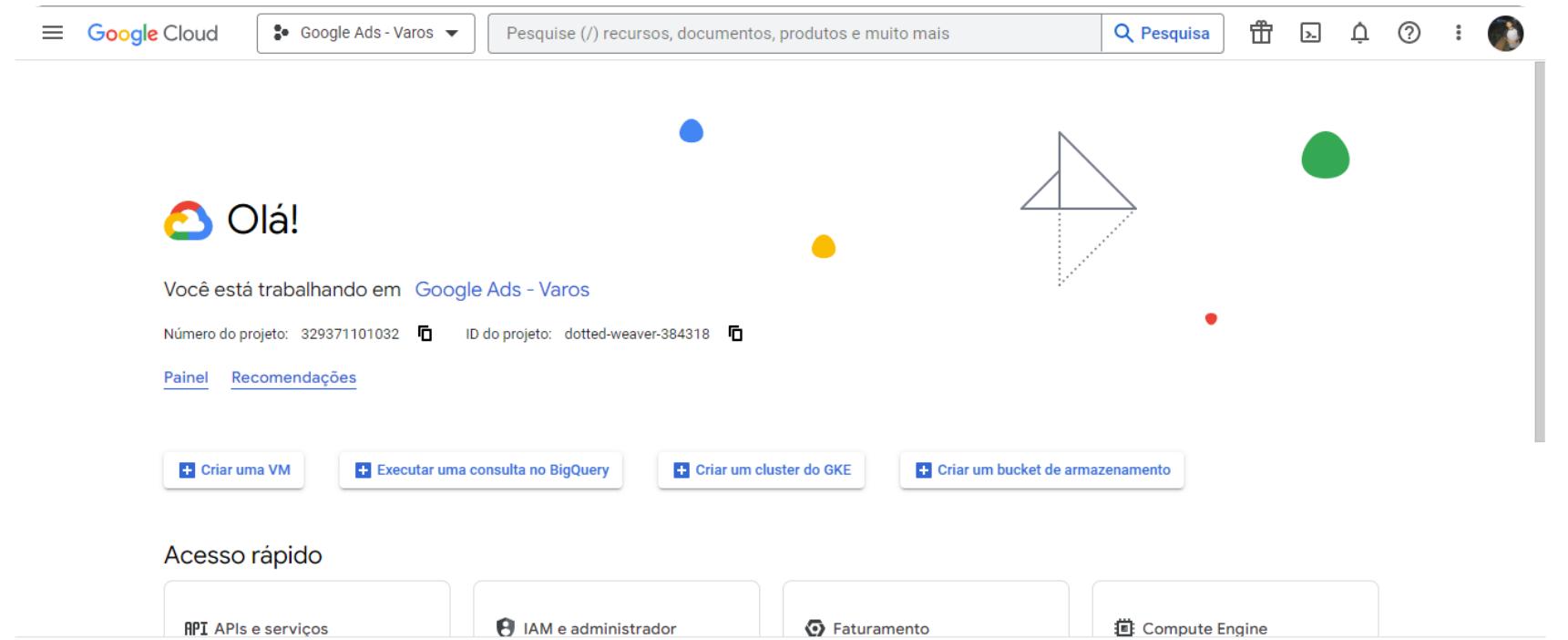
Baixe os seguintes pacotes para ter contato direto com o sistema do Google:

```
%pip install -U googlemaps #se conecta ao google maps  
%pip install pprintpp #interpreta dados json  
%pip install python-dotenv #gerencia dados sensiveis
```

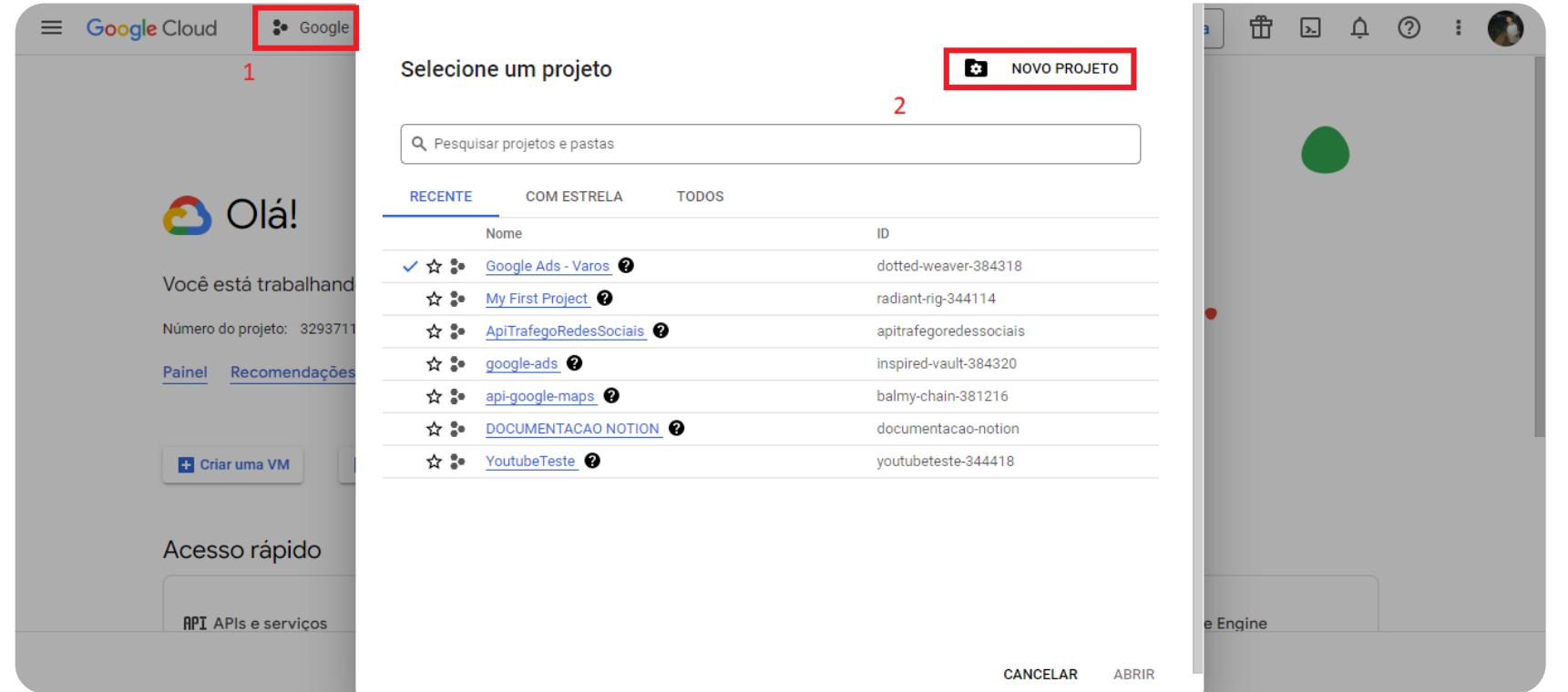
5.3.2. Criando uma conta no Google Maps

Após ter a conta criada, acesse o seguinte site:

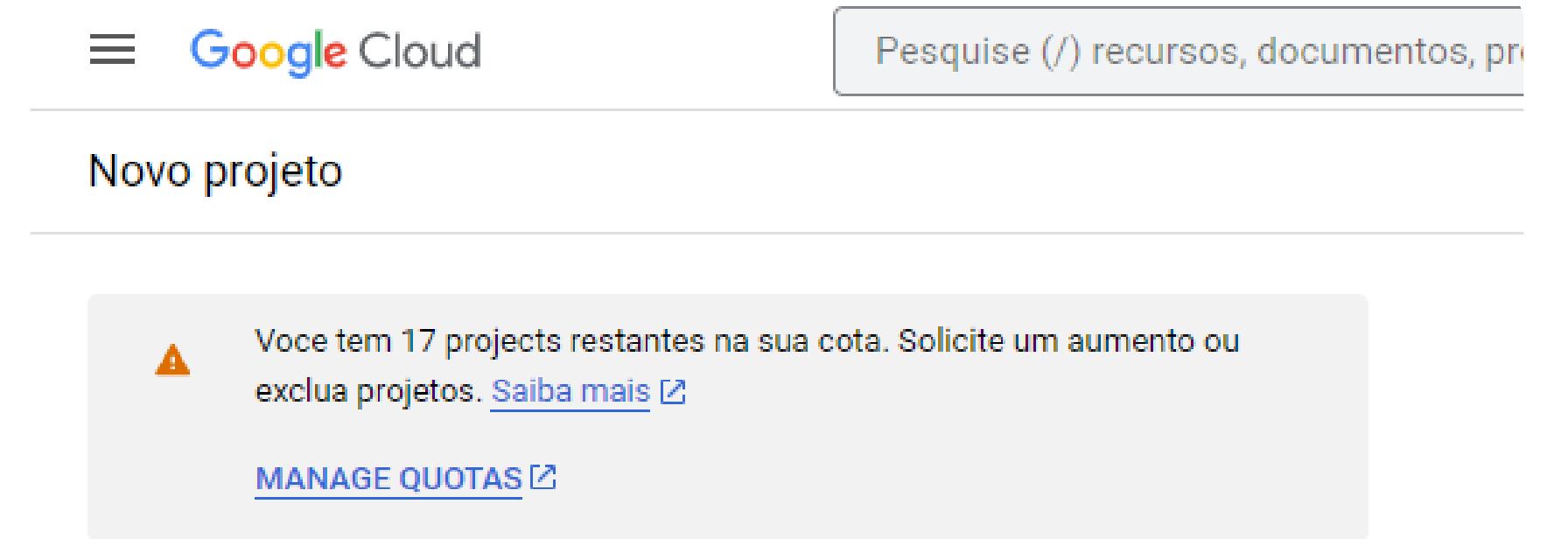
<https://bit.ly/46nLBUT>



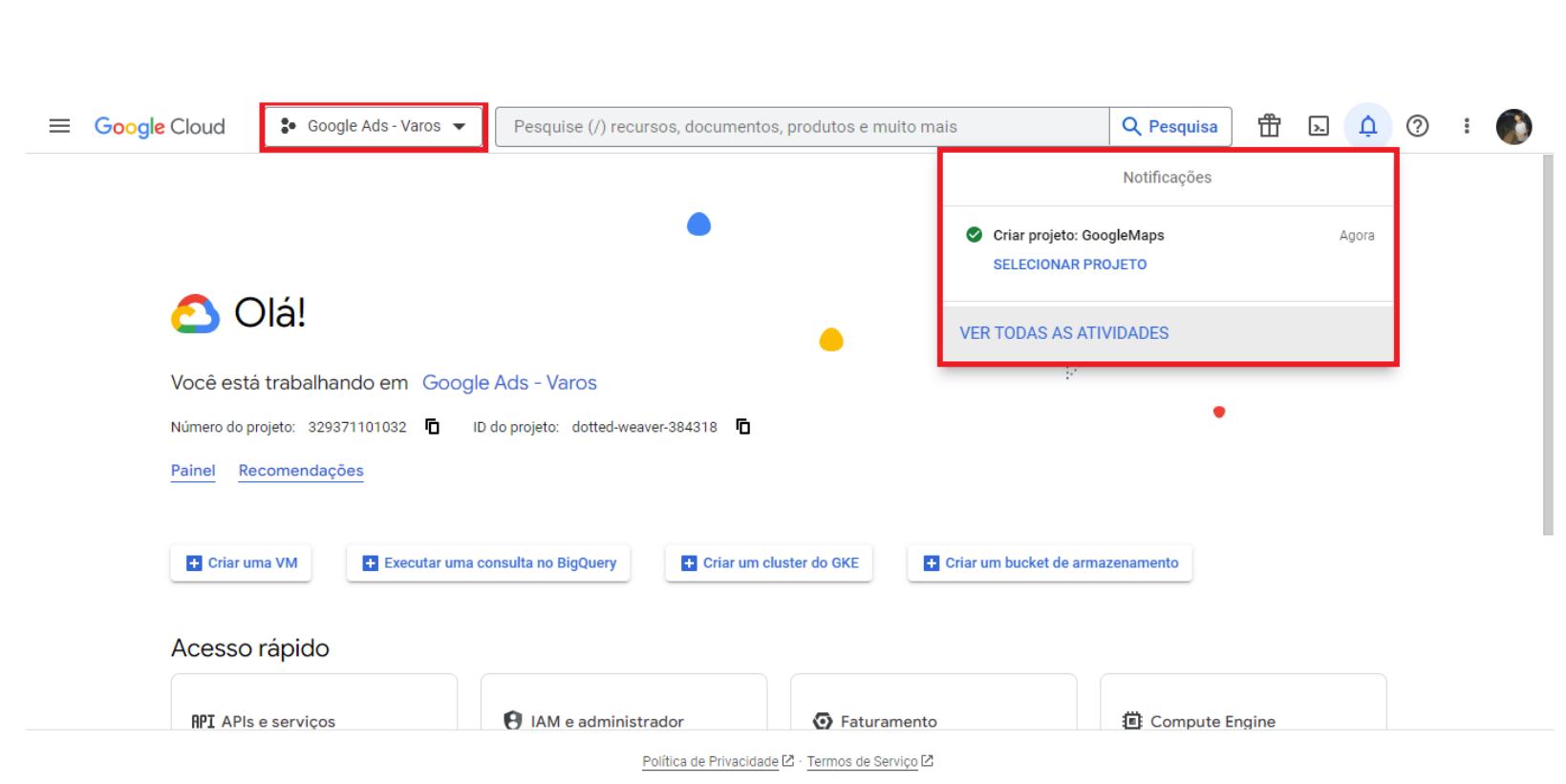
Clique no canto superior esquerdo e comece um novo projeto:



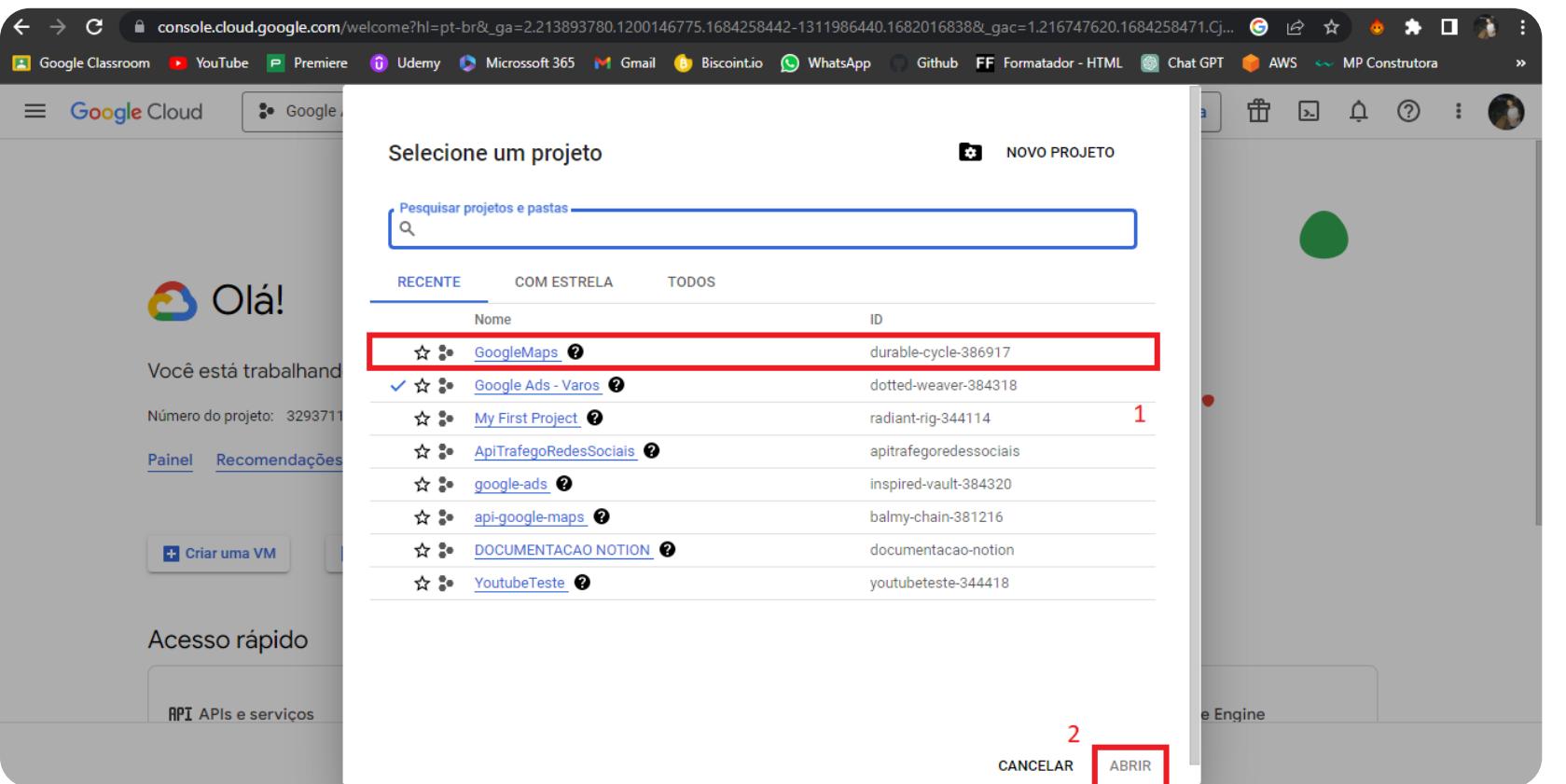
Após ser redirecionado, escolha um nome para o seu projeto e clique no botão azul em “criar”:



Aparecerá uma notificação confirmando que foi tudo criado corretamente, depois clique no canto esquerdo superior e selecione o projeto que você acabou de criar.



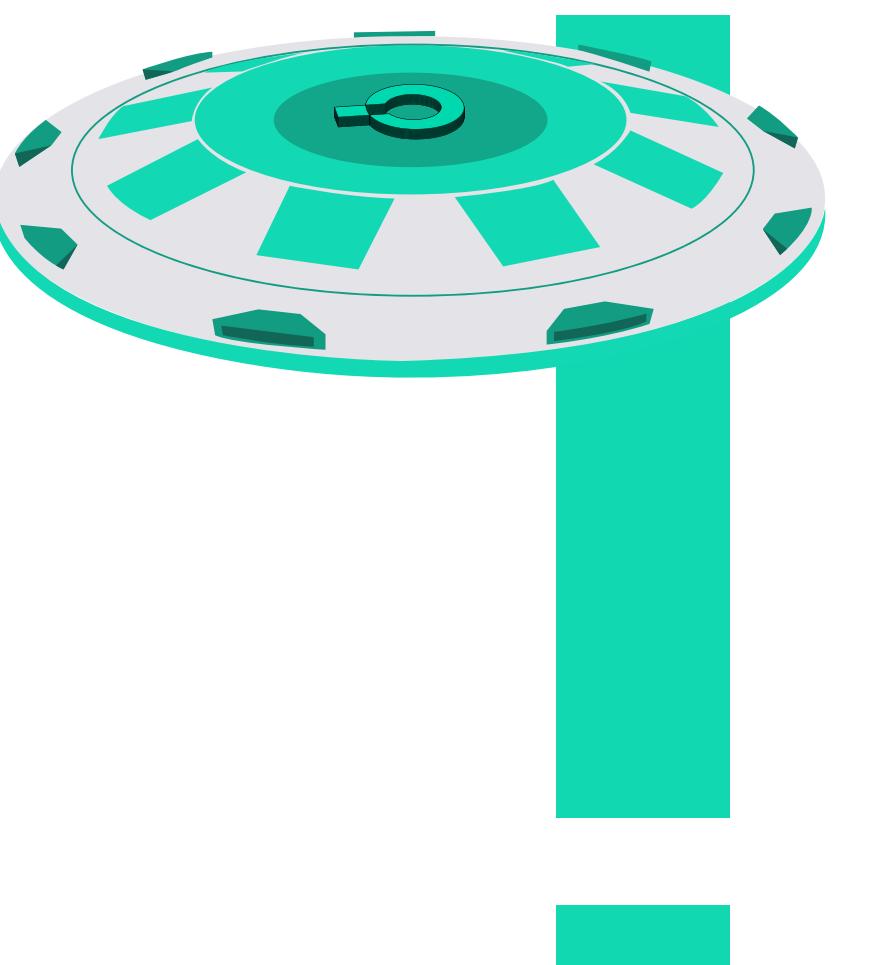
E clique em abrir:



Vá até o link abaixo:

<https://bit.ly/3PpHa5R>

Vá até a parte de [+ CRIAR CREDENCIAIS] e clique em "Chaves de API".



Google Cloud API APIs e serviços Credenciais + CRIAR CREDENCIAIS EXCLUIR RESTAURAR CREDENCIAIS EXCLUIDAS

Chave de API Crie credenciais para acessar suas APIs ativas. Saiba mais

ID do cliente OAuth Lembre-se de configurar a tela de consentimento para seu aplicativo.

Contas de serviço Usa contas robô para ativar a autenticação do nível do app entre servidores.

Ajude-me a escolher Faz algumas perguntas para ajudar você a decidir que tipo de credencial usar.

IDs do cliente OAuth 2.0

Contas de serviço

Chave de API criada

Sua chave de API: **Chave de API 1**

Esta chave não tem restrições. Para evitar o uso não autorizado, recomendamos restringir onde e para quais APIs ela pode ser usada. Edite a chave de API para adicionar restrições. Saiba mais

EXIBIR CHAVE

Automaticamente aparecerá uma nova janela com sua chave de API. Guarde ela corretamente e não a compartilhe... Lembre-se que seu cartão de crédito está vinculado a esta API.

Copie esta chave de API para o .env

Obs: No lugar da linha vermelha deverá ter uma chave de API.

Sua chave aparecerá logo abaixo:

Chaves de API

Nome	Data da criação	Restrições	Ações
Chave de API 1	16 de mai. de 2023	Nenhum	EXIBIR CHAVE

Após ter a chave criada, falta um último passo que é ativar o serviço do Google que utilizaremos que se chama "Geocoding API".

Só clicar no link abaixo e ativar o serviço:

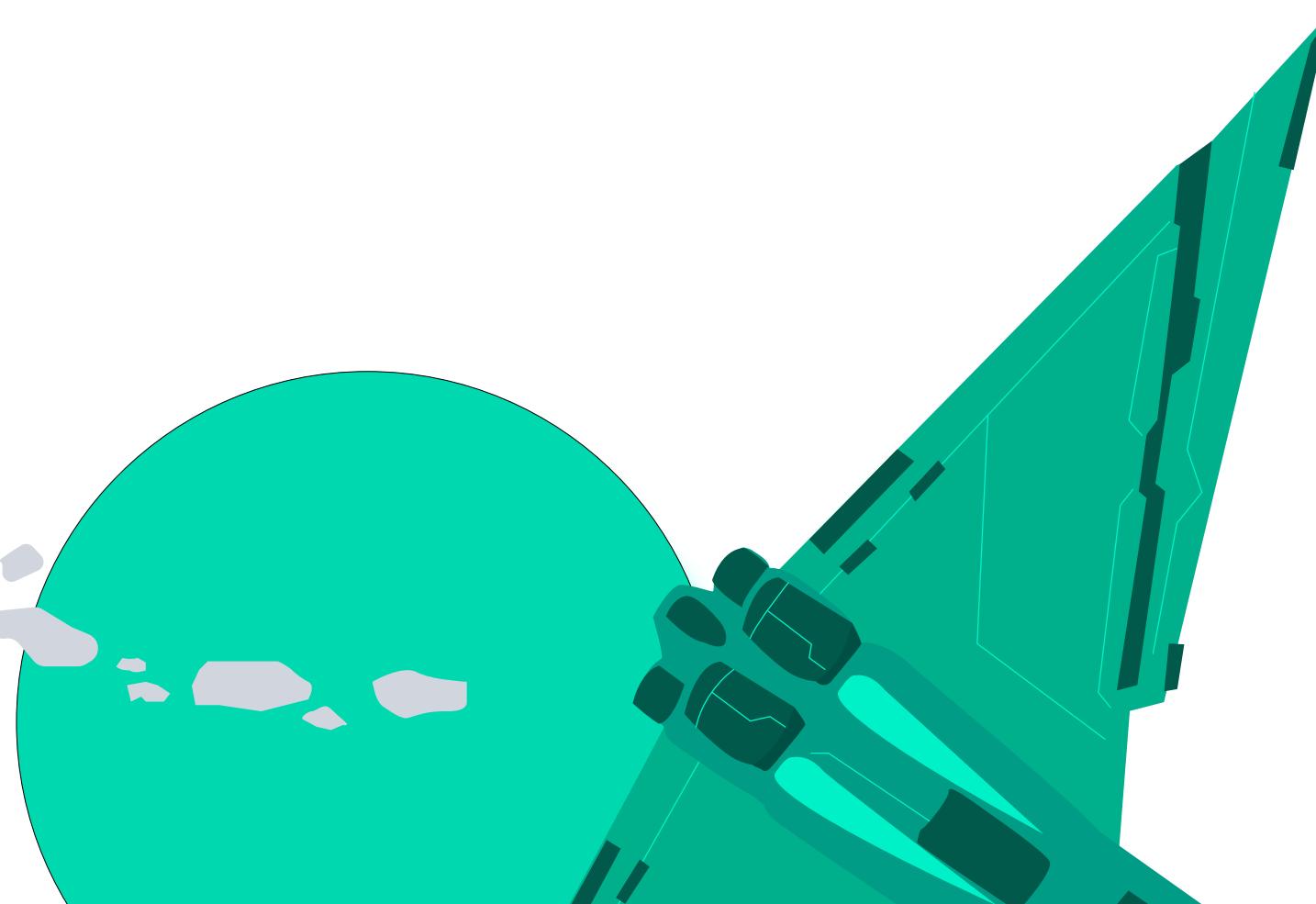
<https://bit.ly/3plda0d>

The screenshot shows the Google Cloud Platform interface with the 'Google Cloud' logo and a dropdown menu for 'GoogleMaps'. Below the header, there's a search bar and a 'Detalhes do produto' button. The main content area displays the 'Geocoding API' product details. It includes a circular icon with a blue dot, the product name 'Geocoding API', and a link to 'Google Enterprise API'. A brief description states: 'Convert between addresses and geographic coordinates.' Below this is a large red 'ATIVAR' (Activate) button. At the bottom of the page, there are tabs for 'VISÃO GERAL' (General View), 'DOCUMENTAÇÃO' (Documentation), 'SUPORTE' (Support), and 'PRODUTOS RELACIONADOS' (Related Products). On the left side, there's a sidebar with sections like 'Visão geral' (General View) and 'Mais detalhes' (More details), which lists the product type as 'SaaS & APIs', the last update date as '28/09/2022', the category as 'Google Enterprise APIs, Maps', and the service name as 'geocoding-backend.googleapis.com'.

5.3.4. Se conectando a API do Google Maps na prática

Vou fazer uma requisição, POST, enviando o endereço do Maracanã e como resposta receberei as informações sobre esse endereço.

Obs: Estamos utilizando a biblioteca do Google, porém por baixo dos panos, tudo que ele está fazendo são requisições para endpoints específicos.



Exemplo:

```
import googlemaps
from pprint import pprint
import time
import os

chave_api = os.getenv("chave_googleMaps")

client = googlemaps.Client(chave_api)

endereco_maracana = "Av. Pres. Castelo Branco, Portão 3 - Maracanã, Rio de Janeiro - RJ"

response = client.geocode(endereco_maracana)

pprint(response)
```

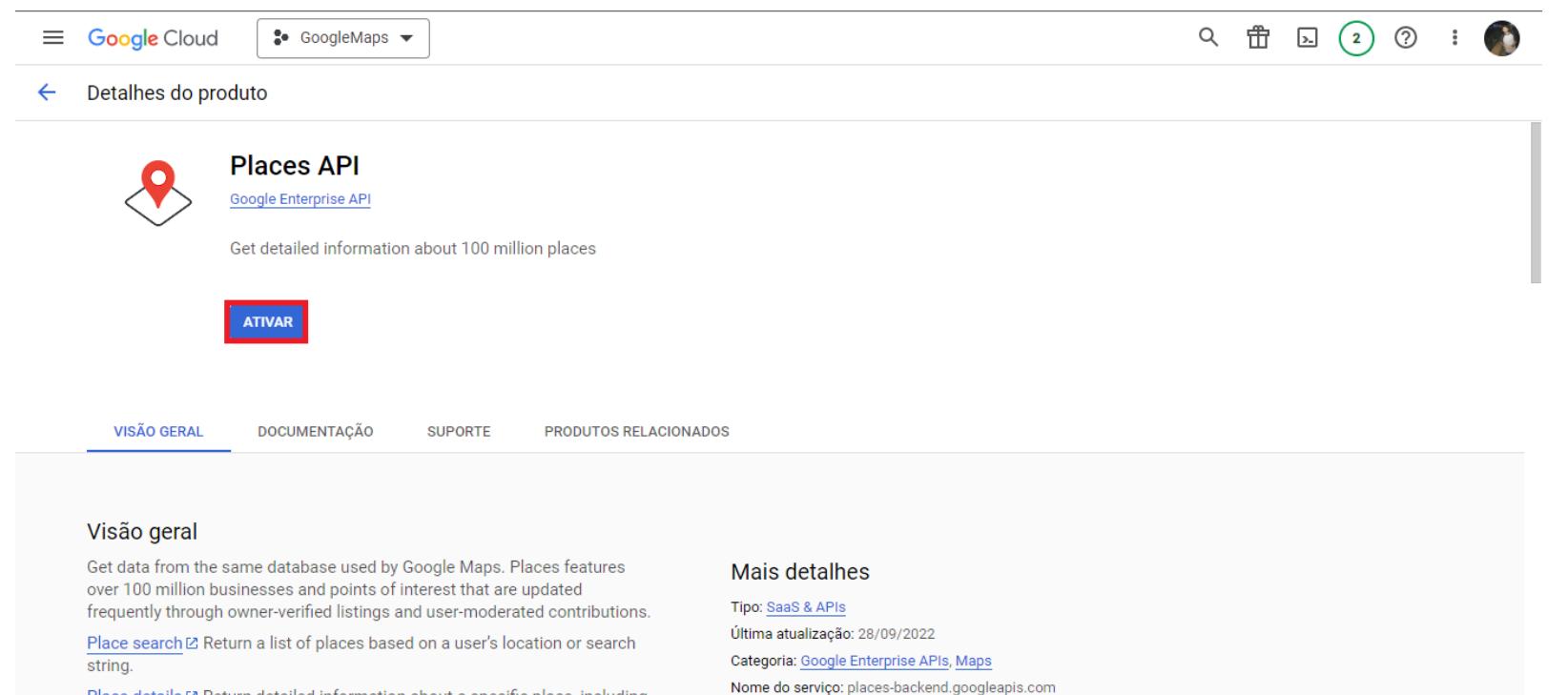
Resposta:

```
[{"address_components": [{"long_name": "Avenida Presidente Castelo Branco",
   "short_name": "Av. Pres. Castelo Branco",
   "types": ["route"]},
  {"long_name": "Maracanã",
   "short_name": "Maracanã",
   "types": ["political",
             "sublocality",
             "sublocality_level_1"]},
  {"long_name": "Rio de Janeiro",
   "short_name": "Rio de Janeiro",
   "types": ["administrative_area_level_2",
             "political"]},
  {"long_name": "Rio de Janeiro",
   "short_name": "RJ",
   "types": ["administrative_area_level_1",
             "political"]},
  {"long_name": "Brazil",
   "short_name": "BR",
   "types": ["country", "political"]},
  {"long_name": "20550-011",
   "short_name": "20550-011",
   "types": ["postal_code"]}],
 "formatted_address": "Av. Pres. Castelo Branco - Maracanã, Rio de Janeiro - RJ, 20550-011, Brazil",
 "geometry": {"bounds": {"northeast": {"lat": -22.9087371, "lng": -43.2381983},
   "southwest": {"lat": -22.9090022, "lng": -43.2384061}},
  "location": {"lat": -22.9087851, "lng": -43.2383529},
  "location_type": "GEOMETRIC_CENTER",
  "viewport": {"northeast": {"lat": -22.9075206697085,
    "lng": -43.2369532197085},
   "southwest": {"lat": -22.9102186302915,
    "lng": -43.2396511802915}}},
 "partial_match": True,
 "place_id": "ChIJdXxR_V5-mQARfuR3KznWT7A",
 "types": ["route"]}]
```

5.4. Ativando o serviço “Places”

Clique no link abaixo e ative o serviço:

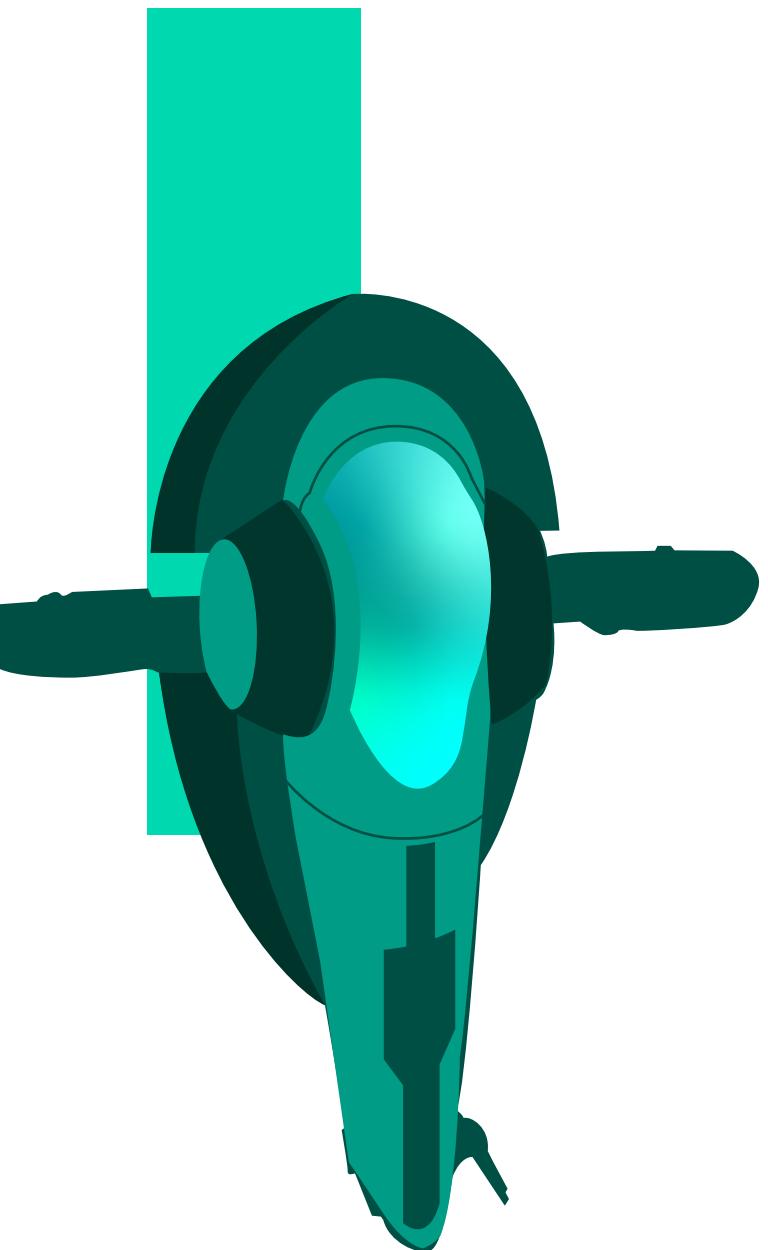
<https://bit.ly/3NvUbIC>



The screenshot shows the Google Cloud Platform interface with the 'Google Cloud' and 'GoogleMaps' tabs selected. Under 'Detalhes do produto', the 'Places API' is listed under 'Google Enterprise API'. It describes the API as providing detailed information about 100 million places. A red box highlights the 'ATIVAR' (Enable) button. Below the API details, there are sections for 'Visão geral' and 'Mais detalhes'.

Visão geral
Get data from the same database used by Google Maps. Places features over 100 million businesses and points of interest that are updated frequently through owner-verified listings and user-moderated contributions.
[Place search](#) Return a list of places based on a user's location or search string.
[Place details](#) Return detailed information about a specific place, including

Mais detalhes
Tipo: SaaS & APIs
Última atualização: 28/09/2022
Categoria: Google Enterprise APIs, Maps
Nome do serviço: places-backend.googleapis.com



Mundo 6

6.1. API do IBGE

Neste mundo aprenderemos sobre a API do IBGE, nesta API você encontrará dados econômicos, socioeconômicos e estatísticos do Brasil e do mundo que farão melhorar suas análises.

Um ponto positivo é que para se conectar a esta API você não precisará de uma chave de autenticação, isso facilita a nossa vida não é mesmo?

<https://servicodados.ibge.gov.br/api/docs>

6.2. Projeções populacionais

Esta é a parte da API do IBGE que disponibiliza a projeção populacional do Brasil ou suas regiões. Sua URL base é:

<https://servicodados.ibge.gov.br/api/v1/projecoes/populacao/{localidade}>

Onde, localidade pode ser:

País: BR.

Regiões:

- 1 - Norte.
- 2 - Nordeste.
- 3 - Sudeste.
- 4 - Sul.
- 5 - Centro-Oeste.

Exemplo:

```
import requests
import pandas as pd
import json

localidade = 'BR'
url = f'https://servicodados.ibge.gov.br/api/v1/projecoes/populacao/{localidade}'

response = requests.get(url)
data = json.loads(response.text)

print(data)
```

6.3. Indicadores econômicos por país

Esta é a parte da API do IBGE que disponibiliza os indicadores do Brasil e do mundo. Sua URL base é:

<https://servicodados.ibge.gov.br/api/v1/paises/{paises}/indicadores/{indicadores}>

Onde indicadores e paises podem ser:

paises: Código de 2 dígitos de acordo com norma ISO 3166-1 AL-PHA-2 (Exemplo: BR, US).

indicador: Um número de acordo com o site do IBGE (ele define id's).

obs: é possível obter mais de um indicador e mais de um país na mesma requisição, para isto você vai utilizar o separador "|".

Exemplo:

```
import requests
import pandas as pd
import json

paises = 'BR|US'
indicadores = '77823' #PIB per capita
nome_pais = ['Brasil', 'EUA']

url = f'{https://servicodados.ibge.gov.br/api/v1/paises/{paises}/indicadores/{indicadores}}'

response = requests.get(url)
data = json.loads(response.text)

print(data)
```

Resposta:

```
{'id': 77823, 'indicador': 'Economia - PIB per capita', 'unidade': {'id': 'US$', 'classe': 'N', 'multiplicador': 1}, 'series': [{['pais': {'id': 'BR', 'nome': 'Brasil'}, 'serie': [{}: None], '1990': '2592.63'}, {'1990-1995': None}, {'1995': '4704.96'}, {'1995-2000': None}, {'1999-2001': None}, {'2000': '3726.81'}, {'2000-2002': None}, {'2000-2005': None}, {'2001': '3142.28'}, {'2001-2003': None}, {'2002': '2824.68'}, {'2002-2004': None}, {'2003': '3056.62'}, {'2003-2005': None}, {'2004': '3623.25'}, {'2004-2006': None}, {'2005': '4773.25'}, {'2005-2007': None}, {'2005-2010': None}], ['pais': {'id': 'US', 'nome': 'Estados Unidos da América'}, 'serie': [{}: None], '1990': '23888.60'}, {'1990-1995': None}, {'1995': '28690.88'}, {'1995-2000': None}, {'1999-2001': None}, {'2000': '36329.96'}, {'2000-2002': None}, {'2000-2005': None}, {'2001': '37133.62'}, {'2001-2003': None}, {'2002': '37997.76'}, {'2002-2004': None}, {'2003': '39490.27'}, {'2003-2005': None}, {'2004': '41724.63'}, {'2004-2006': None}, {'2005': '44123.41'}, {'2005-2007': None}, {'2005-2010': None}]]}
```

Mundo 7

7.1. API da Binance

Neste mundo aprenderemos sobre a API da Binance, onde poderemos fazer operações automáticas utilizando Python.

<https://python-binance.readthedocs.io/en/latest/overview.html>

7.2. Regras

Antes de se conectar a API da Binance você precisa conhecer algumas regras de uso.

- 1 - Você precisa ter uma conta na Binance.
- 2 - Você precisa ter o 2FA habilitado.
- 3 - Você precisa habilitar trades pela API.
- 4 - Você precisa desabilitar a caixa de segurança.
- 5 - Você não pode fazer mais de 1200 requisições por minuto.

7.3. Criando uma conta na Binance

Acesse o site abaixo e se registre:

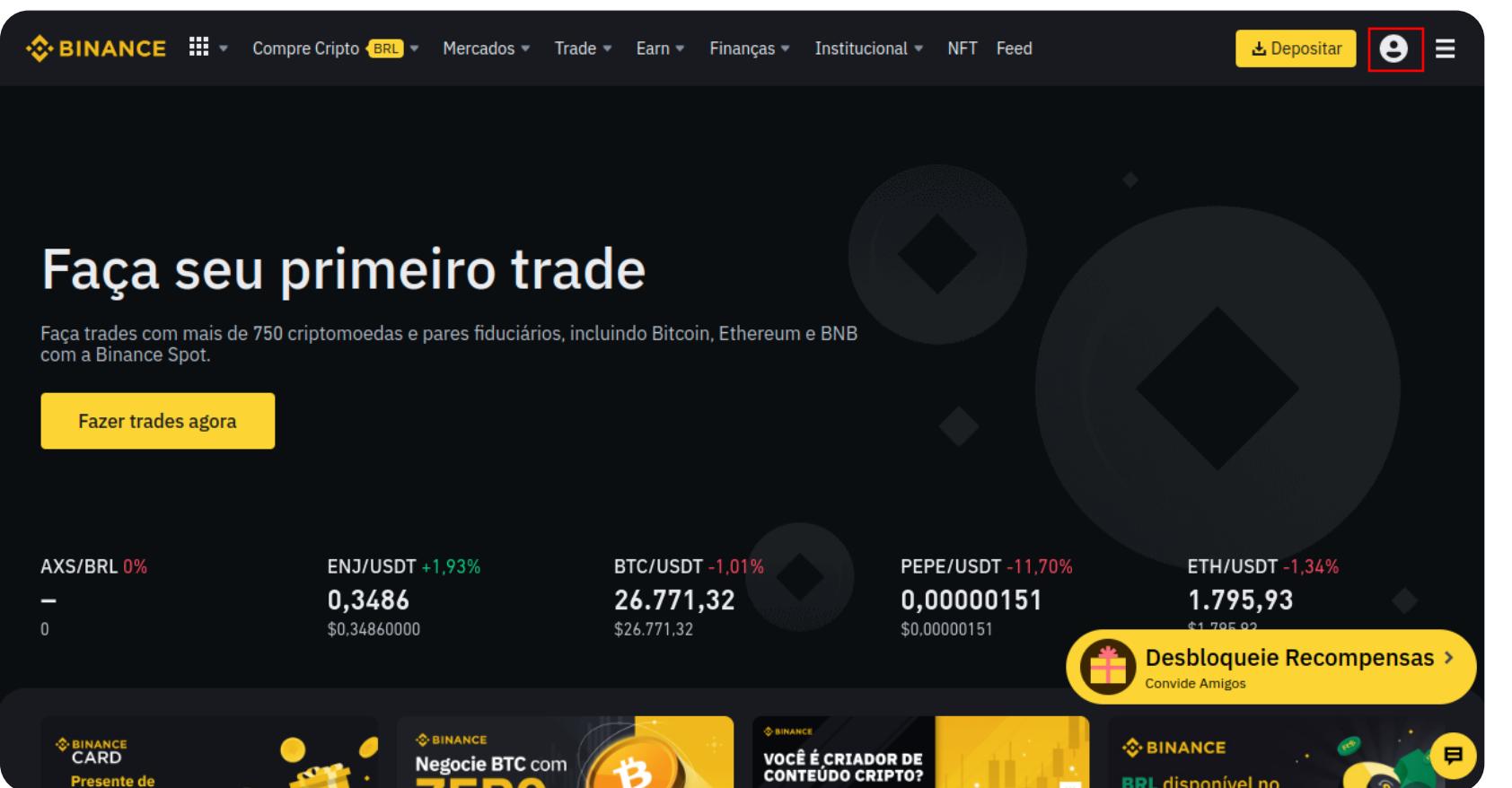
<https://accounts.binance.com/pt-BR/register-person>

7.4. Ativando 2FA

Acesse o site abaixo, com a conta já criada:

<https://www.binance.com/pt-BR>

E clique no canto direito superior, em cima do seu perfil:



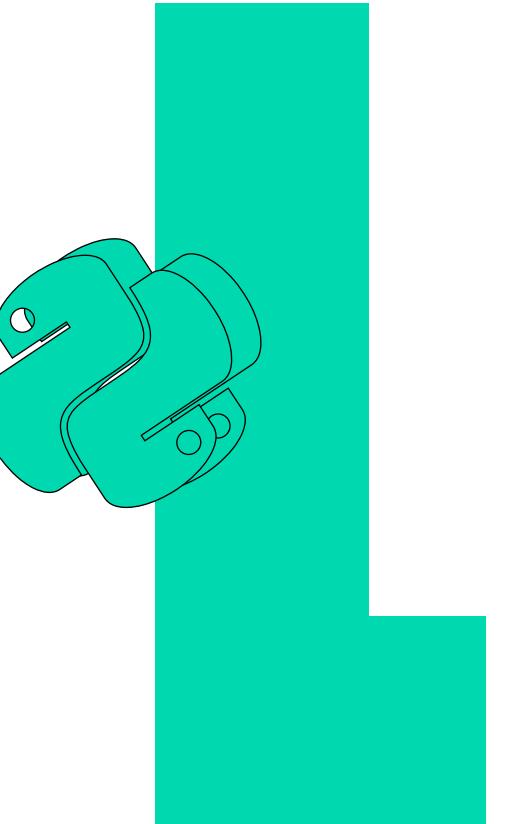
Automaticamente aparecerá uma nova janela com o campo "Segurança", clique e espere ser redirecionado:

Binance homepage featuring a 'Faça seu primeiro trade' (Make your first trade) banner. Below it, there are four crypto price cards: CTSI/USDT +22,17% (0,2072), GALA/USDT +7,07% (0,03224), PHB/USDT +14,03% (0,7836), and MATIC/USDT (0,8544). A sidebar on the right includes a 'Segurança' button highlighted with a red box.

Binance security settings page under 'Segurança'. It lists various security options: Autenticação de Dois Fatores (2FA), Verificação de Identidade, Código Anti-Phishing, and Lista de Permissões para Saques. The 'Autenticação de Dois Fatores (2FA)' section is expanded, showing Passkeys e Biometria (Not configured), Autenticador Binance/Google (Recomendado) (Not configured), Verificação por Número de Telefone (219*****161), and Verificação por Endereço de E-mail (vi***@gmail.com). A red box highlights the 'Habilitar' (Enable) button for the Google Authenticator option.

Ative os campos "Autenticador Binance/Google (Recomendado)".

Siga as etapas de cada um, para o processo ser finalizado.



7.5. Criando uma API Key

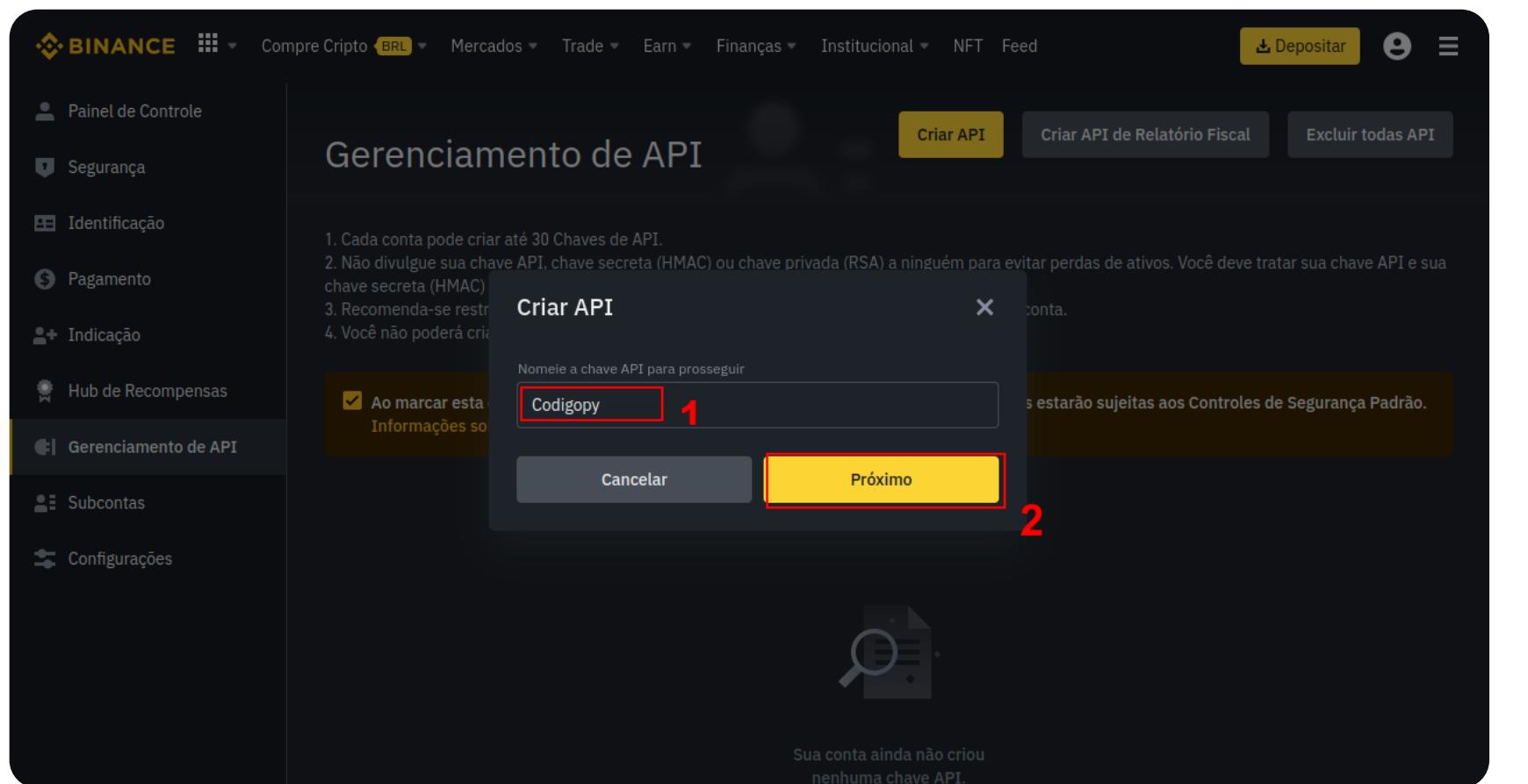
Quando esta chave estiver habilitada, como abaixo:

Clique em Gerenciamento de API.

Clique em "Criar API"

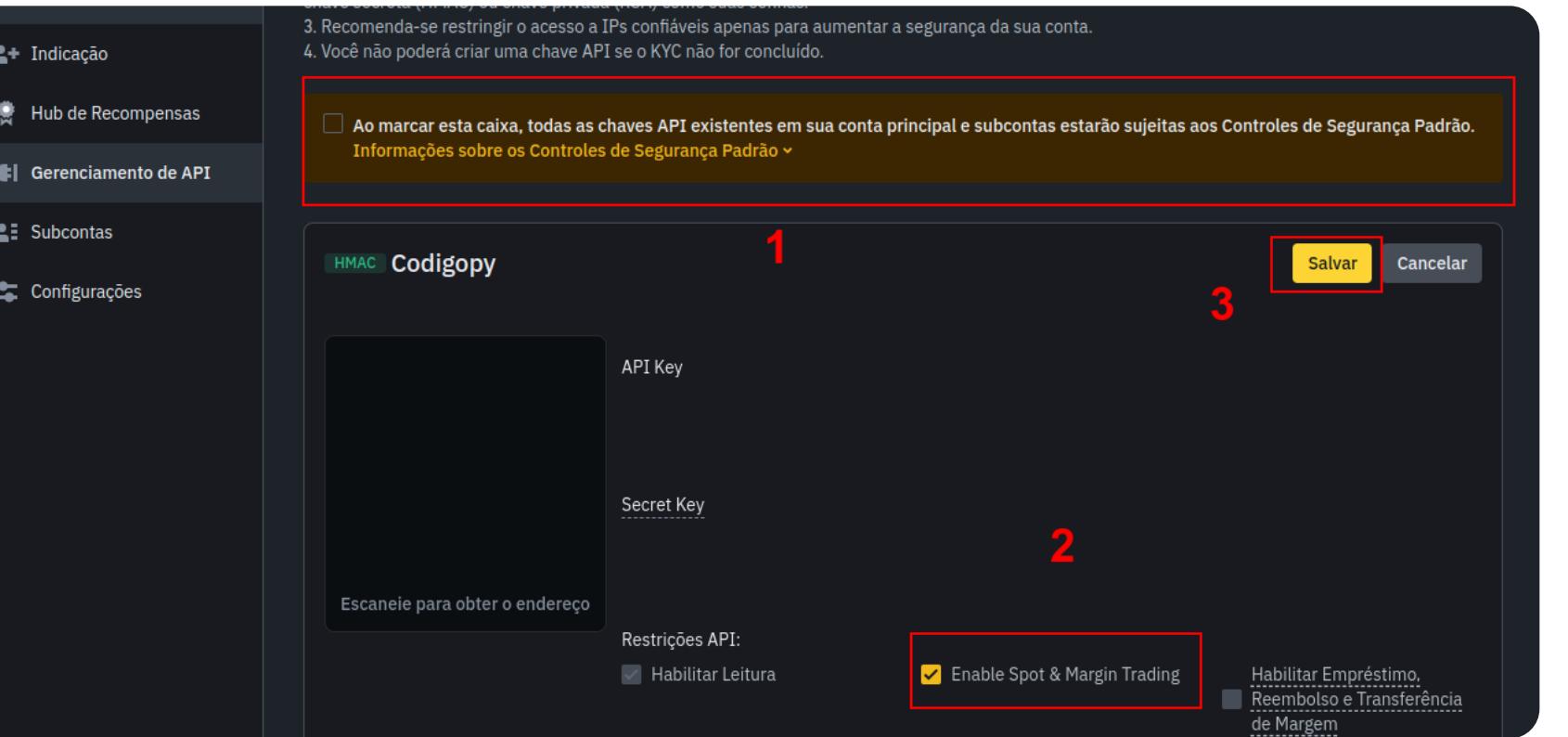
Selecione "Gerada pelo sistema" e depois "próximo":

Crie um nome e depois clique em “Próximo”:



Desmarque a opção de segurança e habilite o “Enable Spot & Margin Trading”. Não se esqueça de clicar em “Salvar”.

Obs: As informações abaixo foram escondidas



Após seguir estas etapas, você está pronto para acessar a API da Binance.

7.6. Se conectando à API através do Python

7.6.1. Instalando as dependências necessárias

```
pip install python-binance
```

```
from binance.client import Client
from binance.enums import *
from pprint import pprint
import os

api_key = os.getenv("KEY_BINANCE")
api_secret = os.getenv("SECRET_BINANCE")

cliente_binance = Client(api_key, api_secret)

print(cliente_binance)
```

7.6.2. Criando a conexão Client

Exemplo:

```
>> <binance.client.Client object at 0x7fc2b182db20>
```

7.7. Criando ordem de compra

Resposta:

Neste script abaixo, estamos efetuando uma ordem de compra de BNB em reais.

É importante definir os parâmetros corretamente, como:

SIDE_BUY = Informa que estamos efetuando a compra

SIDE_SELL = Informa que estamos efetuando a venda

ORDER_TYPE_MARKET = Estamos comprando pelo preço de mercado no momento.

Se você rodar o script conseguirá efetuar a compra da moeda dese-

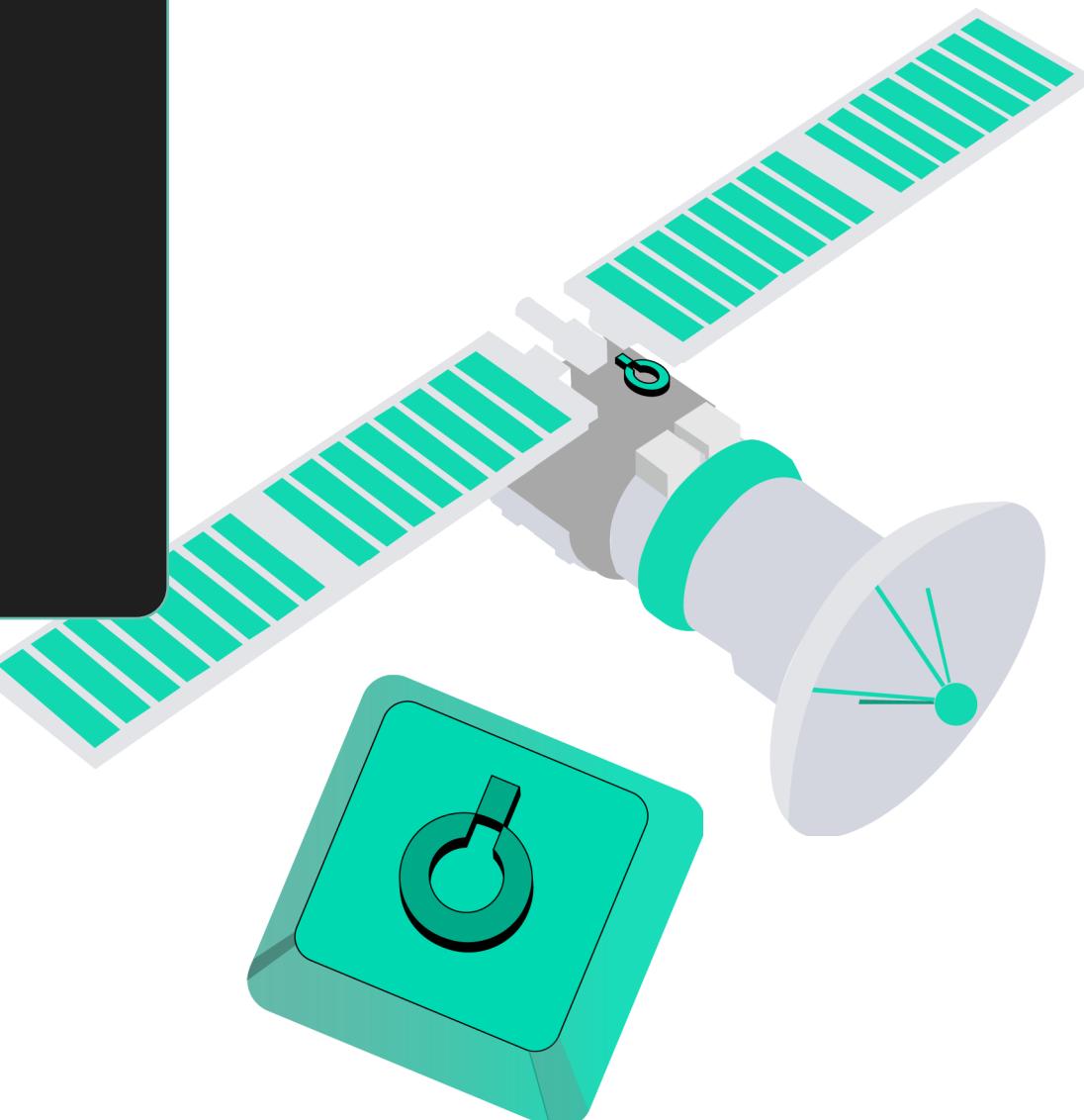
jada.

```
from binance.client import Client
from binance.enums import *
from pprint import pprint
import os

api_key = os.getenv("KEY_BINANCE")
api_secret = os.getenv("SECRET_BINANCE")

cliente_binance = Client(api_key, api_secret)

order = cliente_binance.create_order(
    symbol='BNBBRL',
    side=SIDE_BUY,
    type=ORDER_TYPE_MARKET,
    quantity=0.01)
```



Exemplo:

```
>> <binance.client.Client object at 0x7fc2b182db20>
```

7.8. Histórico de transação

7.8.1. Pegar todas as ordens

Respostas:

```
from binance.client import Client
from binance.enums import *
from pprint import pprint
import os

api_key = os.getenv("KEY_BINANCE")
api_secret = os.getenv("SECRET_BINANCE")

cliente_binance = Client(api_key, api_secret)

orders = cliente_binance.get_all_orders(symbol='BNBBRL', limit=10)
print(orders)
```

7.8.2. Pegar apenas as ordens efetuadas

Exemplo:

```
from binance.client import Client
from binance.enums import *
from pprint import pprint
import os

api_key = os.getenv("KEY_BINANCE")
api_secret = os.getenv("SECRET_BINANCE")

cliente_binance = Client(api_key, api_secret)

trades = cliente_binance.get_my_trades(symbol='BNBBRL')
pprint(trades)
```

7.9. Preços em tempo real

Exemplo:

```
from binance.client import Client
from binance.enums import *
from pprint import pprint
import os

api_key = os.getenv("KEY_BINANCE")
api_secret = os.getenv("SECRET_BINANCE")

cliente_binance = Client(api_key, api_secret)

transacoes_btc = cliente_binance.get_recent_trades(symbol='BNBBRL')
pprint(transacoes_btc[-1])
```