



APIS WEB

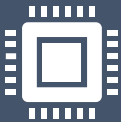
prof. Me. Hélio Esperidião



API - *Application Programming Interface*



API significa Interface de Programação de Aplicações. É um conjunto de definições e protocolos que permite a comunicação entre diferentes softwares.



Uma API define como diferentes componentes de software devem interagir uns com os outros.



Uma API pode ser entendida como uma ponte que permite que diferentes sistemas, aplicativos ou serviços se comuniquem e compartilhem recursos e funcionalidades entre si.

APIs

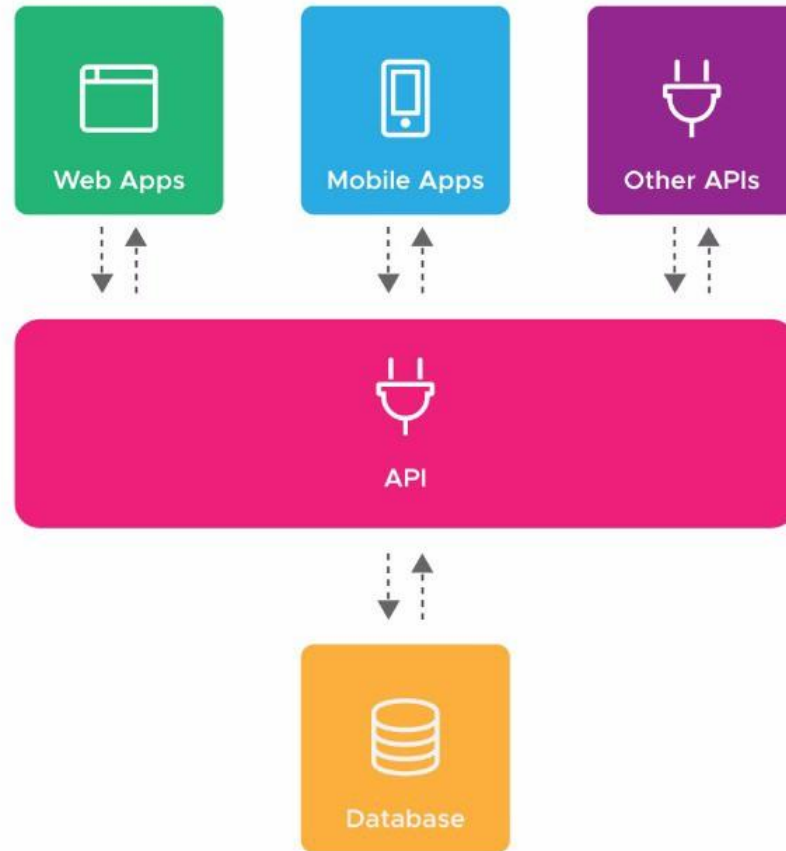


API é um conjunto de normas que possibilita a comunicação entre plataformas através de uma série de padrões e protocolos.



Por meio de APIs, desenvolvedores podem criar novos softwares e aplicativos capazes de se comunicar com outras plataformas

Esquema de funcionamento de apis



JSON - JavaScript Object Notation

É uma formatação leve de troca de dados.

Para seres humanos, é fácil de ler e escrever.

Para máquinas, é fácil de interpretar e gerar.

É baseado em um subconjunto da linguagem de programação JavaScript, Standard ECMA-262 3a Edição -Dezembro - 1999.

JSON

01

JSON é em formato texto e completamente independente de linguagem

02

Formato ideal de troca de dados

03

é um formato compacto, de padrão aberto independente, de troca de dados simples e rápida (parsing) entre sistemas

ESTRUTURA

Uma coleção de pares nome/valor. Em várias linguagens, isto é caracterizado como um *object*, record, struct, dicionário, hash table, keyed list, ou arrays associativas.

Uma lista ordenada de valores. Na maioria das linguagens, isto é caracterizado como uma *array*, vetor, lista ou sequência.

Exemplos de json

```
{"altura":57.7,"largura":55}
```

```
{"nome":"helio","cargo":"professor"}
```

Como Criar JSON em PHP com a vetor associativo

- O que é um vetor associativo em php?
 - É um vetor onde as posições podem ser textos
 - Muito usada para montar dados antes de converter para JSON

```
$pessoa = [  
    "nome" => "João",  
    "idade" => 20,  
    "cidade" => "Porto Alegre"  
];
```

```
$pessoa = array(  
    "nome" => "Maria",  
    "idade" => 25,  
    "cidade" => "São Paulo"  
);
```


```
$pessoa["nome"] = "Carlos";  
$pessoa["idade"] = 30;  
$pessoa["cidade"] = "Rio de Janeiro";
```

Arrays multidimensionais

```
$usuario = [  
  "nome" => "Lucas",  
  "endereço" => [  
    "rua" => "Rua A",  
    "cidade" => "Curitiba"  
  ]  
];
```

Criando um Objeto JSON Simples

```
$app->get(  
    '/teste',  
    function (Request $request, Response $response): ResponseInterface {  
  
        $dados = [  
            "nome" => "João",  
            "idade" => 30,  
            "ativo" => true  
        ];  
  
        $json = json_encode($dados);  
  
        $response->getBody()->write($json);  
  
        return $response->withHeader('Content-Type', 'application/json');  
    }  
);
```



```
{  
    "nome": "João",  
    "idade": 30,  
    "ativo": true  
}
```

Estruturas mais elaboradas

```
<?php
$app->get(
    '/teste',
    function (Request $request, Response $response): ResponseInterface {
        $dados = [
            "nome" => "Maria",
            "email" => "maria@email.com",
            "cargo" => [
                "id" => 1,
                "nome" => "Técnico em Informática"
            ]
        ];

        $json = json_encode($dados);
        $response->getBody()->write($json);
        return $response->withHeader('Content-Type', 'application/json');
    }
);
```

```
{
  "nome": "Maria",
  "email": "maria@email.com",
  "cargo": {
    "id": 1,
    "nome": "Técnico em Informática"
  }
}
```

Observe que há uma hierarquia de objetos

Vetor de objetos JSON

```
$app->get(
    '/usuarios',
    function (Request $request, Response $response): ResponseInterface {
        $usuarios = [
            [
                "nome" => "Carlos",
                "idade" => 28
            ],
            [
                "nome" => "Ana",
                "idade" => 32
            ]
        ];
        $json = json_encode($usuarios);

        $response->getBody()->write($json);

        return $response->withHeader('Content-Type', 'application/json');
    }
);
```

```
[
  {
    "nome": "Carlos",
    "idade": 28
  },
  {
    "nome": "Ana",
    "idade": 32
  }
]
```

XML = eXtensible Markup Language

- Linguagem de marcação para estruturar documentos
- Muito usada em integrações entre sistemas

```
< Pessoa >  
  < nome > João </ nome >  
  < idade > 20 </ idade >  
</ Pessoa >
```

Array para XML

```
$app->get(
    '/teste',
    function (Request $request, Response $response): ResponseInterface {
        $dados = [
            "nome" => "João",
            "idade" => 30,
            "ativo" => true
        ];
        // Cria a raiz do XML
        $xml = new SimpleXMLElement('<peessoa/>');
        // Converte o array em XML
        foreach ($dados as $chave => $valor) {
            $xml->addChild($chave, $valor);
        }
        // Converte para string
        $xmlString = $xml->asXML();
        // Escreve no response
        $response->getBody()->write($xmlString);
        return $response->withHeader('Content-Type', 'application/xml');
    }
);
```

```
<peessoa>
  <nome>João</nome>
  <idade>30</idade>
  <ativo>1</ativo>
</peessoa>
```

Aplicações modernas.



Tipicamente em uma aplicação moderna o frontend que não utiliza o **get** ou **delete** envia um json com os dados para um controle que utiliza as classes necessárias para resolver o problema.

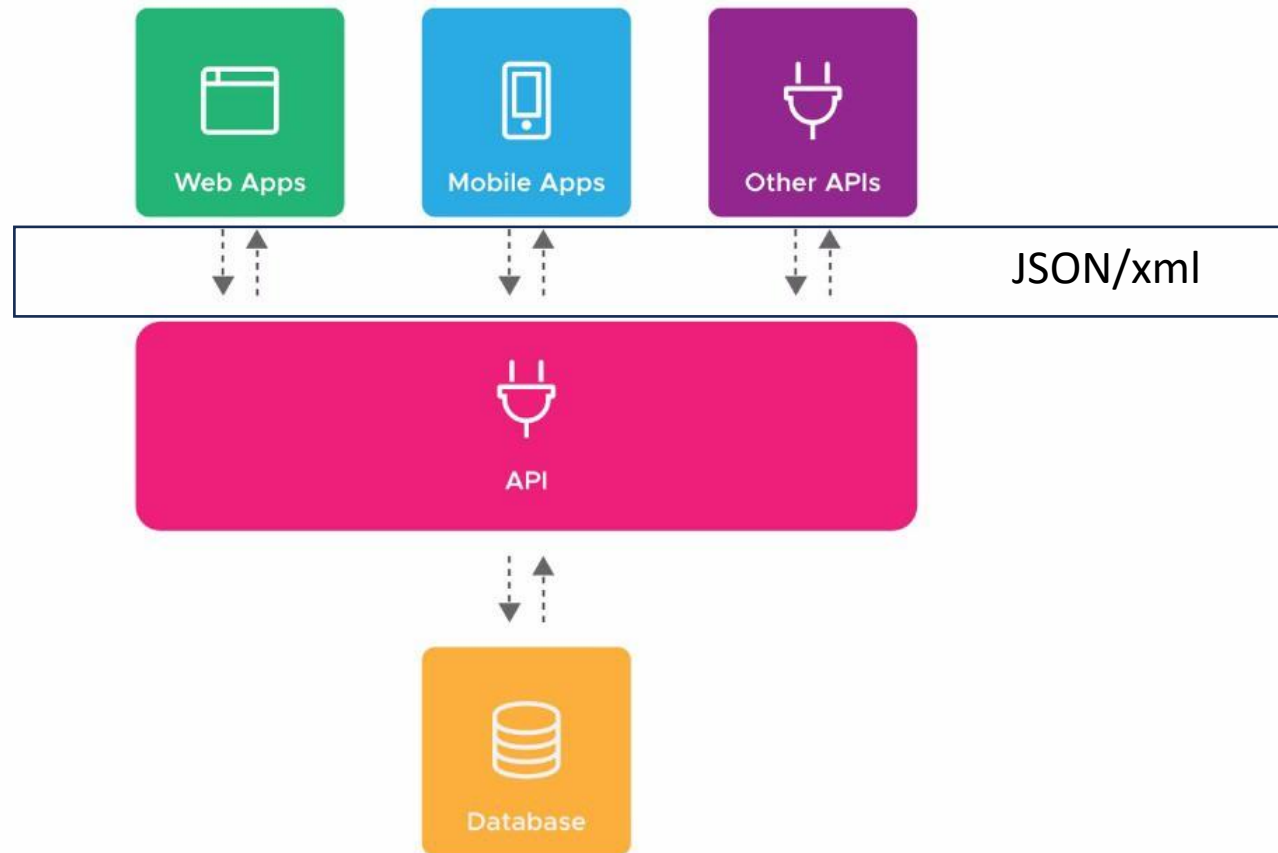


O controle também responde para o front-end um json.



Se o front trabalhar com o método get tipicamente os dados são passados na uri.

Comunicação da API



insomnia

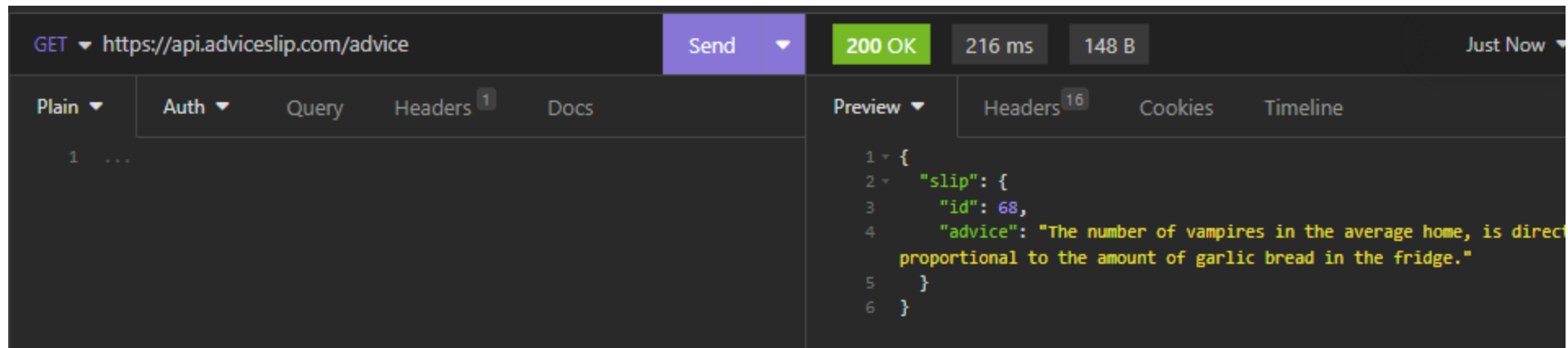
- É um software que nos permite fazer requisições à APIs.
- Ajuda no desenvolvimento do Backend (ex.: testar as chamadas para cada **endpoint** sem necessitar de um Frontend)
- Download free:
<https://insomnia.rest/>



Insomnia

Testando apis:

- Acesse a URL:
 - <https://api.adviceslip.com/advice>



```
GET https://api.adviceslip.com/advice 200 OK 216 ms 148 B Just Now
```

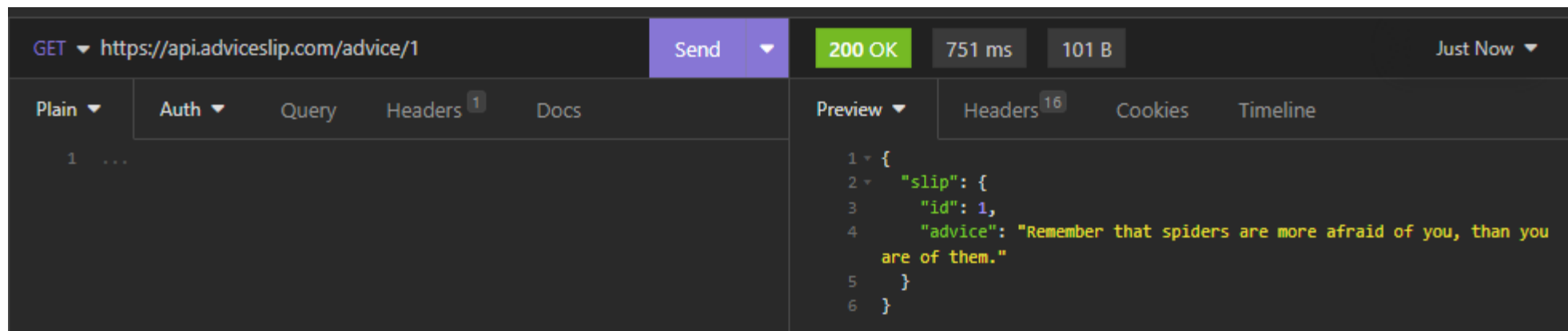
Plain Auth Query Headers 1 Docs

Preview Headers 16 Cookies Timeline

```
1 {
2   "slip": {
3     "id": 68,
4     "advice": "The number of vampires in the average home, is direct
5   }
6 }
```

https://api.adviceslip.com/advice/1

- Observe que é enviado o número 1 pela a url, é possível notar que o 1 é referente a uma chave primária em um banco de dados e que você pode escolher outros números correspondentes a conelhos diferentes.



```
GET https://api.adviceslip.com/advice/1 200 OK 751 ms 101 B Just Now
```

```
Plain Auth Query Headers 1 Docs Preview Headers 16 Cookies Timeline
```

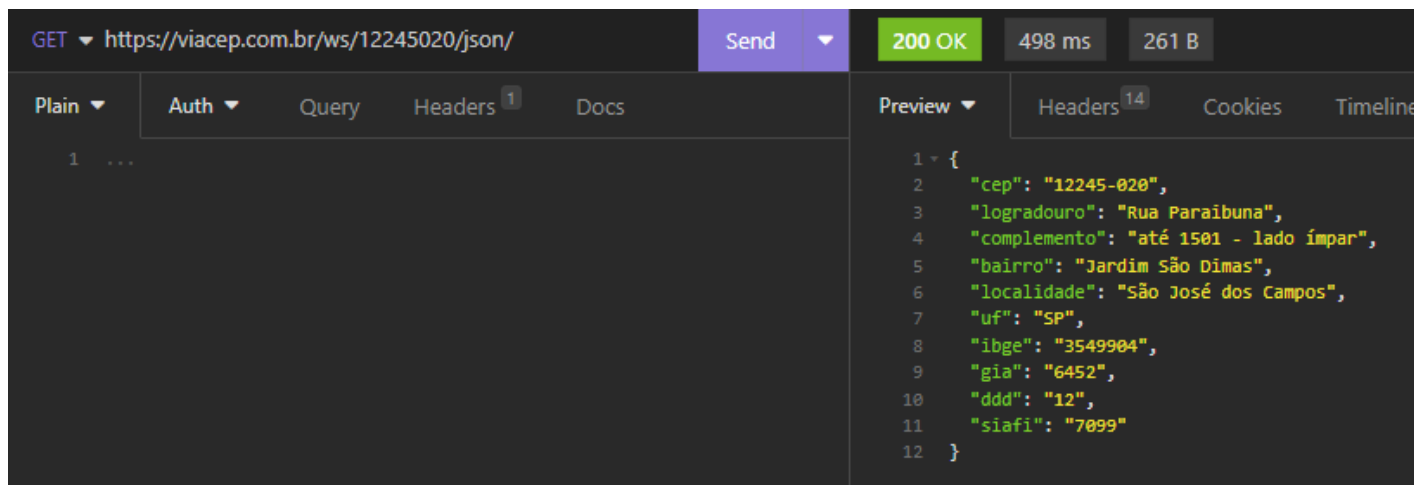
```
1 ...
```

```
1 {
2   "slip": {
3     "id": 1,
4     "advice": "Remember that spiders are more afraid of you, than you
5     are of them."
6   }
}
```

Via cep:

<https://viacep.com.br/ws/12245020/json/>

- Observe que a ferramenta envia um get para o endereço:
 - <https://viacep.com.br/ws/12245020/json/>
- Observe o formato da url, **não** é utilizado o padrão "? &" para identificar as variáveis e dados que serão enviados para o servidor.
- É possível notar que o cep é enviado na url.



```
GET https://viacep.com.br/ws/12245020/json/ 200 OK 498 ms 261 B
Plain Auth Query Headers 1 Docs Preview Headers 14 Cookies Timeline
1 ...
1 {
2   "cep": "12245-020",
3   "logradouro": "Rua Paraibuna",
4   "complemento": "até 1501 - lado ímpar",
5   "bairro": "Jardim São Dimas",
6   "localidade": "São José dos Campos",
7   "uf": "SP",
8   "ibge": "3549904",
9   "gia": "6452",
10  "ddd": "12",
11  "siafi": "7099"
12 }
```

apis interessantes.

- <https://deividfortuna.github.io/fipe/>
 - <https://parallelum.com.br/fipe/api/v1/carros/marcas>
 - <https://parallelum.com.br/fipe/api/v1/carros/marcas/59/modelos>

<https://parallelum.com.br/fipe/api/v1/carros/marcas>

The screenshot shows a REST client interface with the following components:

- Request Bar:** Method: GET, URL: `https://parallelum.com.br/fipe/api/v1/carros/marcas`, Status: 200 OK, Time: 311 ms, Size: 3.4 KB.
- Request Body:** Params, Body, Auth, Headers (3), Scripts, Docs.
- Response Body:** Preview, Headers (16), Cookies, Tests (0/0), Mock, Console.
- URL Preview:** `https://parallelum.com.br/fipe/api/v1/carros/marcas`
- Query Parameters:** Import from URL, Bulk Edit, Add, Delete all, Description.
- Response Data:** A JSON array of car brands with their codes and names.

```
398 {
399   "codigo": "56",
400   "nome": "Toyota"
401 },
402 {
403   "codigo": "57",
404   "nome": "Troller"
405 },
406 {
407   "codigo": "58",
408   "nome": "Volvo"
409 },
410 {
411   "codigo": "59",
412   "nome": "VW - VolksWagen"
413 },
```

<https://parallelum.com.br/fipe/api/v1/carros/marcas/59/modelos>

GET <https://parallelum.com.br/fipe/api/v1/carros/marcas/59/modelos> Send

200 OK 50 ms 35 KB Just Now

Params Body Auth Headers 3 Scripts Docs

Preview Headers 16 Cookies Tests 0/0 → Mock Console

URL PREVIEW

<https://parallelum.com.br/fipe/api/v1/carros/marcas/59/modelos>

QUERY PARAMETERS Import from URL Bulk Edit

+ Add Delete all Description

name	value
------	-------

Preview

```
1 {
2   "modelos": [
3     {
4       "codigo": 5585,
5       "nome": "AMAROK CD2.0 16V/S CD2.0 16V TDI 4x2 Die"
6     },
7     {
8       "codigo": 5586,
9       "nome": "AMAROK CD2.0 16V/S CD2.0 16V TDI 4x4 Die"
10    },
11    {
12     "codigo": 9895,
13     "nome": "AMAROK Comfor. 3.0 V6 TDI 4x4 Dies. Aut."
14    },

```