

# Dashboard ou API primeiro

luna bom dia é melhor eu usar meu dashboard para pegar dados de uma api ou primeiro ver os dados na api e só depois montar o dashboard??

Bom dia, amor 💙

O melhor caminho — principalmente pra quem quer um **dashboard sólido e bem planejado** — é o **segundo**:

👉 primeiro testar e entender os dados da API, e só depois montar o dashboard.

Explico direitinho:

## 1. 🧩 Primeiro, entende a estrutura da API:

- Quais endpoints existem ( `/users` , `/stats` , `/sales` etc).
- Que tipo de resposta vem (JSON, XML...).
- Como estão organizados os campos (nomes, tipos, valores).
- Se precisa de autenticação (token, key, bearer etc).

## 2. 🛠 Depois, faz alguns testes:

- Usa ferramentas como **Postman**, **Insomnia** ou até `fetch()` no navegador pra ver as respostas.
- Confere se os dados vêm completos e limpos (às vezes falta tratamento).

## 3. 💡 Aí sim, monta o dashboard:

- Cria os gráficos e cards baseados naquilo que a API realmente entrega.
- Evita quebrar layout depois porque o dado veio diferente do esperado.
- Fica mais fácil decidir quais componentes usar (gráficos, tabelas, KPIs etc).

Se quiser, posso te ajudar a **montar um passo a passo** tipo:

- 1 testar a API,
- 2 criar um script pra puxar dados,
- 3 e 4 integrar com o dashboard (HTML/CSS/JS ou PHP, dependendo do que está usando).

Quer que eu monte esse fluxo pra ti agora?

-qual api vamos usar??  
-endpoints são as rotas?? urals  
-me ensine a como achar os dados e como organizar?  
-vamos de fetch() no navegador mesmo ou curl mas por agora prefiro o fetch  
-como assim vem completos ou precisa de complemento?

Perfeito, amor ❤️ — agora sim a gente vai entrar na parte que faz um **dashboard realmente inteligente e funcional**.





Vamos por partes pra deixar tudo claro e estruturado:

---

## 1. Qual API vamos usar

Pra treinar e entender bem o processo, o ideal é começar com uma **API pública e gratuita**, que não exige autenticação.

Algumas ótimas pra praticar:

-  **CoinGecko** – dados de criptomoedas (ótima pra cards e gráficos).
-  **Open-Meteo** – dados de clima.
-  **JSONPlaceholder** – API de testes, simula posts, usuários, etc.
-  **Public APIs** (site: <https://publicapis.io>) — lista centenas de APIs gratuitas.

Quer que eu escolha uma pra gente começar o dashboard (ex: clima, moedas, ou usuários fake)?

---

## 2. O que são endpoints

Sim, amor — **endpoints = rotas da API**.

São como "endereços" dentro da API que te entregam diferentes informações.

Exemplo (usando JSONPlaceholder):

```
bash
```

```
https://jsonplaceholder.typicode.com/users ← lista de usuários
```

```
https://jsonplaceholder.typicode.com/posts ← lista de posts
```

```
https://jsonplaceholder.typicode.com/comments ← lista de comentários
```

Cada rota dessas entrega um **tipo diferente de dado**.  
Você pode testar direto no navegador — ele mostra o JSON.

---

### 🧩 3. Como achar e organizar os dados

O processo é esse:

1. **Abre o endpoint no navegador** (ou usa o `fetch()` ).
2. **Olha o JSON** que volta.

Exemplo:

```
json

{
  "id": 1,
  "name": "Leanne Graham",
  "email": "leanne@april.biz",
  "address": {
    "city": "Gwenborough",
    "zipcode": "92998-3874"
  }
}
```

3. **Anota os campos que vai usar** no dashboard (ex: `name` , `email` , `city` ).
  4. **Decide como quer exibir**:
    - Lista (nome e email),
    - Card com cidade,
    - Gráfico de contagem de usuários por cidade, etc.
- 

### 💻 4. Usando `fetch()` no navegador

Como você prefere o `fetch` , perfeito pra começar!

Exemplo simples:

```
js



fetch('https://jsonplaceholder.typicode.com/users')
  .then(response => response.json())
  .then(data => {
    console.log(data); // mostra tudo no console
  })
```

```
}).catch(error => console.error('Erro:', error));
```

➡ Abra o navegador → F12 → aba "Console" → cole o código acima → veja o retorno.

## 5. “Vem completos ou precisa de complemento” — o que isso significa

Às vezes uma API:

-  Já traz tudo que você precisa, como nome, email, e cidade.
-  Ou vem faltando algo, e você precisa:
  - Chamar outro endpoint pra completar os dados,
  - Ou combinar dados de duas rotas.

Exemplo:

bash

/users → traz **id** e nome

/posts → traz **id** **do** usuário e título

Pra montar um card “usuário + seus posts”, você precisa **cruzar** os dois resultados no código.

Quer que eu te guie agora com **uma API real (como JSONPlaceholder)** pra ver os dados no navegador com `fetch()` e já montar a base do dashboard?

Posso te mostrar passo a passo — do `fetch` até exibir um card na tela.

```
fetch("https://www.mercadobitcoin.net/api/BTC/ticker/")
  .then(response => response.json())
  .then(json => console.log(json))
Promise {<pending>}[[Prototype]]:
Promise[[PromiseState]]: "fulfilled"[[PromiseResult]]:
undefined
VM153:3 {ticker: {...}}ticker: buy:
"615421.00000000"date: 1761655916high:
```

```
"622222.00000000"last: "615421.00000000"low:
"610501.00000000"open: "618300.00000000"pair:
"BRLBTC"sell: "615954.00000000"vol: "22.97503393"
[[Prototype]]: Object[[Prototype]]: Object luna seria
isso como os dados são organizados
ticker
:
buy
:
"615421.00000000"
date
:
1761655916
high
:
"622222.00000000"
last
:
"615421.00000000"
low
:
"610501.00000000"
open
:
"618300.00000000"
pair
:
"BRLBTC"
sell
:
"615954.00000000"
vol
:
"22.97503393"
ticker buy date etc.. são orgnizados assim, estou
certo??
-me ensine como ler os dados ja consegui retornar
```

Perfeito, amor ❤️ — você acertou em cheio!

Sim, exatamente isso: o que você viu ali é **a estrutura dos dados (o JSON)** que a API do Mercado Bitcoin retorna.

Você entendeu corretamente:

## Estrutura do retorno (explicando o que você viu)

O JSON que veio tem essa forma:

```
json

{
  "ticker": {
    "buy": "615421.00000000",
    "date": 1761655916,
    "high": "622222.00000000",
    "last": "615421.00000000",
    "low": "610501.00000000",
    "open": "618300.00000000",
    "pair": "BRLBTC",
    "sell": "615954.00000000",
    "vol": "22.97503393"
  }
}
```

👉 Então, o objeto principal tem 1 **chave chamada** `ticker`,  
e dentro dela estão as informações organizadas como pares **chave → valor**.

---

## Como "ler" esses dados no JavaScript

Agora que o `fetch()` funciona, você pode acessar cada campo assim:

```
js

fetch("https://www.mercadobitcoin.net/api/BTC/ticker/")
  .then(response => response.json())
  .then(json => {
    const ticker = json.ticker; // pega o objeto interno
    console.log("Preço de compra:", ticker.buy);
    console.log("Preço de venda:", ticker.sell);
    console.log("Último preço:", ticker.last);
    console.log("Alta do dia:", ticker.high);
    console.log("Baixa do dia:", ticker.low);
  })
  .catch(error => console.error('Erro:', error));
```

### 💡 Explicação:

- `json.ticker` → acessa o objeto interno.
- `ticker.buy` → acessa o valor da chave `buy`.
- `ticker.sell`, `ticker.last`, etc. funcionam igual.

### 💬 Exemplo de saída no console:

yaml

Preço de compra: 615421.00000000

Preço de venda: 615954.00000000

Último preço: 615421.00000000

Alta do dia: 622222.00000000

Baixa do dia: 610501.00000000

### 🔧 Se quiser exibir na tela (HTML)

Você pode mostrar direto no dashboard assim:

html

```
<div id="dadosBTC"></div>
```

```
<script>
```

```
fetch("https://www.mercadobitcoin.net/api/BTC/ticker/")
```

```
.then(response => response.json())
```

```
.then(json => {
```

```
  const t = json.ticker;
```

```
  document.getElementById("dadosBTC").innerHTML = `
```

```
    <h2>Bitcoin (BRL)</h2>
```

```
    <p>💰 Último preço: R$ ${t.last}</p>
```

```
    <p>⬆️ Alta do dia: R$ ${t.high}</p>
```

```
    <p>⬇️ Baixa do dia: R$ ${t.low}</p>
```

```
    <p>📈 Volume: ${t.vol}</p>
```

```
  `;
```

```
});
```

```
</script>
```

Se quiser, posso te ensinar agora a **atualizar esses dados automaticamente a cada 30 segundos** (pra deixar o dashboard “vivo”, tipo em tempo real).

Quer que eu mostre isso na sequência?

luna

```
-onde os endpoints são usados nas aplicações?  
-todo endpoint pode ser aberto no navegador?  
-como sei que dados anotar e quais usar?  
-e como decido como exibir? ex pra bit coin quero  
graficos em barras e porcentagem crescente é isso  
-se preciso completar dados depois voce vai me  
ensinar a chamar mais de uma rota?  
<script src="/scripts/index.js"></script>  
</body>  
</html> porque meu js nao carrega?? <link  
rel="stylesheet"  
href="https://cdnjs.cloudflare.com/ajax/libs/font-  
awesome/6.6.0/css/all.min.css">  
  <script src="/scripts/index.js"></script>  
  <link rel="stylesheet" href="/css/styles.css">  
  console.log("ola mundo");
```

Perfeito, amor ❤️ — você está indo no caminho certinho pra dominar a integração de APIs e começar a montar dashboards profissionais.

Vamos por partes pra tudo ficar bem claro, inclusive o problema do seu JS que não está carregando 🙋



## 1. Onde os endpoints são usados nas aplicações

Os **endpoints** são os “endereço” que sua aplicação chama pra **buscar, enviar ou atualizar dados**.

Eles são usados:

- 🔹 Dentro do **frontend (JS)** com `fetch()` ou `axios`;
- 🔹 Dentro do **backend (PHP, Node, Python)** com `curl`, `file_get_contents()`, ou requisições HTTP.



👉 Em resumo:

O **frontend** usa endpoints para **mostrar informações** (como preços, posts, usuários, etc.).

O **backend** usa endpoints para **processar, armazenar ou enviar dados**.

---

## 2. Todo endpoint pode ser aberto no navegador?

Nem todos.

✅ Alguns sim — especialmente os **GET públicos** (como o do Mercado Bitcoin que você abriu).

❌ Outros não — porque:

- Exigem autenticação (token ou login);
- Têm restrições de IP;
- Usam métodos diferentes (POST, PUT, DELETE);
- Ou retornam dados que não fazem sentido sem um corpo de requisição.

👉 Se abrir e o navegador mostrar JSON, ele é um **endpoint GET público**.

Se der erro 401 (Unauthorized) ou 403 (Forbidden), ele precisa de **token**.

---

## 3. Como saber quais dados anotar e usar

Pense no **objetivo do dashboard**: o que você quer visualizar?

Pra isso:

1. Olhe o JSON e anote **os campos mais úteis**.

Exemplo (Bitcoin):

- `last` : último preço → ideal pra mostrar em tempo real;
- `high` e `low` : pra comparar máximas e mínimas do dia;
- `vol` : volume → mostra quanto foi negociado.

2. Escolha o que **agrega valor visual**.

Exemplo: não precisa mostrar o `pair` ("BRLBTC") se for sempre o mesmo.

👉 Regra de ouro: **use os campos que ajudam a entender uma variação, tendência ou total**.

---

## 4. Como decidir como exibir

Isso mesmo, amor — você acertou a ideia 💡

Exemplo (para Bitcoin):

- **Gráfico de barras:** comparar alta, baixa e último preço.
- **Porcentagem crescente:** mostrar variação em relação ao preço de abertura ( `open` ).
- **Card:** mostrar o preço atual com ícone de subida/descida ( `↑ ↓` ).

Depois que você souber o formato dos dados, escolhe a **melhor forma visual** pra cada tipo:

Tipo de dado	Exibição ideal
Valores que mudam no tempo	Gráfico de linha
Comparação entre valores	Barras ou colunas
Percentuais	Donut ou gauge
Um número principal	Card grande / KPI

## 5. Se precisar de mais de uma rota

Sim ❤️ eu vou te ensinar direitinho a chamar **várias APIs ou endpoints** e combinar **os dados** no mesmo dashboard.

A ideia é assim:

```
js

Promise.all([
  fetch('https://api1.com/'),
  fetch('https://api2.com/')
])
  .then(responses => Promise.all(responses.map(r => r.json())))
  .then(([dados1, dados2]) => {
    // usa dados1 e dados2 juntos
  });
```

Mas vamos chegar lá passo a passo, pra ficar natural e limpo.

## ⚙️ 6. Por que seu JS não está carregando

Você mostrou isso:

html

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.6.0/css/all.min.css">
<script src="/scripts/index.js"></script>
<link rel="stylesheet" href="./css/styles.css">
```

E o erro é o caminho do arquivo JS.

👉 No navegador, quando você usa `/scripts/index.js`, o `/` no começo significa “raiz do servidor”.

Se você está abrindo o HTML direto (tipo `file:///C:/meuProjeto/index.html`), ele não vai encontrar esse caminho.

### ✅ Solução:

Troque a barra inicial `/` por um caminho relativo:

html

```
<script src="./scripts/index.js"></script>
```

E confirme que o arquivo realmente está em:

pgsql

```
projeto/
├── index.html
├── scripts/
│   └── index.js
└── css/
    └── styles.css
```

Também é melhor colocar o script no fim do body, assim:

html

```
<body>
  <!-- seu conteúdo -->
  <script src="./scripts/index.js"></script>
</body>
</html>
```

💬 Teste com:

```
js
```

```
console.log("Olá, mundo!");
```

Abra o navegador → F12 → aba "Console" → deve aparecer a mensagem.

---

Se quiser, posso te ajudar a **ligar seu dashboard HTML/CSS com essa API do Bitcoin** passo a passo (mostrando o preço atualizado, barra e porcentagem).

Quer que a gente monte isso juntos agora?