**Entendendo e Praticando APIs com JavaScript**

**Projeto Conectados**

Vídeo 1 - **O que é uma API REST? Entenda tudo sobre API, HTTP e RESTful**

**Application Programming Interface**

***(Interface de Programação de Aplicações)***

**Uma API é um conjunto de regras e definições que permite que diferentes softwares "conversem" entre si.**

**Ela define como você pode interagir com um software, quais dados pode pedir e como deve pedir.**

**Tipos de API**

## **API Pública**

### O que é:

É uma API aberta de dados abertos para qualquer desenvolvedor usar, geralmente sem custo

Exemplos:

* **ViaCEP** – retorna endereço a partir do CEP.
* **API do IBGE** – dados populacionais, geográficos, etc.
* **OpenWeather** – previsão do tempo.

Usada quando você quer consumir dados ou serviços públicos em seu app/site sem restrições corporativas.

## **API Privada**

É uma API interna, usada somente dentro de uma empresa ou sistema. Não é acessível ao público.

* Controlada por uma organização.
* Requer autenticação interna.

### Exemplos:

* A API do sistema de estoque que só o setor de vendas da empresa pode acessar.

Usada quando você quer integrar sistemas internamente, com controle total e sem expor nada para o público.

## **API de Terceiros**

É uma API criada e mantida por outra empresa, mas que você pode usar em seus próprios projetos. Pode ser pública ou paga.

* Você depende de outro fornecedor.
* Normalmente exige registro, autenticação e integração bem definida.
* Pode ter custos ou planos escaláveis.

Exemplos:

* **Google Maps API** – mapas, geolocalização.
* **Mercado Pago API** – pagamentos.
* **Stripe API** – cobranças com cartão.
* **Correios API** (SIGEP) – cálculo e rastreio de frete (só com contrato).

Usada quando você quer incluir funcionalidades complexas no seu sistema (pagamento, login social, entrega, etc.) sem ter que desenvolver tudo do zero

**Comunicação com API**

**É o processo de enviar dados para uma API e receber uma resposta dela, geralmente pela internet, usando o protocolo HTTP (o mesmo usado para acessar sites).**

**Essa comunicação acontece através de requisições e respostas.**

**Verbos de HTTP**

**Os verbos HTTP (também chamados de métodos HTTP) são ações usadas para dizer ao servidor o que você quer fazer com um recurso (como dados, arquivos, produtos, usuários, etc.).**

| **Método** | **O que faz** | **Exemplo** |
| --- | --- | --- |
| **GET** | Buscar dados | Buscar endereço por CEP |
| **POST** | Enviar/criar novos dados | Criar um novo pedido ou cadastro |
| **PUT** | Atualizar dados existentes | Atualizar todos os dados de um produto. |
| **DELETE** | Apagar dados | Cancelar um pedido ou excluir um item |
| **PATCH** | Atualiza parte dos dados de um recurso | Atualizar só o preço de um produto. |

## **O que é um *endpoint*?**

**Um endpoint é um endereço (URL) de uma API que representa um ponto de acesso para realizar uma ação específica.**

## **O que é REST?**

**REST é a sigla para Representational State Transfer (em português, Transferência de Estado Representacional).  
 É um estilo de arquitetura para criar APIs que usam o protocolo HTTP de forma simples, padronizada e eficiente.**

## **Ideia central do REST**

A ideia é organizar as comunicações entre cliente e servidor com base em:

* Recursos (ex: usuários, produtos, pedidos)
* Verbos HTTP (GET, POST, PUT, DELETE...)
* URLs limpas e padronizadas
* Respostas em JSON (geralmente)

**REST não é uma tecnologia, mas um conjunto de princípios para estruturar uma API de forma clara e previsível.**

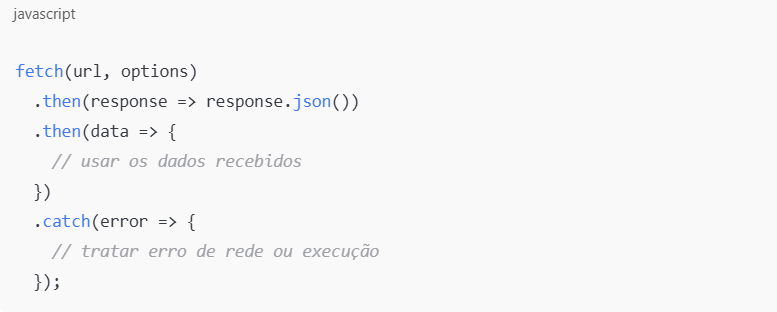
Vídeo 2 - **Como consumir de uma API usando a Fetch API! - Level Up #05**

## **O que é a Fetch API?**

**A Fetch API é uma forma moderna de fazer requisições HTTP no JavaScript. Ela serve para buscar ou enviar dados para um servidor, como por exemplo:**

* Buscar dados de uma API (ex: clima, CEP, produtos).
* Enviar um formulário para um servidor.
* Atualizar ou deletar um recurso via API.

**Estrutura geral**

****

* url: o endereço (endpoint) da API.
* options: objeto com configurações da requisição (método, cabeçalhos, corpo etc).
* then(response => response.json()): converte a resposta para JSON.
* catch(...): captura erros de rede ou sintaxe.

## 

## **Exemplo prático de GET (buscar dados)**



* response.ok: verifica se o código HTTP foi 200-299.
* response.json(): transforma a resposta em um objeto JavaScript.

**Exemplo de POST (enviar dados)**

**Usando async/await (forma moderna e mais clara)**

****

* await: espera o resultado da promessa (Promise) antes de seguir.
* try/catch: captura erros de forma mais limpa do que then/catch.