



C . E . S . A . R  
school

# NEXT

## Introdução ao HTTP e consumo de APIs com Fetch

Copyright © CESAR School 2024 | Todos os direitos reservados.

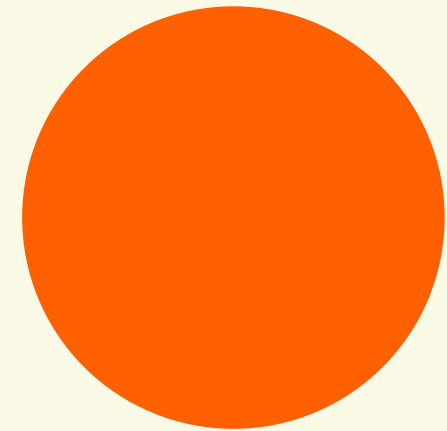
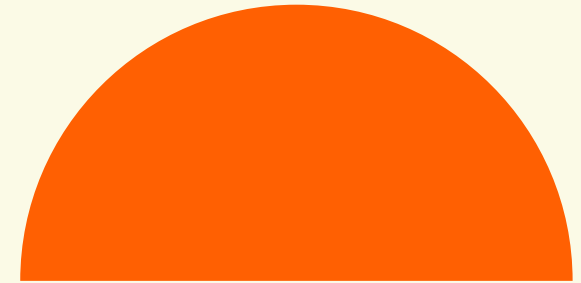
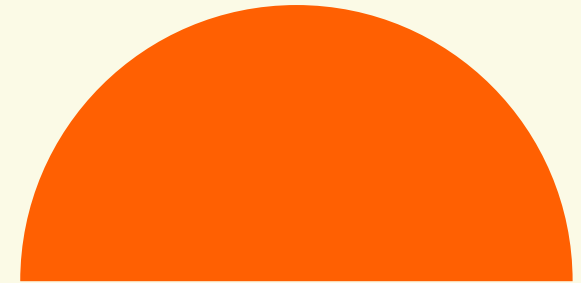
Este material, ou qualquer parte dele, não pode ser reproduzido, divulgado ou usado de forma alguma sem autorização escrita.



# Aula 10 - Introdução ao HTTP e consumo de APIs com Fetch

**Introdução ao HTTP e consumo de APIs  
com Fetch**

**Aula 10**

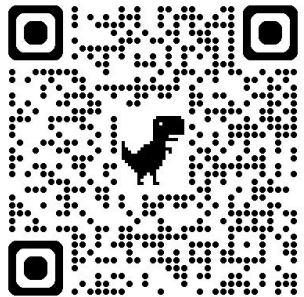


# Olá!

## Eu sou Lucas Marques

Estou aqui porque sou entusiasta de tecnologia e apaixonado por compartilhar conhecimento

Adoro facilitar o processo de ensino-aprendizagem, tornando conceitos complexos mais simples e acessíveis.



# O que veremos hoje?

- História do HTTP;
- Introdução ao HTTP (Componentes, Como Funciona);
- Estrutura de uma Requisição HTTP;
- Estrutura de uma Resposta HTTP;
- HTTP vs HTTPS;
- O que é uma API?;
- Consumo de APIs com fetch;



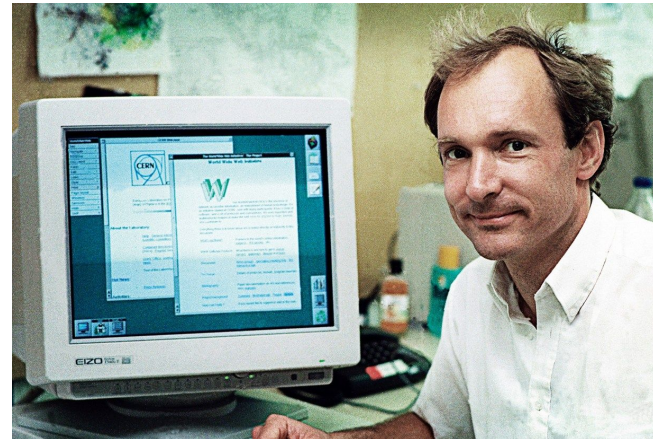


# História do HTTP

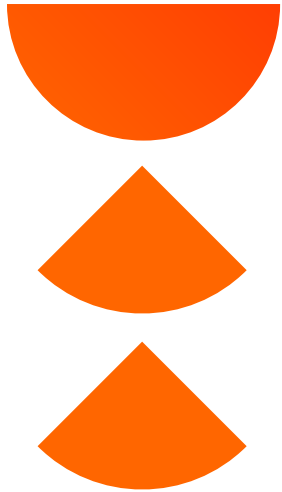


# História do HTTP

- Nos anos 1980, o cientista britânico Tim Berners-Lee, enquanto trabalhava no CERN (Organização Europeia para Pesquisa Nuclear), começou a idealizar formas de facilitar o compartilhamento de informações entre cientistas e instituições ao redor do mundo.



- Em 1989, ele teve a ideia de criar um sistema global que ajudasse nessa colaboração e chamou o projeto de **WorldWideWeb** (www). No ano seguinte, em 1990, ele apresentou uma proposta que se tornaria a base de um sistema para gerenciar informações usando hipertexto.
- Essa proposta introduziu o **Protocolo de Transferência de Hipertexto (HTTP)**. A primeira versão, chamada de 0.9, era bem simples: permitia apenas solicitações do tipo GET, usadas para buscar páginas de um servidor. As respostas não tinham cabeçalhos, apenas o conteúdo da página em HTML.



## História do HTTP

- Com o passar do tempo, o HTTP foi evoluindo. Em 1996, surgiu o HTTP 1.1, que trouxe melhorias importantes. Essa versão adicionou **métodos novos, como o POST**, e **tornou possível manter uma mesma conexão ativa para enviar várias solicitações e respostas**. Isso **reduziu o tempo de espera** ao evitar a abertura de novas conexões para cada solicitação.
- Em 2015, foi lançada a versão 2.0 do HTTP, que trouxe mudanças significativas na forma como os dados eram transmitidos. A **inclusão de recursos como multiplexação, compactação de cabeçalhos e priorização de fluxo** tornou as conexões mais **rápidas e eficientes**, melhorando a experiência dos usuários ao navegar na internet.
- Já em 2018, o HTTP 3 foi lançado, trazendo ainda mais avanços. Essa nova versão **aumentou a eficiência e o desempenho no envio de informações**, mantendo as melhorias das versões anteriores e garantindo uma navegação mais fluida e moderna.



# Introdução ao HTTP







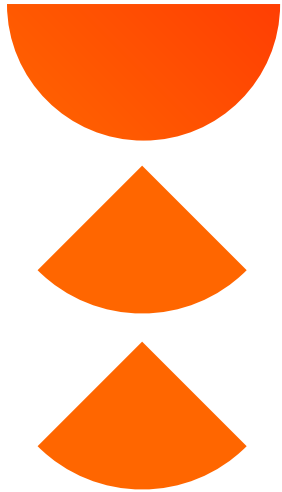
## O que é HTTP?

A sigla HTTP vem de Hypertext Transfer Protocol. Traduzido para o português, HTTP significa “Protocolo de Transferência de Hipertexto”.

É um protocolo de comunicação utilizado para a transferência de informações na World Wide Web (WWW) e em outros sistemas de rede.

O HTTP é a base para que o cliente e um servidor web **troquem** informações. Ele permite a requisição e a resposta de recursos, como imagens, arquivos e as próprias páginas webs que acessamos, por meio de mensagens padronizadas. Com ele, é possível que um estudante num café em São Paulo leia um artigo que está armazenado em um servidor no Japão.

- Três características importantes que precisamos saber sobre o HTTP são:
- Baseado em cliente-servidor.
  - Cada requisição é tratada de forma independente. Ele não "lembra" o que aconteceu antes.
  - Funciona por meio de troca de mensagens (requisição/resposta).



# Componentes do HTTP

Os componentes do HTTP são os elementos principais que participam do processo de comunicação no protocolo. Eles definem como o cliente e o servidor interagem para trocar informações. Vamos detalhar os principais componentes:

## CLIENTE:

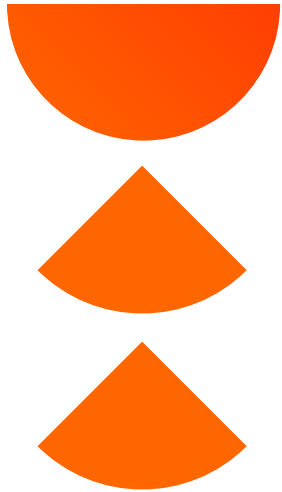
O cliente é quem inicia a comunicação no protocolo HTTP, enviando uma **requisição** ao servidor.

### Exemplos de clientes:

- Navegadores web (Google Chrome, Firefox, Safari)
- Aplicativos móveis
- Ferramentas como Postman ou Insomnia (HTTP Clients)

### Responsabilidades do cliente:

- Criar a requisição HTTP com o método (GET, POST, etc.), URL, cabeçalhos e, se necessário, corpo (body).
- Enviar uma requisição ao servidor por meio de uma conexão (geralmente TCP/IP).
- Interpretar a resposta HTTP recebida do servidor.



# Componentes do HTTP

Os componentes do HTTP são os elementos principais que participam do processo de comunicação no protocolo. Eles definem como o cliente / servidor interagem para trocar informações. Vamos detalhar os principais componentes:

## **SERVIDOR:**

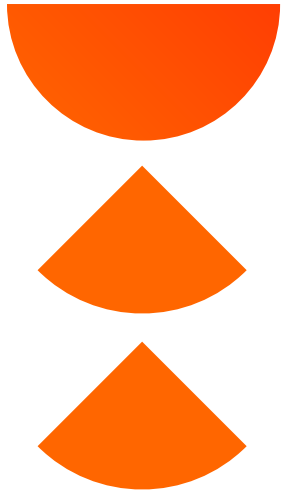
O servidor é quem recebe a requisição HTTP do cliente, processa a solicitação e devolve uma **resposta**.

### **Exemplos de servidores:**

- Servidores web como Apache, Nginx...
- APIs implementadas com frameworks como Express.js (Node.js), Django (Python), ou Spring Boot (Java).

### **Responsabilidades do servidor:**

- Analisar a requisição do cliente para identificar o que está sendo solicitado.
- Processar a requisição, que pode incluir acessar bancos de dados, arquivos ou executar lógica de negócio.
- Enviar uma resposta HTTP com os dados requisitados ou uma mensagem de erro.



# Componentes do HTTP

Os componentes do HTTP são os elementos principais que participam do processo de comunicação no protocolo. Eles definem como o cliente e o servidor interagem para trocar informações. Vamos detalhar os principais componentes:

## PROXIES (ou representantes):

Os proxies atuam como intermediários entre clientes e servidores, facilitando o fluxo de tráfego de dados entre eles.

### → Forward Proxy

- Fica posicionado entre o cliente e o servidor, agindo em nome dos clientes para buscar recursos dos servidores

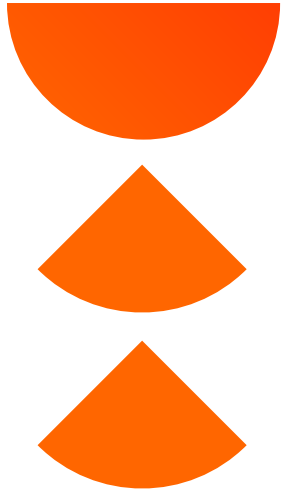
### → Reverse Proxy

- Fica posicionado entre clientes e servidores, mas operando em nome dos servidores para receber solicitações dos clientes.

Além de otimizar o tráfego e melhorar a eficiência da rede, muitos proxies implementam mecanismos de cache para armazenar localmente recursos frequentemente solicitados, reduzindo a carga nos servidores.

Esses intermediários também oferecem benefícios em termos de segurança, permitindo anonimato na web, filtragem de conteúdo malicioso e controle de acesso.

Adicionalmente, proxies podem realizar balanceamento de carga entre servidores, acelerar o carregamento de páginas por meio da compressão de dados, e fornecer logging e monitoramento para análise de padrões de uso e garantia de conformidade com políticas de segurança.



## Como funciona?

O HTTP opera em um modelo cliente-servidor, em que um cliente, geralmente um navegador web, faz solicitações a um servidor para obter recursos, como páginas da web, imagens ou arquivos.

O ciclo se inicia quando:

1. O cliente estabelece contato com o servidor, encaminhando uma requisição HTTP;
2. Nessa solicitação, o cliente especifica o método pretendido (por exemplo, GET) e o caminho do recurso desejado;
3. Ao receber essa requisição, o servidor a processa e responde com uma mensagem HTTP, incluindo o recurso requisitado e informações adicionais no cabeçalho da resposta.

## Desafio: O que significa a sigla HTTP?

- a) Hypertext Transfer Protocol
- b) Hyperlink Text Protocol
- c) High Transfer Protocol
- d) Hyper Transfer Protocol



# Desafio: O que significa a sigla HTTP?

a) Hypertext Transfer Protocol

b) Hyperlink Text Protocol

c) High Transfer Protocol

d) Hyper Transfer Protocol





## Desafio: Qual das seguintes afirmações sobre o HTTP está correta?

- a) O HTTP é um protocolo usado apenas para transferência de arquivos.
- b) O HTTP é baseado em um modelo cliente-servidor e permite a troca de mensagens entre eles.
- c) O HTTP é um protocolo que só funciona em redes locais.
- d) O HTTP não permite a transferência de imagens ou páginas web.





## Desafio: Qual das seguintes afirmações sobre o HTTP está correta?

- a) O HTTP é um protocolo usado apenas para transferência de arquivos.
- b) O HTTP é baseado em um modelo cliente-servidor e permite a troca de mensagens entre eles.**
- c) O HTTP é um protocolo que só funciona em redes locais.
- d) O HTTP não permite a transferência de imagens ou páginas web.



# Estrutura de uma Requisição HTTP





# Método HTTP

O método HTTP define a ação a ser executada no recurso do servidor.

[ 🚨 **Importante!** ] Os métodos mais comuns são:

- **GET**: solicita um recurso do servidor (página, imagem, etc.).
- **POST**: envia dados ao servidor para criar ou atualizar um recurso.
- **PUT**: atualiza um recurso existente no servidor
- **DELETE**: apaga um recurso no servidor.



# URL (Uniform Resource Locator)

A URL é o **endereço completo** do recurso que está sendo solicitado.

A URL inclui o **protocolo** (http/https), o **domínio**, o **caminho do recurso** e, opcionalmente, parâmetros de consulta (query string).

## Exemplo de URL:

<https://api.exemplo.com/produtos?categoria=eletronicos>

- Protocolo: **https://** – Indica que a comunicação será feita de forma segura.
- Domínio: **api.exemplo.com** – O servidor que está sendo acessado.
- Caminho: **/produtos** – O recurso desejado.
- Query String: **?categoria=eletronicos** – Parâmetro passado para filtrar os dados.

# Desafio URLs 1: Identificar os componentes da URL

Dada a seguinte URL:

<https://www.site.com/loja/produtos?categoria=roupas&cor=azul>

Responda as perguntas:

1. Qual é o protocolo utilizado?
2. Qual é o domínio da URL?
3. Qual é o caminho do recurso?
4. Quais são os parâmetros da query string e seus valores?



## Desafio URLs 2: Construção de uma URL

2. Construa uma URL completa para acessar uma página de detalhes de um produto com o seguinte:

1. Protocolo?
2. Domínio?
3. Caminho?
4. Parâmetro?





## Desafio URLs 3: Modificação de uma URL

Dada a seguinte URL:

<https://www.exemplo.com/cursos?categoria=tecnologia&nivel=iniciantes>

Responda as perguntas:

1. Modifique a URL para adicionar um novo parâmetro preco=baixo.
2. Agora, modifique a URL para que o parâmetro nivel passe a ser avançado.



## Desafio URLs 4: Análise de URL

Dada a seguinte URL:

<ftp://ftp.servidor.com/arquivos/imagens/logo.png>

Responda as perguntas:

1. Qual protocolo está sendo utilizado?
2. O que é o "ftp" nesse caso?
3. Qual é o caminho do recurso?
4. Existe alguma query string na URL?







# Cabeçalhos (Headers)

Os cabeçalhos fornecem informações adicionais sobre a requisição.

Informações que podem aparecer nos headers incluem:

- **ContentType**: indica o tipo de dado que está sendo enviado (ex: application/json).
- **Authorization**: usado para fornecer tokens ou credenciais de autenticação.
- **User-Agent**: informa qual o navegador ou cliente está fazendo a requisição.
- **MUITOS OUTROS**

Exemplo de header em uma requisição:

```
1 {  
2   "Content-Type": "application/json",  
3   "Authorization": "Bearer abc123xyz456token",  
4   "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"  
5 }  
6
```



## Corpo (Body)

Nem todas as requisições HTTP têm um corpo, mas quando há, ele contém os dados enviados para o servidor.

Geralmente, é o caso das requisições **POST** e **PUT** onde o corpo contém dados em formatos como JSON, XML, ou form-data.

Exemplo de body em uma requisição **POST**:



```
1 {  
2   "nome": "Teclado Locipech",  
3   "preco": 99.90  
4 }  
5
```

**Desafio: Qual dos seguintes métodos HTTP é usado para enviar dados ao servidor, como em um formulário de cadastro?**

- a) GET
- b) PUT
- c) POST
- d) DELETE



**Desafio: Qual dos seguintes métodos HTTP é usado para enviar dados ao servidor, como em um formulário de cadastro?**

- a) GET
- b) PUT
- c) POST**
- d) DELETE



## Desafio: O que é uma URL em uma requisição HTTP?

- a) Um código de erro que indica problemas na requisição.
- b) O endereço completo do recurso que está sendo solicitado, incluindo protocolo, domínio e caminho.
- c) Um cabeçalho que define o tipo de conteúdo da requisição.
- d) O corpo da mensagem enviada ao servidor.





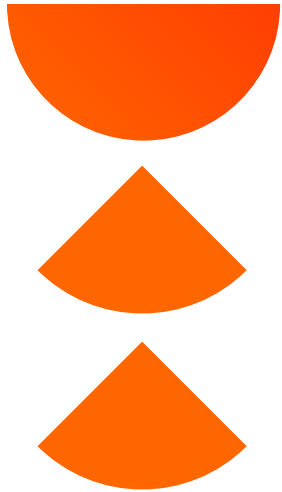
## Desafio: O que é uma URL em uma requisição HTTP?

- a) Um código de erro que indica problemas na requisição.
- b) O endereço completo do recurso que está sendo solicitado, incluindo protocolo, domínio e caminho.**
- c) Um cabeçalho que define o tipo de conteúdo da requisição.
- d) O corpo da mensagem enviada ao servidor.



# Estrutura de uma Resposta HTTP





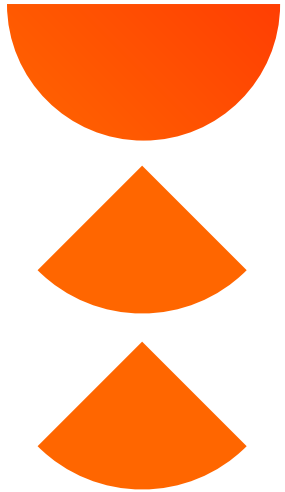
## Código de Status (Status Code)

O código de status é um número de 3 dígitos que indica o resultado da requisição. Ele é uma das primeiras informações na resposta HTTP.

Alguns dos códigos mais comuns são:

- **200 OK:** a requisição foi bem-sucedida e o recurso foi retornado.
- **201 Created:** a requisição foi bem-sucedida e um novo recurso foi criado.
- **400 Bad Request:** a requisição é inválida ou malformada.
- **401 Unauthorized:** o cliente não está autorizado a acessar o recurso.
- **404 Not Found:** o recurso solicitado não foi encontrado.
- **500 Internal Server Error:** o servidor encontrou um erro inesperado.





## Cabeçalhos (Headers)

Assim como na requisição, os cabeçalhos da resposta fornecem informações adicionais sobre o retorno do servidor, como:

- Cache-Control: Instruções sobre como o conteúdo deve ser armazenado em cache.
- Location: Usado em respostas de redirecionamento (por exemplo, após a criação de um recurso).

Exemplo de header em uma **response**:

```
1  {  
2    "Content-Type": "application/json",  
3    "Cache-Control": "no-cache, no-store, must-revalidate",  
4    "Location": "https://api.exemplo.com/produtos/123"  
5  }  
6
```



## Corpo (Body)

O corpo da **resposta** contém os dados solicitados ou informações sobre o resultado da requisição. Pode ser um JSON, HTML, XML ou outro tipo de conteúdo.

→ Em respostas GET, o corpo geralmente contém o recurso solicitado.

Exemplo de **body** de resposta com dados:

```
1  {
2    "id": 123,
3    "nome": "Monitor KELL",
4    "preco": 899.90
5  }
6
```

Em uma resposta de **erro**, o body pode conter uma mensagem explicativa:

```
1  {
2    "erro": "Produto não encontrado",
3    "codigo": 404
4  }
5
```

**Desafio: Qual código de status HTTP indica que o recurso solicitado não foi encontrado?**

- a) 200
- b) 404
- c) 500
- d) 301



**Desafio: Qual código de status HTTP indica que o recurso solicitado não foi encontrado?**

a) 200

**b) 404**

c) 500

d) 301



**Desafio: O que o corpo (body) de uma resposta HTTP geralmente contém em uma requisição GET bem-sucedida?**

- a) Uma mensagem de erro.
- b) Os dados do recurso solicitado, como HTML, JSON ou XML.
- c) Os cabeçalhos da requisição.
- d) O método HTTP utilizado.





**Desafio: O que o corpo (body) de uma resposta HTTP geralmente contém em uma requisição GET bem-sucedida?**

- a) Uma mensagem de erro.
- b) Os dados do recurso solicitado, como HTML, JSON ou XML.**
- c) Os cabeçalhos da requisição.
- d) O método HTTP utilizado.





# INTERVALO!





# HTTP vs HTTPS

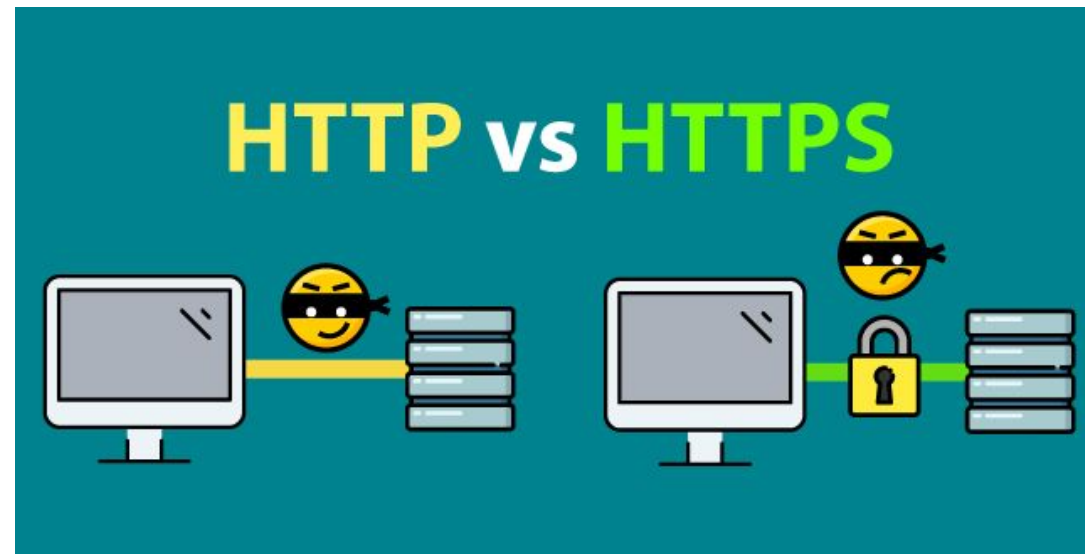




# HTTP vs HTTPS

Sabemos que o HTTP é um protocolo usado para comunicação entre cliente e servidor. Porém, você já deve ter notado que alguns sites utilizam HTTPS em vez de HTTP. Mas, afinal, o que os diferencia?

- A sigla HTTPS significa Hypertext Transfer Protocol **Secure**, ou Protocolo de Transferência de Hipertexto **Seguro**. Como o nome sugere, trata-se de uma versão **mais segura** do HTTP.
- Enquanto o HTTP transmite informações sem qualquer tipo de proteção, deixando os dados vulneráveis a possíveis ataques, o HTTPS utiliza **criptografia SSL/TLS**.
- Essa criptografia garante a proteção e a integridade dos dados, tornando-os **ilegíveis** caso a comunicação seja interceptada (Man-in-the-Middle).





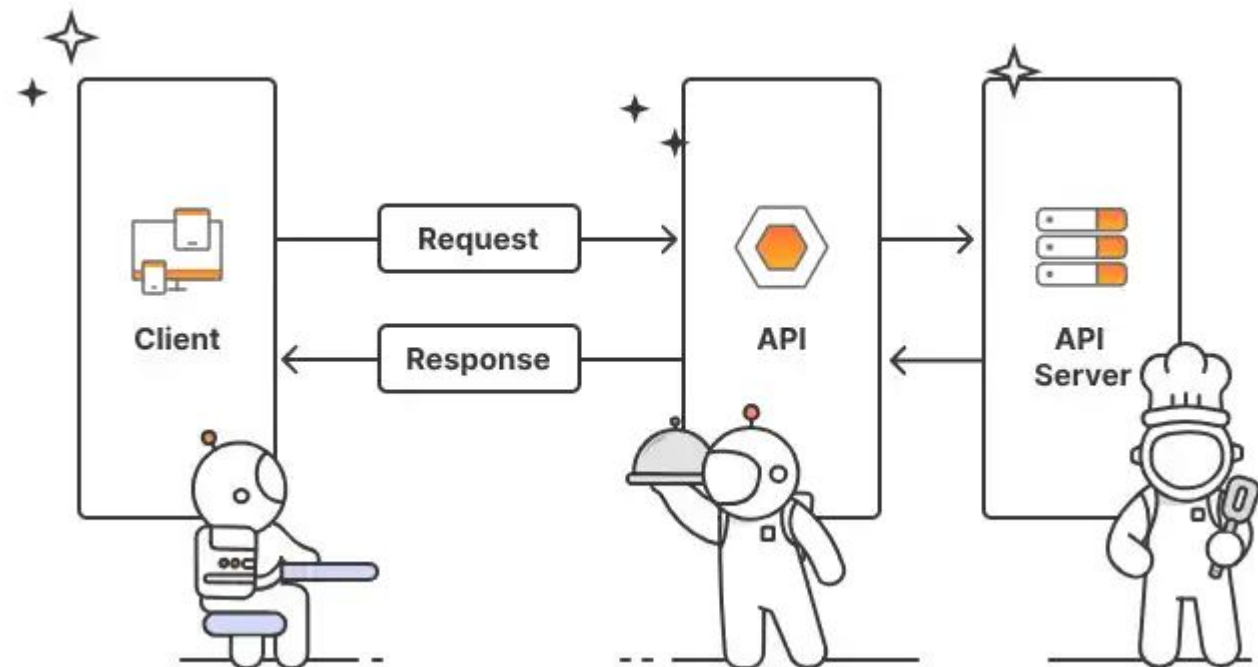
# O que é uma API?



# O que é uma API?

APIs (Application Programming Interface) são mecanismos que permitem que dois componentes se comuniquem usando um conjunto de definições e protocolos.

- Por exemplo, o sistema de software do instituto meteorológico contém dados meteorológicos diários.
- A aplicação para a previsão do tempo em seu telefone “fala” com esse sistema por meio de APIs e mostra atualizações meteorológicas diárias no telefone.



## Desafio: O que significa a sigla API?

- a) Application Programming Interface
- b) Advanced Programming Interface
- c) Application Protocol Interface
- d) Advanced Protocol Interface



# Desafio: O que significa a sigla API?

**a) Application Programming Interface**

b) Advanced Programming Interface

c) Application Protocol Interface

d) Advanced Protocol Interface





## Desafio: Qual das seguintes afirmações sobre APIs está correta?

- a) APIs são usadas apenas para comunicação entre navegadores e servidores.
- b) APIs permitem que dois componentes se comuniquem usando um conjunto de definições e protocolos.
- c) APIs são exclusivamente usadas para transferência de arquivos.
- d) APIs não podem ser usadas para acessar dados de terceiros.



## Desafio: Qual das seguintes afirmações sobre APIs está correta?

- a) APIs são usadas apenas para comunicação entre navegadores e servidores.
- b) APIs permitem que dois componentes se comuniquem usando um conjunto de definições e protocolos.**
- c) APIs são exclusivamente usadas para transferência de arquivos.
- d) APIs não podem ser usadas para acessar dados de terceiros.







# Consumo de APIs com fetch






## Consumo de APIs com fetch

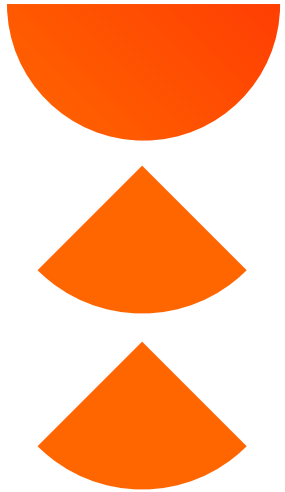
O fetch é uma API do JavaScript que permite fazer requisições HTTP de maneira fácil e assíncrona.

- Substitui o antigo XMLHttpRequest, oferecendo uma interface mais moderna e baseada em **Promises**.
- Promises são objetos que representam o resultado de uma operação assíncrona, podendo ser resolvida (sucesso) ou rejeitada (falha). Pensa assim: você pede algo e espera a resposta. Quando chega, pode ser boa ou ruim. 😊

Fazendo uma requisição **GET**:



```
1 fetch("https://api.exemplo.com/data")
2   .then((response) => response.json())
3   .then((data) => console.log(data))
4   .catch((error) => console.error("Erro:", error));
```



## Consumo de APIs com fetch

O **then** é usado para manipular o resultado bem-sucedido de uma Promise. Em outras palavras, ele será executado quando a requisição for completada sem erros. No caso de fetch, o then pode ser usado para:

- Processar a resposta recebida do servidor.
- Converter os dados para um formato utilizável, como JSON.
- Realizar ações com os dados obtidos (exibir na tela, armazenar, etc.).

O **catch** é usado para tratar erros que possam ocorrer durante a execução da Promise. Isso inclui:

- Problemas de rede (exemplo: sem conexão com a internet).
- Erros ao processar a resposta do servidor.
- Qualquer outro erro que ocorra no encadeamento da Promise.

Resumindo em termos simples:

**then**: serve para processar o resultado da Promise quando tudo dá certo.

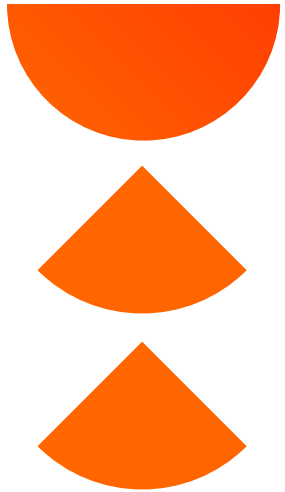
**catch**: serve para capturar e tratar erros caso algo dê errado.



# Consumo de APIs com fetch

Enviando dados com POST:

```
1 fetch("https://api.exemplo.com/data", {
2   method: "POST", // Define o método da requisição como POST.
3   headers: {
4     "Content-Type": "application/json", // Define o tipo de conteúdo enviado como JSON.
5   },
6   body: JSON.stringify({ name: "João" }), // Converte o objeto em string JSON para ser enviado no corpo da requisição.
7 })
8   .then((response) => response.json()) // Converte a resposta para um objeto JavaScript utilizando .json().
9   // Este método retorna uma Promise que será resolvida quando os dados forem processados.
10  .then((data) => console.log(data)) // Quando os dados são recebidos e convertidos, são exibidos no console.
11  .catch((error) => console.error("Erro:", error)); // Caso ocorra algum erro durante o processo, ele será capturado aqui e exibido no console.
12
```



## Consumo de APIs com fetch

Uma **async function** é uma função no JavaScript que permite trabalhar com operações assíncronas de forma mais simples e legível, usando as palavras-chave **async** e **await**. Essas funções sempre retornam uma **Promise**, mesmo que você não declare explicitamente.

- **await**: só pode ser usada dentro de uma async function.
  - Faz com que a execução da função seja pausada até que a **Promise** que está sendo aguardada seja resolvida.
  - Isso permite trabalhar com operações assíncronas como se fossem síncronas.

# Consumo de APIs com fetch

Uso de `async/await`:

```
1 // Define uma função assíncrona para buscar os dados.
2 async function fetchData() {
3   try {
4     // Tenta fazer a requisição HTTP para a URL 'https://api.exemplo.com/data' usando fetch.
5     const response = await fetch("https://api.exemplo.com/data");
6     // O uso de 'await' faz com que a execução da função pause até que a Promise do fetch seja resolvida.
7
8     // Verifica se a resposta é bem-sucedida (status no intervalo 200-299).
9     if (!response.ok) throw new Error(`Erro: ${response.status}`);
10    // Se o status da resposta não for OK, lança um erro com o status HTTP.
11
12    // Converte o corpo da resposta para JSON.
13    const data = await response.json();
14    // O 'await' aqui garante que a conversão seja concluída antes de continuar.
15
16    // Exibe os dados recebidos no console.
17    console.log(data);
18  } catch (error) {
19    // Captura e trata erros que possam ocorrer durante o fetch ou a conversão do JSON.
20    console.error(error);
21  }
22 }
23
```



# MONITORIA!



## Desafio: Buscar dados de um usuário

Use a API pública de usuários fictícios da **JSONPlaceholder** para buscar os dados de um usuário específico.

1. Faça uma requisição para o endpoint:  
<https://jsonplaceholder.typicode.com/users/1>
2. Mostre o nome do usuário no console.



## Desafio: Listar títulos de posts

Use a API de posts da **JSONPlaceholder** para listar os títulos dos 5 primeiros posts.

1. Faça uma requisição para <https://jsonplaceholder.typicode.com/posts>
2. Filtre os 5 primeiros posts e mostre seus títulos no console.



## Desafio: Buscar uma imagem aleatória

Use a API pública de imagens aleatórias <https://picsum.photos/v2/list> para buscar a URL de uma imagem.

1. Faça a requisição para obter uma lista de imagens.
2. Exiba a URL da primeira imagem no console.





## Desafio: Listar comentários de um post

Use a API de comentários da **JSONPlaceholder** para buscar os comentários de um post específico.

1. Faça uma requisição para <https://jsonplaceholder.typicode.com/comments?postId=1>
2. Exiba os e-mails dos usuários que comentaram no console.



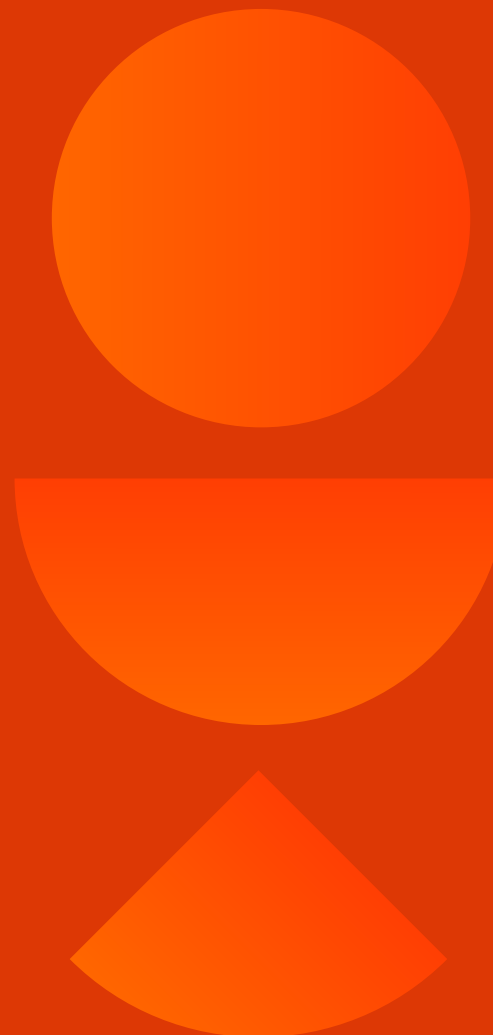
## Desafio: Mostrar uma frase aleatória

Use a API pública de citações aleatórias  
<https://jsonplaceholder.typicode.com/comments>  
para buscar um comentário aleatório.

1. Faça uma requisição para obter um comentário.
2. Mostre o comentário e o autor no console.



**DÚVIDAS?! :)**









C . E . S . A . R



C . E . S . A . R

school