# KORU


DESENVOLVE
programa de formação
e inclusão de talentos
em tecnologia
Grupo
Boticário
KORU

# JS NO BROWSER EVENTOS E WEB APIS)

# ONDE ESTAMOS?

- Vimos **HTML** para a estrutura (os elementos na página).
- Vimos **CSS** para a estilização (como os elementos se parecem).
- Começamos a ver **Javascript** para a interatividade e lógica.
- Na última aula, aprendemos a **selecionar e manipular elementos HTML usando o DOM** (Document Object Model).

# MAS...

Tudo bem, sabemos mudar o texto de um parágrafo ou a cor de um botão...

... mas como fazemos isso acontecer QUANDO o usuário clica no botão?

Precisamos reagir às ações do usuário e do navegador!

**Resposta: Eventos!**

# EVENTOS NO NAVEGADOR

Um **evento** é **algo que acontece** em uma página web.

Pode ser uma **ação do usuário**:

**Clicar** em um botão

**Digitar** em um campo de texto

**Mover** o mouse

**Enviar** um formulário

Pode ser algo que o **navegador faz**:

A página **terminou de carregar**

Uma imagem **falhou ao carregar**

O **tamanho da janela mudou**

# EVENTOS NO NAVEGADOR

O navegador possui um **sistema** para detectar e gerenciar esses eventos.

Quando um **evento acontece** (ex: um clique em um botão), o **navegador cria um Objeto Evento** com informações sobre o que aconteceu.

**Ele então "dispara" esse evento** para o elemento onde ele ocorreu.

**Nós podemos "ouvir"** esses eventos usando **Event Listeners.**

# EVENT LISTENERS

# EVENT LISTENERS

A forma mais comum e recomendada de lidar com eventos em Javascript é usando o método **addEventListener**.

Você "anexa" um ouvinte (listener) a um elemento HTML específico.

Quando o evento especificado acontece neste elemento, uma função (que você define) é executada.

# EVENT LISTENERS - SINTAXE

**element**: O elemento HTML que você quer "ouvir".

**eventName**: Uma string com o nome do evento que você quer capturar (ex: 'click', 'mouseover', 'submit').

**callbackFunction**: A função que será executada quando o evento acontecer.

# PRÁTICA 01. BOTÃO CLICADO MUDA O TEXTO!

# PRÁTICA 02. DIV MUDA DE COR COM O MOUSE!

# REMOVENDO UM EVENT LISTENER

Se você não precisa mais reagir a um evento em um elemento, é uma boa prática **remover o listener.**

Isso ajuda a evitar problemas de memória (memory leaks) em aplicações mais complexas.

Sintaxe similar a addEventListener.

Importante: A callbackFunction passada para removeEventListener deve ser a mesma referência da função que foi passada para addEventListener.  Por isso, geralmente usamos funções nomeadas quando planejamos remover listeners.

```
element.removeEventListener(eventName, callbackFunction);
```

# EVENTOS COMUNS

# EVENTOS COMUNS

**Eventos de Mouse:**

- *click*: O usuário clica em um elemento.
- *dblclick*: O usuário clica duas vezes.
- *mouseover*: O ponteiro do mouse entra no elemento.
- *mouseout*: O ponteiro do mouse sai do elemento.
- *mousemove*: O ponteiro do mouse se move sobre o elemento.

# EVENTOS COMUNS

**Eventos de Teclado:**

- *keydown*: Uma tecla é pressionada.
- *keyup*: Uma tecla é liberada.
- *keypress*: Uma tecla geradora de caractere é pressionada e liberada (alguns eventos de tecla moderna preferem keydown/keyup).

# EVENTOS COMUNS

**Eventos de Formulário e Input:**

- *submit*: Um formulário é enviado.
- *change*: O valor de um elemento de formulário (input, select, textarea) muda e o elemento perde o foco (blur).
- *input*: O valor de um input/textarea muda (dispara imediatamente conforme digita).
- *focus*: Um elemento recebe foco.
- blur: Um elemento perde foco.

# EVENTOS COMUNS

**Eventos de Documento / Janela**

- *DOMContentLoaded*: O documento HTML inicial foi completamente carregado e parseado, sem esperar por stylesheets, imagens, e subframes para finalizar o carregamento. (Ideal para começar a manipular o DOM).
- *load*: A página inteira (incluindo assets como imagens) terminou de carregar.
- *resize*: A janela do navegador é redimensionada.
- *scroll*: O usuário rola a página.

# OBJETO EVENTO

# OBJETO EVENTO

Quando um listener é disparado, o **navegador cria e passa um objeto como primeiro argumento para a sua função de callback**.

Este é o **Objeto Evento** (event).

Ele contém informações valiosas sobre o evento que ocorreu.

```
element.addEventListener('click', function(event) {
  console.log(event); // Veja o que tem dentro!
});
```

# OBJETO EVENTO

**event.target:**

Referências ao elemento HTML onde o evento originalmente ocorreu.

Muito útil, especialmente quando buscamos saber qual elemento disparou e o valor que aquele elemento contém (ex: texto de input)

**event.type:**

A string do tipo do evento (ex: 'click', 'submit').

# OBJETO EVENTO

**event.preventDefault()**

Alguns eventos têm um comportamento padrão no navegador.

Clicar em um link (<a>) navega para outra página.

Enviar um formulário (<form>) recarrega a página e envia os dados.

Você pode impedir esse comportamento padrão com event.preventDefault().

# WEB APIS E NODE APIS

# O QUE SÃO WEB APIS?

**Web APIs** são interfaces que o **navegador** oferece para o JavaScript interagir com:

A página web (HTML e CSS)

O próprio navegador (histórico, janela, etc.)

Recursos do dispositivo do usuário (com permissão!)

Pense nelas como "controles" ou "funcionalidades" que o navegador disponibiliza para o seu código JS usar.

**Elas NÃO são parte da linguagem JavaScript** em si, mas do ambiente onde o JS está rodando (o navegador).

# O QUE SÃO WEB APIS?

**Já conhecemos algumas Web APIs...**

**DOM API**: Interagir com o HTML e CSS da página.

Ex: document.getElementById(), element.style.color = 'red'

**Events API**: Reagir a ações do usuário (cliques, digitação, etc.).

Ex: element.addEventListener('click', callback)

**Fetch API**: Fazer requisições de rede (pedir dados para outros lugares).

Ex: fetch('https://...') .then(...)

# O QUE SÃO NODE APIS?

Da mesma forma que **Web APIs são** interfaces que o **navegador** oferece, **Node APIs** são interfaces que o Node oferece para o JavaScript interagir com:

- O sistema de arquivos do seu computador
- O sistema operacional
- Criação de servidores, etc.

**Da mesma forma, Elas NÃO são parte da linguagem JavaScript** em si, mas do ambiente onde o JS está rodando (Node.js).

# O QUE SÃO NODE APIS?

Onde seu código JavaScript está rodando define quais APIs estão disponíveis:

**Browser (Navegador):**

Ambiente Front-end.

Acesso a Web APIs (interação com a página, a janela do navegador, etc.).

Não tem acesso direto ao sistema de arquivos, rede do servidor, etc.

**Node.js (Servidor):**

Ambiente Back-end ou CLI (Command Line Interface).

Acesso a Node APIs (interação com o sistema operacional, sistema de arquivos, rede, etc.).

Não tem acesso direto ao DOM, `window`, `document`, etc

# WEB APIS

# WEB APIS

**O que já vimos**

- DOM API: Manipular a estrutura HTML.
- Events API: Reagir a ações (o que acabamos de ver!).
- Fetch API: Fazer requisições HTTP (visto em JS Assíncrono).

**Outras Web APIs úteis:**

- Timers: `setTimeout()`, `setInterval()` (executar código depois de um tempo).
- Web Storage API: `localStorage`, `sessionStorage` (armazenar pequenos dados no navegador - veremos em detalhe na próxima aula!).
- Geolocation API, Canvas API, Web Audio API, etc.

# WEB APIS

**Existem Muitas!**

https://developer.mozilla.org/pt-BR/docs/Web/API

## APIs da Web

Ao escrever código para a Web, há muitas APIs da Web disponíveis. Abaixo está uma lista de todas as APIs e interfaces (tipos de objeto) que você pode usar ao desenvolver seu aplicativo ou site da Web.

As APIs da Web são normalmente usadas com JavaScript, embora isso nem sempre tenha que ser o caso.

### Especificações

Esta é uma lista de todas as APIs disponíveis.

A

- Attribution Reporting API (inglês)
- API de Dispositivos de Saída de Áudio

B

- Background Fetch API
- Background Synchronization API (inglês)
- Background Tasks API (inglês)
- Badging API (inglês)
- Barcode Detection API (inglês)
- Battery Status API
- Beacon API
- Web Bluetooth API (inglês)
- Broadcast Channel API (inglês)

C

- CSS Custom Highlight API (inglês)
- CSS Font Loading API (inglês)
- CSS Painting API (inglês)
- CSS Properties and Values API (inglês)
- CSS Typed Object Model API (inglês)
- CSS Object Model
- Canvas
- Channel Messaging API (inglês)
- Clipboard API (inglês)
- Compression Streams API (inglês)
- Compute Pressure API (inglês)
- Console API
- Contact Picker API (inglês)
- Content Index API (inglês)
- Cookie Store API (inglês)
- Credential Management API (inglês)

D

- Modelo de Objeto de Documento (DOM)
- Device Memory API (inglês)
- Device orientation events (inglês)
- Device Posture API (inglês)
- Document Picture-in-Picture API (inglês)

E

- EditContext API (inglês)
- Encoding API
- Encrypted Media Extensions API (inglês)
- EyeDropper API (inglês)

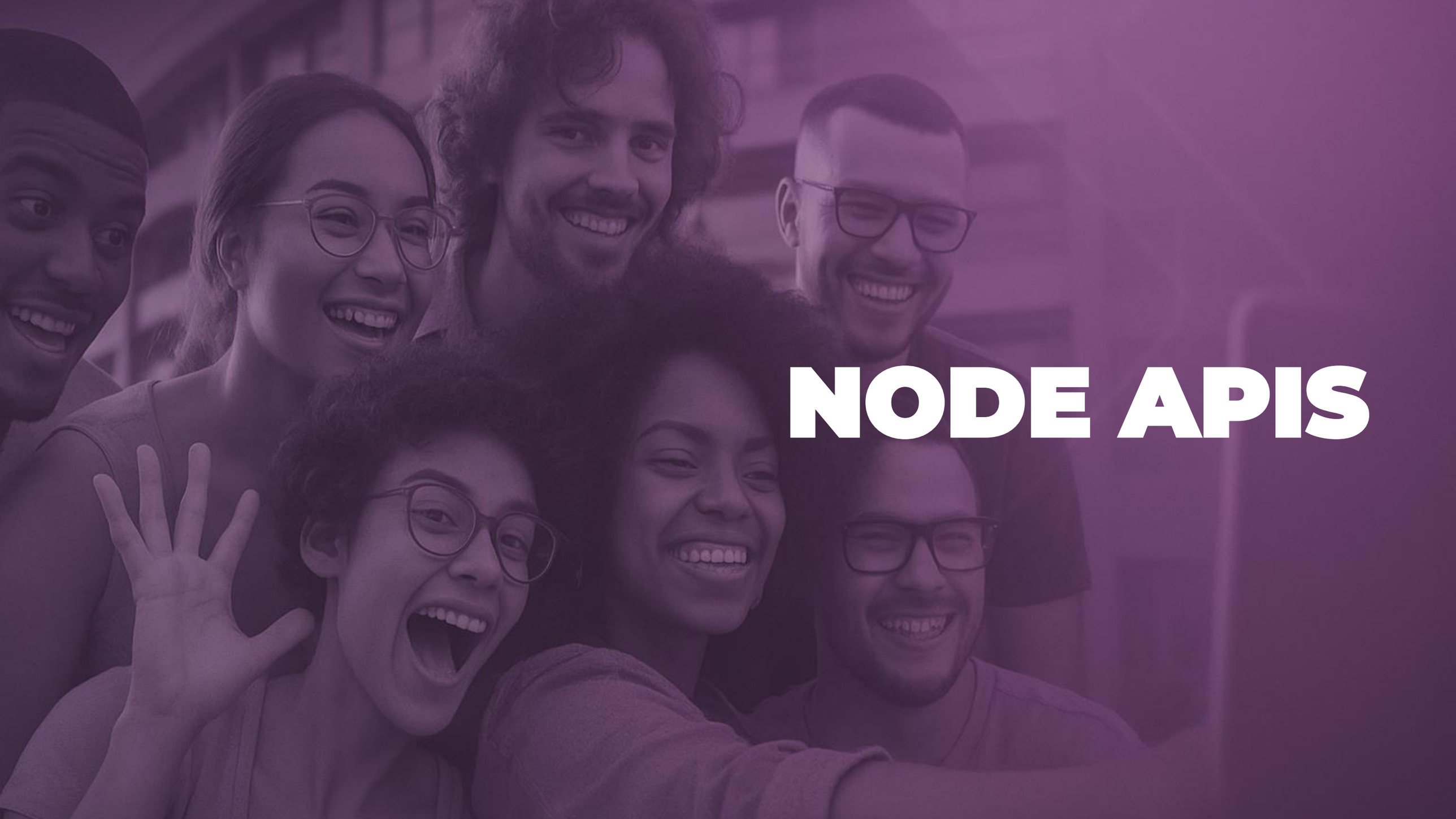
# PRÁTICA 03
## UMA PÁGINA QUE MOSTRA SUA WEBCAM

# PRÁTICA

## O que vamos fazer

Uma página web que exibe nossa Webcam! Para isso vamos usar a MediaDevices API (que é uma Web API).

Crie uma página que mostre a sua câmera.



# NODE APIS

# NODE APIS

APIs fornecidas pelo Node.js para interagir com o sistema operacional, sistema de arquivos, rede, etc.

Permitem que o JavaScript faça coisas que normalmente se faz em ambientes de servidor ou scripts de sistema.

Exemplos de **Módulos (APIs) Nativas do Node**:

- `**fs**` (File System): Ler, escrever, apagar arquivos.
- `**path**`: Manipular caminhos de arquivos e diretórios de forma consistente entre sistemas operacionais (Windows vs Linux/macOS).
- `**http**`: Criar servidores web.
- `**process**`: Acessar informações sobre o processo Node.js atual e o ambiente.

Estas APIs só funcionam no código que roda no Node.js.

# NODE APIS - MÓDULO FS

`**fs**` vem de File System. Permite interagir com arquivos e pastas.

Podemos ler, escrever, atualizar, deletar, verificar arquivos/pastas.

Geralmente usado com Promises para trabalhar de forma assíncrona (não bloquear a execução).

```
// Exemplo: Ler um arquivo
import { promises as fs } from 'fs'; // Usamos a versão com Promises

async function readFileExample(filePath) {
  try {
    const fileContent = await fs.readFile(filePath, 'utf-8');
    console.log('Conteúdo do arquivo:', fileContent);
  } catch (error) {
    console.error('Erro ao ler arquivo:', error);
  }
}

readFileExample('meuarquivo.txt');
```

# NODE APIS - MÓDULO PATH

O módulo `**path**` ajuda a lidar com caminhos de arquivos e diretórios.

Importante porque a forma de escrever caminhos varia entre sistemas operacionais (Windows usa `\` , Linux/macOS usa `/`).

Garante que seu código funcione em qualquer sistema.

```
// Exemplo: Juntar partes de um caminho
import path from 'path';

const directory = 'usuarios';
const fileName = 'dados.json';

// path.join() usa o separador correto para o sistema operacional
const fullPath = path.join(directory, fileName);

console.log('Caminho completo:', fullPath);
// No Windows pode mostrar: usuarios\dados.json
// No Linux/macOS pode mostrar: usuarios/dados.json
```

# RESUMO

# RESUMO

"Ouvir" eventos específicos em elementos HTML com `**addEventListener**()` e a parar de ouvir com `**removeEventListener**()`. O **objeto `event`** nos dá informações sobre o que aconteceu (`event.target`) e permite controlar o comportamento padrão (`event.preventDefault()`).

**APIs** - funcionalidades pré-construídas que o ambiente JavaScript nos oferece.

**Web APIs (Browser):** Ferramentas do navegador para interagir com a página e a janela. Exemplos: DOM, Eventos, Fetch, `localStorage`, `setTimeout`, `navigator.mediaDevices` (Webcam!).

**Node APIs (Node.js):** Ferramentas do Node.js (servidor ou linha de comando) para interagir com o sistema. Exemplos: Módulos `fs` (File System), `path`.

Onde seu código JavaScript roda (Navegador vs Node.js) determina quais APIs você pode usar e o que seu código pode fazer. Ambiente do Navegador ➡️ Web APIs | Ambiente Node.js ➡️ Node APIs

# KORU


DESENVOLVE
programa de formação
e inclusão de talentos
em tecnologia
Grupo
Boticário
KORU