

Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa

JavaScript

Módulo 2



POLitécnico
do Cávado
e do Ave



INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL



PRR
Plano de Recuperação
e Resiliência



REPÚBLICA
PORTUGUESA



Financiado pela
União Europeia
NextGenerationEU



IAPMEI
Parcerias para o Crescimento



PORTUGAL
DIGITAL

DOM – Document Object Model

Sessão 9



POLITÉCNICO
DO CÁVADO
E DO AVE



INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL



PRR
Plano de Recuperação
e Resiliência



REPÚBLICA
PORTUGUESA



Financiado pela
União Europeia
NextGenerationEU



IAPMEI
Parcerias para o Crescimento



PORTUGAL
DIGITAL

Objetivo geral

Capacitar os formandos na interação programática com a estrutura do documento, promovendo o domínio do DOM (Document Object Model) como uma interface dinâmica para o desenvolvimento de aplicações interativas. A sessão foca na transição do HTML estático para o State-Driven UI, ensinando a selecionar elementos com precisão cirúrgica e a manipular a árvore de nós de forma eficiente. Através da criação e remoção dinâmica de elementos e da gestão programática de estilos e conteúdos, os formandos aprenderão a atualizar a interface em tempo real, garantindo uma experiência de utilizador fluida sem a necessidade de recarregar a página.



Objetivos específicos

Ao final da sessão, os formandos deverão ser capazes de:

- Explicar o que é o **DOM** e como o navegador representa o HTML internamente;
- **Selecionar** elementos da página utilizando querySelector, querySelectorAll e getElementById;
- **Manipular** conteúdo e estilos de elementos DOM, distinguindo innerHTML de textContent e style de classList;
- **Criar, adicionar e remover** elementos HTML dinamicamente através de JavaScript.





DOM



POLitécnico
do Cávado
e do Ave



INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL



PRR
Plano de Recuperação
e Resiliência



REPÚBLICA
PORTUGUESA



Financiado pela
União Europeia
NextGenerationEU



IAPMEI
Parcerias para o Crescimento



PORTUGAL
DIGITAL



Como o navegador trabalha?

O navegador recebe um ficheiro HTML (texto) e transforma o HTML em uma estrutura em memória, essa estrutura chama-se DOM. O DOM, por sua vez, é uma representação do HTML em formato de objetos.



O objeto document

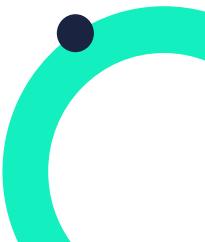
O objeto **document** é a propriedade do objeto **Window** que representa o conteúdo exibido em uma janela ou quadro do navegador. Ele é o objeto central do **Document Object Model (DOM)**, uma API fundamental que transforma documentos HTML ou XML estáticos em uma árvore de objetos interativa.



O que podemos fazer com o DOM?

O DOM permite que o JavaScript (JS) transforme documentos estáticos em aplicativos interativos, fornecendo o controle total sobre o conteúdo, a estrutura e a apresentação da página. Veja algumas ações que podem ser realizadas com o DOM:

- Seleção e Consulta de Elementos;
- Navegação e Travessia;
- Manipulação de Atributos;
- Modificação de Conteúdo;
- Interatividade com Eventos.





Seletores do DOM



Para manipular é preciso selecionar

O DOM contém muitos elementos antes de alterar algo o JS precisa saber exatamente qual elementos queremos manipular, para fazer isso, precisamos conhecer mais sobre os seletores.



Seletores HTML

Os seletores são padrões fundamentais utilizados para identificar e localizar elementos específicos dentro de um documento HTML ou XML para que possam ser manipulados via JavaScript. O DOM dessa forma define várias maneiras de selecionar elementos do HTML via JS, e são elas:

- Por Identificação (id): getElementById();
- Pelo nome (name): getElementsByName();
- Pelo Tipo ou Tag: getElementsByTagName();
- Pela Classe CSS getElementsByClassName();
- Por Seletores CSS: querySelector() e querySelectorAll().

getElementsByTagName

```
// Seleciona todos os elementos <p>
const paragrafos = document.getElementsByTagName('p');

// Use um ciclo for para aplicar o estilo a cada um
for (let i = 0; i < paragrafos.length; i++) {
    paragrafos[i].style.color = "blue";
    paragrafos[i].style.fontSize = "18px";
}
```

É possível obter todos os elementos de um determinado tipo (como todos os `<div>` ou ``) na ordem em que aparecem no documento.



getElementById

```
const titulo = document.getElementById("titulo");
```

```
function boasVindas() {  
    const nome = document.getElementById("campo-nome").value;  
    alert("Olá, " + nome + "!");  
}
```

Localiza o elemento
pelo seu identificador
único ID;



getElementsByName

```
// Seleciona todos os elementos com name="usuario"
const campos = document.getElementsByName("usuario");

// Altera o valor do primeiro elemento encontrado
campos[0].value = "João Silva";
```

Seleciona elementos que possuem um atributo name específico. E podemos manipular esse elemento de forma direta, através da posição desse elemento dentro do array.



getElementsByClassName

```
// Seleciona todos os elementos com a classe "destaque"  
const itens = document.getElementsByClassName("destaque");  
  
// Percorre a coleção para alterar a cor de fundo de cada um  
for (let i = 0; i < itens.length; i++) {  
    itens[i].style.backgroundColor = "yellow";  
}
```

Permite selecionar conjuntos de elementos que compartilham identificadores em seu atributo class



querySelector

```
const titulo = document.querySelector("h1");
```

```
document.querySelector(".card");
document.querySelector("#menu");
document.querySelector("ul li");
```

```
// O seletor de ID usa o cardinal (#)
const titulo = document.querySelector("#titulo-principal");
titulo.textContent = "Bem-vindo a 2026!";
```

Utilizam a poderosa sintaxe de seletores CSS para localizar elementos de forma precisa.

Ele retorna o primeiro elemento encontrado.





querySelectorAll

```
const itens = document.querySelectorAll("li");
```

```
itens.forEach(item => {  
  console.log(item.textContent);  
});
```

Retorna uma lista de elementos do tipo NodeList. É possível utilizar um loop for para acessar a lista de nodos encontrados.





Manipulação de Conteúdos e Estilos



InnerHTML

O innerHTML é uma propriedade que permite ler ou modificar o conteúdo HTML de um elemento de forma direta. Diferente de propriedades que tratam apenas o texto, o innerHTML interpreta as strings como código, permitindo a renderização de tags dentro do navegador.



InnerHTML - exemplo

```
// Selecionando o elemento pai
const container = document.querySelector('#meu-container');

// Injetando uma estrutura complexa de uma só vez
container.innerHTML =
`

## Novo Título



Este parágrafo foi criado dinamicamente.


<button class="btn">Clique aqui</button>
`;
```





innerHTML – por que ainda utilizar?

```
// Uso comum com Template Strings (ácento grave)
const produto = { nome: "Teclado", preco: 50 };

container.innerHTML =
  `<div class="card">
    <h3>${produto.nome}</h3>
    <p>Preço: ${produto.preco}€</p>
    <button onclick="comprar()">Comprar</button>
  </div>
`;
```





createElement - Criando elementos

```
const produto = { nome: "Teclado", preco: 50 };
const container = document.querySelector('#container');

// 1. Criar a Div principal
const card = document.createElement('div');
card.classList.add('card');

// 2. Criar o Título
const titulo = document.createElement('h3');
titulo.textContent = produto.nome; // Seguro: trata como texto puro

// 3. Criar o Parágrafo
const preco = document.createElement('p');
preco.textContent = `Preço: ${produto.preco}€`;

// 4. Criar o Botão
const botao = document.createElement('button');
botao.textContent = 'Comprar';
botao.addEventListener('click', comprar); // Forma moderna de add evento

// 5. Montar a hierarquia (Append)
card.appendChild(titulo);
card.appendChild(preco);
card.appendChild(botao);

// 6. Inserir no DOM
container.appendChild(card);
```



Alterar estilos com Style

O style irá aplicar estilos inline, o que

```
titulo.style.color = "red";  
titulo.style.fontSize = "24px";
```

gera o primeiro problema, contudo, é

uma técnica utilizada para mudanças

pontuais.



Alterar estilos com classList

```
// Muito mais limpo e respeita a cascata  
titulo.classList.add('titulo-destaque');
```

```
.titulo-destaque {  
    color: blue;  
    font-size: 20px;  
    margin-top: 10px;  
}
```

O **classList** exerce a mesma tarefa que o **style**, contudo, sem adicionar o estilo em linha, permitindo que o elemento seja alterado pela folha de estilo no futuro, caso necessário.



Criação Dinâmica de Elementos no DOM



HTML não precisa estar todo no ficheiro

Nem todos os elementos precisam existir no HTML inicialmente, pois o

JS pode:

- Criar;
- Inserir;
- Remover elementos da DOM.

Isso permite a criação de interface dinâmicas.

Criar um elemento: createElement

O elemento é criado em memória, o que significa que ele ainda não aparece na página.

O createElement É um método de fábrica do objeto Document utilizado para criar novos nós de elemento

```
const paragrafo = document.createElement("p");
```



Configurar o elemento criado

Podemos definir o texto, as

```
paragrafo.textContent = "Olá mundo!";  
paragrafo.classList.add("mensagem");
```

classes e os atributos desse
elemento de forma livre.



Inserir no DOM: appendChild

```
const container = document.querySelector("#container");
container.appendChild(paragrafo);
```

appendChild é um método da interface Node utilizado para inserir um nó na árvore do documento. A partir desse momento o elemento aparece na página.





Criar vários elementos

```
const lista = document.createElement("ul");

for (let i = 1; i <= 3; i++) {
  const item = document.createElement("li");
  item.textContent = `Item ${i}`;
  lista.appendChild(item);
}

document.body.appendChild(lista);
```

Podemos utilizar de laços de repetição para a criação de vários elementos.



Remover elementos

A remoção de elementos é tão simples quanto criar, como esse elemento foi criado a partir de uma constante, o próprio JS oferece um método simples para a remoção do elemento criado.

```
paragrafo.remove();
```



Síntese

- Utilização da DOM no JS;
- Manipulação de elementos HTML através dos seletores DOM;
- Manipulação de conteúdo através de métodos do DOM;
- Manipulação de estilo para a criação de páginas dinâmicas;
- Criação dinâmica de elementos.





Conclusão



Conclusão

“Um bom design é aquele que é o mais simples possível.”

Dieter Rams



Programação em JavaScript

ServiceNow - Deloitte

Rodrigo Costa