PERTEMUAN 4 | JAVASCRIPT DOM

4.1 Pengertian DOM

Document Object Model (DOM) adalah antarmuka yang merepresentasikan halaman HTML dalam bentuk pohon (tree). Setiap elemen HTML (seperti <html>, <body>, <h1>, <p>) dianggap sebagai node dalam pohon ini.

Tanpa DOM, JavaScript tidak akan bisa mengakses atau memodifikasi halaman web setelah halaman dimuat.

4.2 Kenapa DOM diperlukan?

DOM memiliki peran penting dalam pengembangan web modern, di antaranya:

- **Menghubungkan HTML, CSS, dan JavaScript**
  DOM menjadi jembatan yang menghubungkan struktur halaman (HTML), tampilan (CSS), dan logika interaktif (JavaScript).
- **Membuat Halaman Dinamis**
  DOM memungkinkan perubahan konten, gaya, atau struktur elemen tanpa perlu memuat ulang halaman.
- **Event Handling**
  DOM memfasilitasi interaksi dengan pengguna melalui berbagai event, misalnya klik, input, atau submit form.

4.3 Contoh Dasar DOM

Misalkan Anda memiliki file **index.html** sebagai berikut:

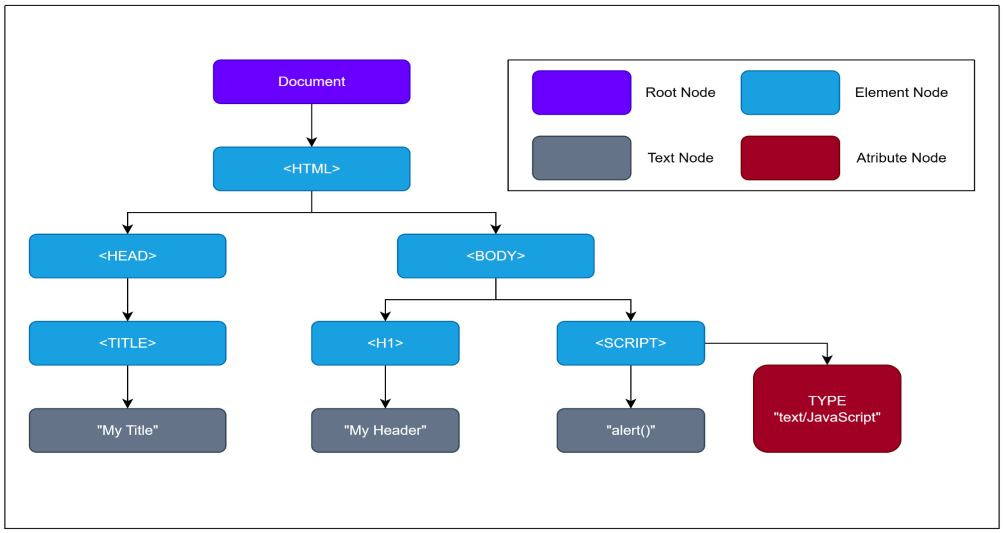
```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
  <title>JavaScript DOM</title>
 </head>
 <body>
  <h1 id="judul">Hello DOM!</h1>
 </body>
</html>
```

Jika dibuka di browser, halaman akan menampilkan teks **Hello DOM!**.

Selain itu, browser juga membangun **DOM Tree**, yaitu representasi dokumen HTML dalam bentuk struktur pohon, di mana setiap elemen, atribut, dan teks dianggap sebagai node yang saling terhubung secara hierarkis.

Visualisasi DOM Tree



4.4 DOM API dalam JavaScript

JavaScript dapat menggunakan **DOM API** untuk mengakses dan memanipulasi elemen.

a. **Memilih Elemen**

DOM menyediakan berbagai cara untuk **mengambil elemen HTML** sehingga kita bisa memanipulasinya dengan JavaScript. Berikut adalah beberapa metode yang umum digunakan:

- **Berdasarkan ID**

  Metode **getElementById()** dipakai untuk mencari elemen berdasarkan nilai atribut id. Karena id harus unik, hasil yang dikembalikan selalu satu elemen.

```
<p id="deskripsi">Belajar DOM itu menyenangkan!</p>

<script>
        let teks = document.getElementById("deskripsi");
        teks.style.color = "green"; // ubah warna teks jadi hijau
</script>
```

- **Berdasarkan Class**

  Metode **getElementsByClassName()** akan memilih semua elemen yang memiliki nama class tertentu. Hasilnya berupa HTMLCollection yang bisa diakses seperti array.

```
<div class="kotak">Kotak 1</div>
<div class="kotak">Kotak 2</div>

<script>
        let kotak = document.getElementsByClassName("kotak");
        kotak[1].style.backgroundColor = "lightblue";
</script>
```

- **Berdasarkan Tag**

  Metode **getElementsByTagName()** digunakan untuk mengambil elemen sesuai nama tag-nya, misalnya <p>, , atau <li>.

```
<ul>
        <li>Apel</li>
        <li>Pisang</li>
        <li>Mangga</li>
</ul>

<script>
        let buah = document.getElementsByTagName("li");
        buah[0].style.fontWeight = "bold";
</script>
```

- **Menggunakan querySelector()**

  Metode **querySelector()** lebih fleksibel karena memakai aturan CSS selector. Namun, ia hanya mengambil elemen pertama yang cocok.

```
<div id="info">
        Informasi penting
</div>

<p class="teks">
        Belajar JavaScript
</p>

<script>
    // berdasarkan class
    let teks = document.querySelector(".teks");
    teks.style.fontSize = "20px";

    // berdasarkan id
    let info = document.querySelector("#info");
    info.style.border = "2px solid red";
```

```
    // berdasarkan tag
    let paragraf = document.querySelector("p");
    paragraf.style.backgroundColor = "yellow";
</script>
```

Selain itu, **querySelector()** juga mendukung seleksi yang lebih kompleks:

- **Nested element (elemen bersarang)**

```
let pertama = document.querySelector("ul li");
pertama.style.color = "blue";
```

- **Filter atribut**

```
let i = document.querySelector("input[type='password']");
input.setAttribute("placeholder", "Masukkan password");
```

- **Menggunakan querySelectorAll()**

Jika ingin mengambil semua elemen yang cocok dengan selector CSS, gunakan **querySelectorAll()**. Hasilnya berupa NodeList yang bisa di-loop dengan **forEach()**.

```
<p class="catatan">Catatan A</p>
<p class="catatan">Catatan B</p>
<p class="catatan">Catatan C</p>

<script>
        let semua = document.querySelectorAll(".catatan");
        semua.forEach(item => {item.style.textTransform = "uppercase"; // semua
teks jadi huruf besar
        });
</script>
```

b. Memanipulasi Elemen

Manipulasi elemen berarti kita bisa mengubah isi, atribut, gaya (style), bahkan menambah atau menghapus elemen. Berikut beberapa cara yang sering digunakan:

- **Mengubah Isi Elemen**

  Konten di dalam sebuah tag HTML bisa diubah dengan properti textContent atau innerHTML.

```
<p id="pesan">Selamat datang!</p>

<script>
        let teks = document.getElementById("pesan");
        teks.textContent = "Halo, JavaScript!";
        // hasil: isi <p> berubah jadi "Halo, JavaScript!"
</script>
```

  textContent hanya untuk teks, sedangkan innerHTML bisa menyisipkan kode HTML baru.

- **Mengubah Atribut Elemen**

  Atribut HTML seperti src, href, atau alt dapat diatur ulang menggunakan setAttribute(). Untuk membaca nilainya, gunakan getAttribute().

```
<img id="gambar" src="foto1.jpg" alt="Foto lama">

<script>
        let img = document.getElementById("gambar");
        img.setAttribute("src", "foto2.jpg"); // ganti gambar
        console.log(img.getAttribute("alt")); // tampilkan "Foto lama"
</script>
```

- **Mengubah Tampilan (Style) Elemen**

Gaya CSS bisa dimodifikasi langsung lewat properti style. Selain itu, kita juga bisa menambah atau menghapus class dengan classList.

```
<div id="box">Kotak</div>

<script>
        let kotak = document.getElementById("box");
        kotak.style.backgroundColor = "lightgreen"; // ubah warna latar
        kotak.classList.add("besar");   // tambahkan class CSS
        kotak.classList.remove("kecil"); // hapus class CSS
</script>
```

- **Menambah dan Menghapus Elemen**

Kita bisa membuat elemen baru dengan createElement() lalu memasukkannya ke dalam DOM menggunakan appendChild() atau append(). Untuk menghapus, gunakan remove().

```
<ul id="daftar"></ul>

<script>
        let ul = document.getElementById("daftar");
        let li = document.createElement("li");
        li.textContent = "Item baru";
        ul.appendChild(li); // tambahkan ke daftar
        // hapus setelah 3 detik
        setTimeout(() => {
            li.remove();
        }, 3000);
</script>
```

- **Hubungan Parent dan Child**

Setiap elemen dalam DOM memiliki hubungan hierarki. Kita bisa menambahkan child ke dalam parent, atau menelusuri child/parent yang ada.

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

<script>
  let parent = document.getElementById("container");

  // buat elemen baru
  let span = document.createElement("span");
  span.textContent = "Ini anak dari container";

  // tambahkan sebagai child
  parent.appendChild(span);

  // akses parent dari span

  console.log(span.parentElement.id); // "container"
</script>
```

c. Event Handler

Event handler adalah fungsi JavaScript yang dipanggil ketika sebuah event (kejadian) terjadi pada elemen HTML. Event bisa berupa klik tombol, mengetik di input, mengirim form, menekan keyboard, atau interaksi lainnya. Dengan event handler, halaman web dapat menjadi interaktif dan responsif terhadap pengguna.

Berikut beberapa cara menggunakan event handler di JavaScript:

- **Inline Event Handler**

  Event ditulis langsung di dalam atribut elemen HTML dengan memanggil sebuah fungsi.

```
<button onclick="tampilkanPesan()">Klik Saya</button>

<script>
  function tampilkanPesan() {
  alert("Tombol berhasil ditekan!");
  }
</script>
```

Cara ini sederhana, tetapi kurang direkomendasikan untuk proyek besar karena membuat HTML dan JavaScript tercampur.

- **Property Event Handler**

  Event handler dipasang dengan mengakses elemen lewat DOM, lalu mengisi propertinya (onclick, onmouseover, dll).

```
<button id="tombol">Klik di sini</button>

<script>
      let tombol = document.getElementById("tombol");
      tombol.onclick = function() {
      console.log("Event onclick berhasil dipicu!");
  };
</script>
```

  Kelemahan: hanya bisa memasang **satu fungsi** untuk tiap event properti. Jika ditimpa, event sebelumnya akan hilang.

- **addEventListener()**

Metode yang paling fleksibel. Dengan addEventListener(), kita bisa menambahkan banyak event handler untuk event yang sama tanpa menimpa yang lain.

```
<button id="tombol2">Klik Lagi</button>

<script>
  let btn = document.getElementById("tombol2");

  btn.addEventListener("click", () => {
    alert("Handler pertama dijalankan");
  });

  btn.addEventListener("click", () => {
    console.log("Handler kedua juga ikut berjalan!");
  });
</script>
```

- **Event Object**

Setiap event yang terjadi otomatis membawa sebuah **event object**. Objek ini berisi informasi detail tentang event, misalnya **id elemen**, **tipe event**, posisi mouse, tombol keyboard yang ditekan, dll.

```
<input id="teks" type="text" placeholder="Ketik sesuatu..." />

<script>
  let input = document.getElementById("teks");

  input.addEventListener("keydown", function(event) {
    console.log("Anda menekan tombol:", event.key);
  });
</script>
```