



JavaScript i interaktywność – manipulacja DOM i obsługa zdarzeń

Sebastian Prawucki 63536
Paweł Ćwikilewicz 61894



DOM – Obiektowy Model Dokumentu

CO TO JEST DOM?

- DOM(Obiektowy Model Dokumentu) to sposób reprezentacji dokumentów HTML i XML w postaci struktury obiektowej.

JAK DZIAŁA DOM?

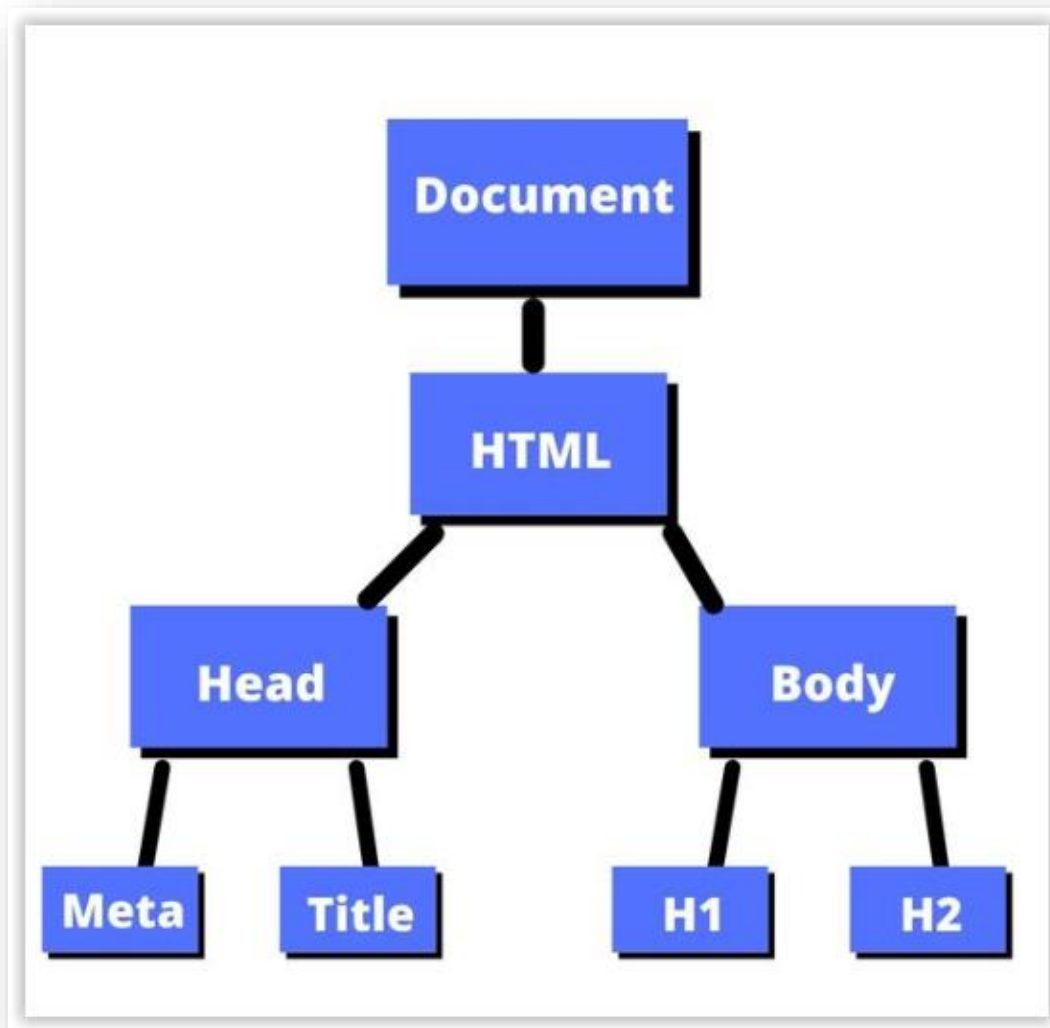
- pobranie pliku html,
- utworzenie drzewa DOM,
- korygacja błędów,
- udostępnienie DOM JavaScriptowi.

DOM – Obiektowy Model Dokumentu

Struktura drzewa DOM

Dlaczego DOM jest istotny?

- Dynamiczne modyfikowanie zawartości
- Manipulacja strukturą strony
- Obsługa zdarzeń użytkownika

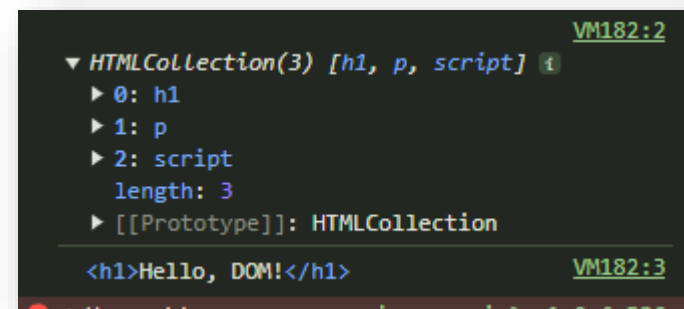


DOM – Obiektowy Model Dokumentu

```
<!DOCTYPE html>
<html>
<head>
  <title>Przykład DOM</title>
</head>
<body>
  <h1>Hello, DOM!</h1>
  <p>Przykładowy akapit.</p>

  <script>
    console.log(document.body.children); // Lista dzieci <body>
    console.log(document.body.firstChild); // Pierwszy element dziecka <body>
  </script>

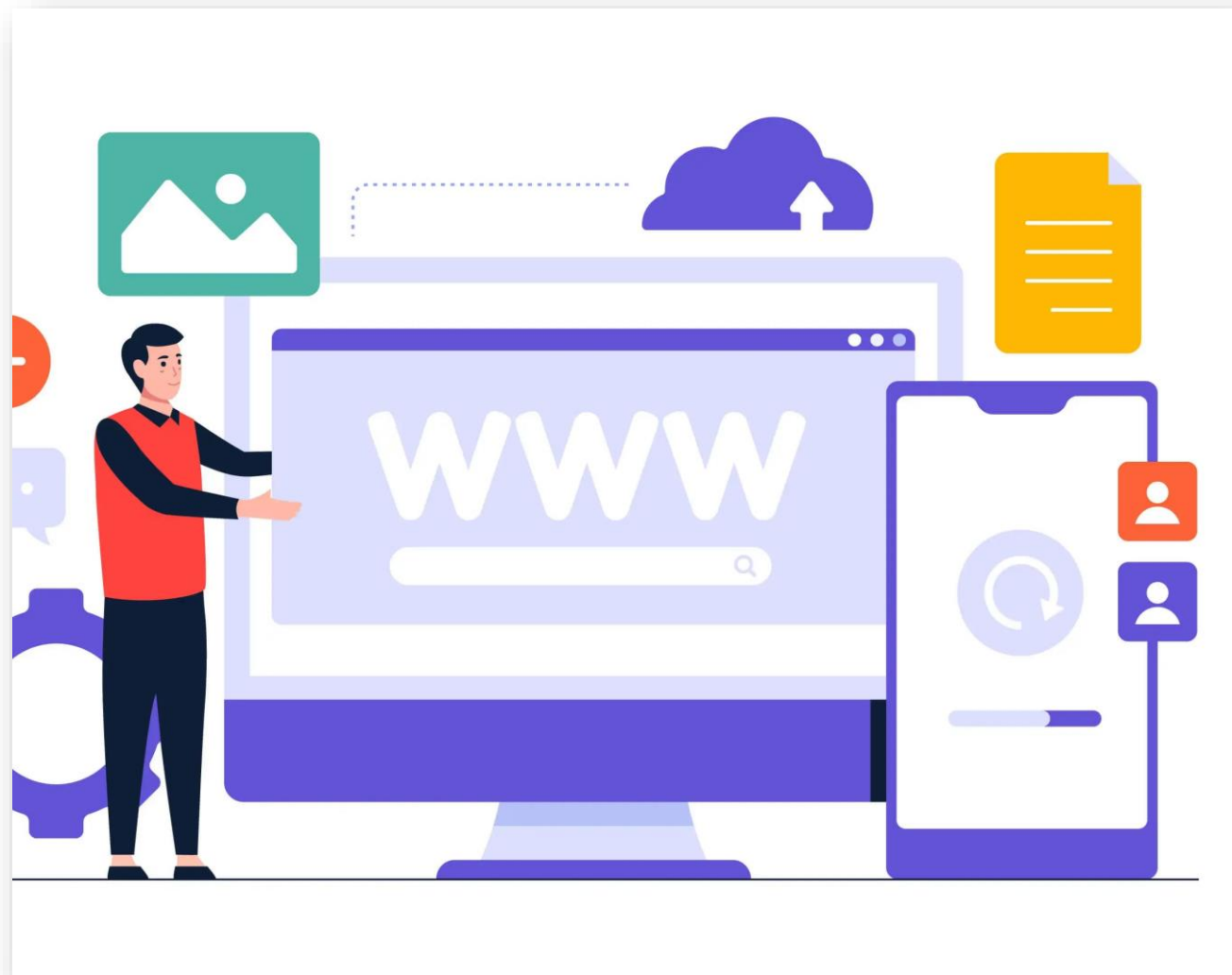
</body>
</html>
```



The screenshot shows a browser's developer console with the DOM tree expanded. The root node is 'HTMLCollection(3) [h1, p, script]' with a reference icon. It contains three children: 'h1', 'p', and 'script'. The 'h1' node is selected, showing its value as '<h1>Hello, DOM!</h1>'. The console also shows the source of the log statement as 'VM182:2'.

```
▼ HTMLCollection(3) [h1, p, script] t VM182:2
  ▶ 0: h1
  ▶ 1: p
  ▶ 2: script
    length: 3
  ▶ [[Prototype]]: HTMLCollection
  <h1>Hello, DOM!</h1> VM182:3
```

Manipulacja DOM za pomocą JavaScript





Wybieranie elementów DOM

- `document.getElementById()`
- `document.getElementsByClassName()`
- `document.getElementsByTagName()`
- `document.querySelector()`
- `document.querySelectorAll()`
- `document.activeElement`

Nawigacja po drzewie DOM

- parentNode, childNodes, firstChild, lastChild
- document.children
- nextElementSibling, previousElementSibling
(Przydatne, gdy chcemy dynamicznie odnaleźć elementy względem innych.)

```
<body>
<button id="btn1">Kliknij mnie</button>
</body>

<script>
let btn1 = document.getElementById("btn1");
console.log(btn1.parentNode.nodeName);
</script>

//w konsoli zostanie napisane "BODY"
```

Fragmenty dokumentu

- `document.createDocumentFragment()` – Tworzy tymczasowy fragment dokumentu, który można modyfikować w pamięci, a następnie dodać do DOM.

```
<script>
const list = document.getElementById("myList");
const fragment = document.createDocumentFragment(); // Tworzymy fragment

for (let i = 0; i < 1000; i++) {
  const li = document.createElement("li");
  li.textContent = `Element ${i}`;
  fragment.appendChild(li); // Dodajemy do fragmentu
}

list.appendChild(fragment); // Jeden update w DOM – duża oszczędność wydajności!
</script>
```


Klonowanie i zastępowanie elementów

- cloneNode(true/false) – Tworzy kopię istniejącego elementu
- replaceChild(newElement, oldElement) – Zastępuje dany element innym w jego rodzicu.

```
<script>
const original = document.querySelector(".box");
const clone = original.cloneNode(true);
document.body.appendChild(clone);

const container = document.getElementById("container");
const oldElement = document.getElementById("old");

// Tworzymy nowy element do zamiany
const newElement = document.createElement("p");
newElement.textContent = "To jest nowy element.";

// Zamieniamy stary element na nowy
container.replaceChild(newElement, oldElement);
</script>
```



Obsługa atrybutów danych (data-*)

Atrybuty **data-*** to specjalne atrybuty HTML, które pozwalają przechowywać **niestandardowe dane** w elementach. Można je łatwo odczytać i zmodyfikować w JavaScript za pomocą **dataset**.

Zastosowania:

- Przechowywanie dodatkowych informacji o elemencie (np. ID użytkownika, status).
- Ułatwienie obsługi dynamicznych interakcji w JS bez potrzeby ukrywania danych w atrybutach **class** lub **id**.

Zmiana treści i atrybutów elementów oraz stylów CSS

textContent

- pobiera lub ustawia **czysty tekst**.
- Bezpieczny przed XSS.
- Ignoruje HTML

innerHTML

- ustawianie i pobieranie **HTML** wewnątrz elementu.
- Podatny na ataki XSS

```
<script>
const content = document.getElementById("content");
console.log(content.innerHTML); // "<b>Tekst pogrubiony</b>"
content1.innerHTML = "<i>Nowa treść z kursywą</i>";

</script>
```

Zmiana treści i atrybutów elementów oraz stylów CSS

- `setAttribute()` – Pozwala dynamicznie zmieniać lub dodawać atrybuty do elementu.
- `getAttribute()` – Pozwala odczytać wartość dowolnego atrybutu elementu.
- `removeAttribute()` – Pozwala usunąć atrybut z elementu.
- `element.style.property` – Pobiera wartość stylu.
- `classList.add()`, `classList.remove()`, `classList.toggle()` – Pozwalają na łatwe dodanie, usunięcie lub przełączenie klas CSS

```
<a id="myLink" href="https://example.com" target="_blank">Kliknij mnie</a>
<p id="text">Przykładowy tekst</p>
<p id="test">Tekst Test</p>

<script>
const link = document.getElementById("myLink");
// GET ATTRIBUTE
console.log(link.getAttribute("href")); // "https://example.com"
console.log(link.getAttribute("target")); // "_blank"
console.log(link.getAttribute("class")); // null (jeśli brak atrybutu)

// SET ATTRIBUTE
link.setAttribute("href", "https://google.com");
link.setAttribute("title", "Przejdź do Google");

console.log(link.getAttribute("href")); // "https://google.com"
console.log(link.getAttribute("title")); // "Przejdź do Google"

// REMOVE ATTRIBUTE
link.removeAttribute("target"); // Usuwa otwieranie w nowym oknie
console.log(link.getAttribute("target")); // null

// STYLE PROPERTY
const text = document.getElementById("text");

text.style.color = "red"; // Zmienia kolor tekstu
text.style.fontSize = "24px"; // Ustawia rozmiar czcionki
text.style.backgroundColor = "yellow"; // Zmienia tło na żółte

// const text = document.getElementById("text");

// Dodajemy klasę "highlight"
test.classList.add("highlight");
test.classList.remove("highlight"); // Usuwa klasę "highlight"
test.addEventListener("click", () => {
  test.classList.toggle("highlight");
});
</script>
```

[Kliknij mnie](#)

Przykładowy tekst

Tekst Test

Tworzenie i usuwanie elementów:

- createElement() – Tworzy nowy element (ale nie dodaje go do DOM)
- appendChild() – dodaje element do innego elementu **zawsze na koniec listy dzieci** tego elementu.
- insertAdjacentHTML() – Pozwala wstawić cały kod HTML jako string

```
<script>
const newDiv = document.createElement("div"); // Tworzymy element <div>
newDiv.textContent = "Nowy element"; // Dodajemy tekst
newDiv.style.color = "blue"; // Dodajemy styl

document.body.appendChild(newDiv); // Teraz <div> pojawi się na stronie!

const container = document.getElementById("container");

container.insertAdjacentHTML("beforeend", "<p>Nowy paragraf</p>");

</script>
```

Usuwanie elementów na różne sposoby

removeChild(childElement)

- Starsza wersja, ale działa we wszystkich przeglądarkach
- Wymaga dostępu do rodzica

innerHTML

- Pozwala usunąć wszystkie elementy wewnątrz kontenera.

element.remove()

- Nie działa ze starszymi przeglądarkami.
- Bezpośrednio usuwa element, bez potrzeby znajdowania rodzica.
- Krótsza i bardziej czytelna.

```
<script>
const element = document.getElementById("usuwany-element");
element.parentNode.removeChild(element);
// Wybrany element zostanie usunięty z DOM

const element = document.getElementById("usuwany-element");
element.remove();
// Wybrany element zostanie usunięty z DOM

let lista = document.getElementById("lista");
lista.innerHTML = "";
// Usunięte zostaną wszystkie elementy wewnątrz
</script>
```

Obługa zdarzeń w JavaScript

1. Co to jest zdarzenie w JavaScript?

2. Przykłady zdarzeń

- | | | | |
|--------------------|-------------------------|-------------------------|---------------------|
| • Zdarzenia myszy: | • Zdarzenia klawiatury: | • Zdarzenia formularzy: | • Zdarzenia strony: |
| -click | -keydown | -submit | -load |
| -dblclick | -keyup | -change | -resize |
| -mouseover | | -input | -focus |
| -mousemove | | | -blur |

Obsługa zdarzeń w JavaScript – sposoby

1. Poprzez atrybut HTML
2. Poprzez właściwość onEvent w JavaScript
3. Poprzez addEventListener()

```
<button id="btn1">Kliknij mnie</button>
<script>

let btn1 = document.getElementById("btn1");

btn1.onclick = function() {
  alert("Przycisk został naciśnięty");
};
</script>
```

```
<button id="btn2">Kliknij mnie</button>

<script>
let btn2 = document.getElementById("btn2");

  btn2.addEventListener("click", function() {
    alert("Przycisk został naciśnięty");
  });

  btn2.addEventListener("click", function() {
    console.log("Kliknięcie zapisane w konsoli!");
  });
</script>
```

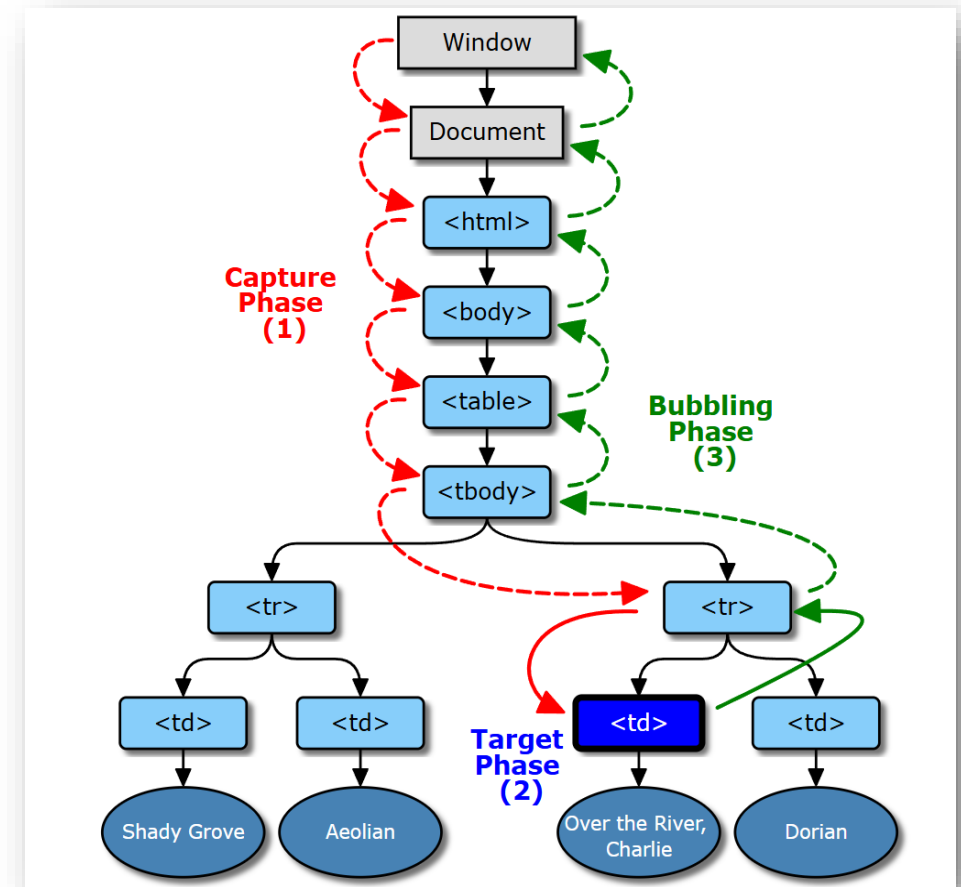
```
<button onclick="alert('Przycisk został naciśnięty')">Kliknij mnie</button>
```


Propagacja zdarzeń

- Propagacja zdarzeń to mechanizm określający kolejność obsługiwanie zdarzeń w hierarchii elementów DOM. Dzielimy ją na 3 fazy:

1. Faza przechwytywania (Capturing Phase)
2. Faza celowa (Target Phase)
3. Faza propagacji (Bubbling Phase)

- Metoda event. stopPropagation()



Delegacja zdarzeń

- Delegacja zdarzeń to technika, która pozwala obsługiwać wiele elementów potomnych, przypisując zdarzenie do ich elementu nadrzędnego.
- Zalety delegacji zdarzeń:
- Mniejsze obciążenie przeglądarki (poprawia wydajność)
- Obsługa dynamicznie dodanych elementów
- W wielu miejscach kod jest bardziej czytelny.

```
<ul id="list">
  <li>Element 1</li>
  <li>Element 2</li>
</ul>
<button id="addItem">Dodaj element</button>

<script>
  document.getElementById("list").addEventListener("click", (event) => {
    if (event.target.tagName === "LI") {
      console.log("Kliknięto: " + event.target.textContent);
    }
  });

  document.getElementById("addItem").addEventListener("click", () => {
    const newItem = document.createElement("li");
    newItem.textContent = "Nowy element";
    document.getElementById("list").appendChild(newItem);
  });
</script>
```

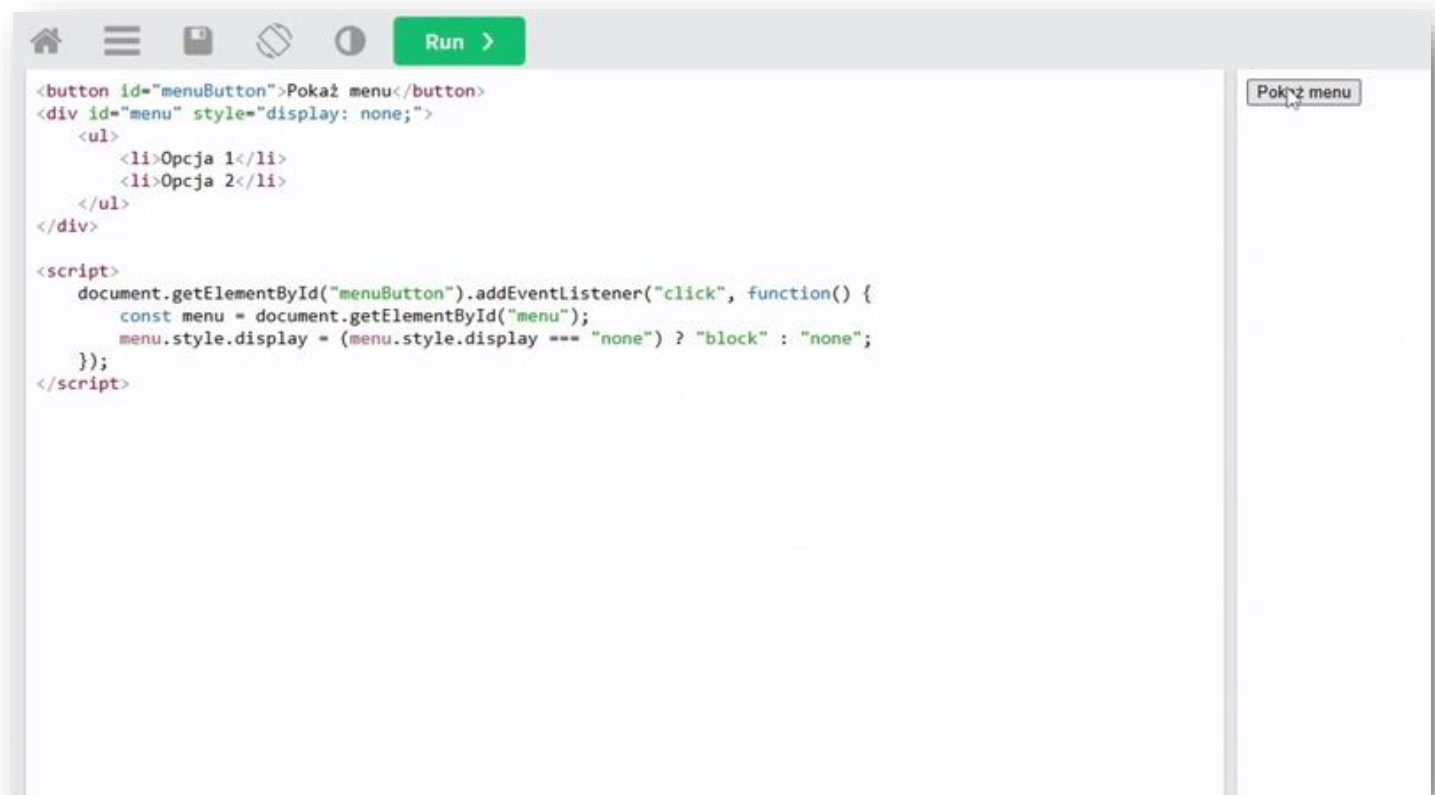


Najlepsze praktyki w manipulacji DOM i obsłudze zdarzeń

1. Wybór odpowiednich metod
2. Używanie `classList` zamiast `className` w celu dodawania, usuwania bądź przełączania klas
3. Unikanie `innerHTML` na dużych stronach
4. Usuwanie nasłuchiwczy zdarzeń

Praktyczne zastosowania interaktywności

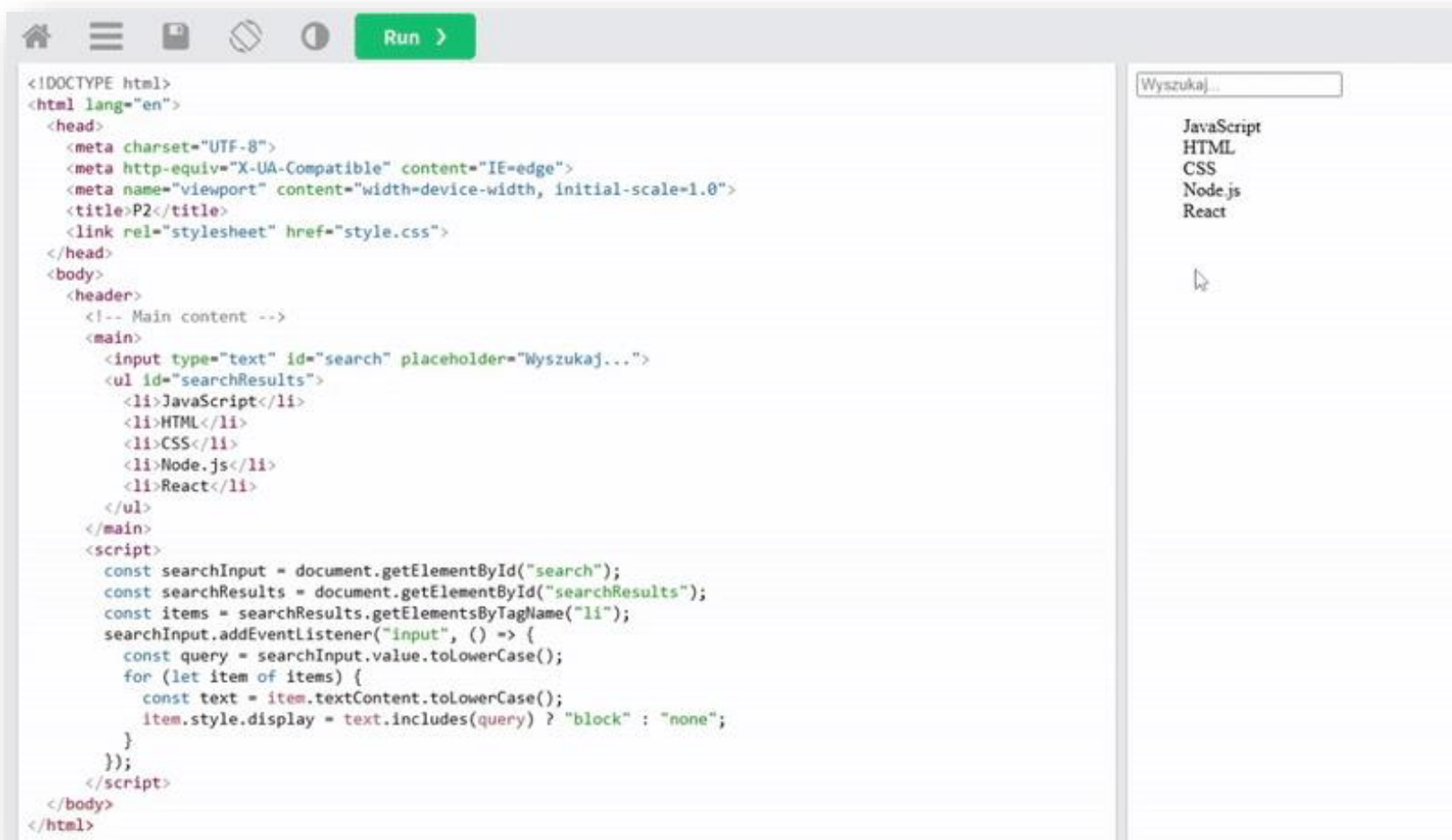
1. Dlaczego interaktywność jest istotna?
2. Przykłady interaktywnych funkcji na stronach

The image shows a web browser window with a light gray header bar containing icons for home, menu, save, and a green 'Run' button. The main content area displays a button with the text 'Pokaż menu' and a hidden menu. The menu is a vertical list with two items: 'Opcja 1' and 'Opcja 2'. The code in the background shows the HTML and JavaScript for this functionality. The HTML includes a button with id 'menuButton' and a div with id 'menu' containing a list of two items. The JavaScript adds a click event listener to the button that toggles the display style of the menu between 'block' and 'none'.

```
<button id="menuButton">Pokaż menu</button>
<div id="menu" style="display: none;">
  <ul>
    <li>Opcja 1</li>
    <li>Opcja 2</li>
  </ul>
</div>

<script>
  document.getElementById("menuButton").addEventListener("click", function() {
    const menu = document.getElementById("menu");
    menu.style.display = (menu.style.display === "none") ? "block" : "none";
  });
</script>
```

Przykład zastosowania interaktywności



The screenshot shows a web browser window with a search bar and a list of technologies. The search bar is labeled "Wyszukaj...". The list of technologies is: JavaScript, HTML, CSS, Node.js, and React. The browser's address bar shows "Run".

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>P2</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <header>
      <!-- Main content -->
      <main>
        <input type="text" id="search" placeholder="Wyszukaj...">
        <ul id="searchResults">
          <li>JavaScript</li>
          <li>HTML</li>
          <li>CSS</li>
          <li>Node.js</li>
          <li>React</li>
        </ul>
      </main>
    </body>
  </html>
```

```
const searchInput = document.getElementById("search");
const searchResults = document.getElementById("searchResults");
const items = searchResults.getElementsByTagName("li");
searchInput.addEventListener("input", () => {
  const query = searchInput.value.toLowerCase();
  for (let item of items) {
    const text = item.textContent.toLowerCase();
    item.style.display = text.includes(query) ? "block" : "none";
  }
});
```