JOURNAL OF APPLIED SCIENCE AND SOCIAL SCIENCE
eISSN 2229-3113 pISSN 2229-3205
Volume 15 Issue 10, October 2025
Impact factor: 2019: 4.679 2020: 5.015 2021: 5.436, 2022: 5.242, 2023: 6.995, 2024 7.75

DOM AND JAVASCRIPT TECHNOLOGIES

Hamroyev Bobirjon Bakhritdinovich
Asia International University,
Teacher of the Department of "General Technical Sciences"

Abstract: The Document Object Model (DOM) is an object-oriented representation of web documents that enables programmatic access and manipulation through JavaScript. DOM and JavaScript together form the core technologies behind dynamic, interactive, and responsive web applications. This article explores the structure of the DOM, fundamental APIs, event handling techniques, efficient manipulation methods, and performance considerations. Furthermore, it compares traditional DOM operations with modern approaches such as Virtual DOM, Shadow DOM, and reactive libraries (e.g., React and Vue) to highlight the practical applications of DOM/JS integration.

Keywords: DOM, JavaScript, Document Object Model, Event Handling, DOM Manipulation, Shadow DOM, Virtual DOM, Web Performance

Introduction

With the evolution of web development, static web pages have transformed into rich, interactive applications. In modern browsers, every HTML document is represented as a DOM tree — a structured collection of nodes and objects that can be accessed and modified using JavaScript. Through the DOM, developers can dynamically create, delete, and modify elements, update attributes, manage CSS classes, and handle user events in real time.The purpose of this article is to examine the integration of DOM and JavaScript from both theoretical and practical perspectives and to provide recommendations for effective and efficient usage.

Main Body

What Is the DOM and How It Works

The Document Object Model is a platform- and language-independent interface that represents an HTML or XML document as a tree structure. The root of this tree is the document object, from which we access document.documentElement (usually <html>), followed by <head> and <body>. Each element in the document is a node with its own attributes, children, and parent references.

Core JavaScript APIs for DOM Manipulation

Some of the most commonly used DOM APIs include:

Selectors:getElementById, getElementsByClassName, getElementsByTagName for traditional

JOURNAL OF APPLIED SCIENCE AND SOCIAL SCIENCE
eISSN 2229-3113 pISSN 2229-3205
Volume 15 Issue 10, October 2025
Impact factor: 2019: 4.679 2020: 5.015 2021: 5.436, 2022: 5.242, 2023: 6.995, 2024 7.75

access.querySelector and querySelectorAll for modern, CSS-style selection.**Element Creation and Modification**:createElement, createTextNode, appendChild, insertBefore, replaceChild, removeChild — used to construct and modify the DOM tree.**Attributes and Classes**:element.classList, setAttribute, and getAttribute to manage element properties and styles.**Content**:innerHTML and textContent are used to set or retrieve element content. Note: innerHTML can introduce XSS vulnerabilities if used with untrusted data.

Events and Event Handling

Events are the core mechanism for interaction within the DOM. Common event types include click, input, submit, keydown, and many others.**Inline Handlers**: e.g., <button onclick="..."> — outdated and not recommended.**addEventListener**: element.addEventListener('click', handler) is the modern method, allowing multiple listeners per event.**Event Object**: Properties such as event.target, and methods like preventDefault() and stopPropagation() provide fine-grained control.

Event Delegation is a powerful pattern for handling events efficiently. Instead of attaching handlers to many individual elements, a single handler on a parent element can manage events for all children, using event.target to determine the source.

DOM Manipulation and Performance

DOM manipulation is inherently costly because each change can trigger reflow and repaint operations in the browser. Efficient strategies include:**Batching Updates**: Use DocumentFragment or manipulate elements off-DOM, then insert them all at once.**Minimizing Reflows**: Avoid mixing reads and writes of layout properties (e.g., avoid repeatedly querying offsetHeight during modifications).**Class Swapping**: Apply multiple style changes by toggling a single class rather than applying many inline changes.**Virtual DOM**: Libraries like React use virtual trees to calculate minimal diffs, applying only necessary changes to the real DOM.

Shadow DOM and Componentization

The **Shadow DOM** is part of the Web Components specification. It allows encapsulating an element's internal structure and styles, shielding it from external CSS and scripts. This enables reusable, isolated, and modular components — a key principle in modern frontend development.Using innerHTML to insert external content can expose applications to XSS (Cross-Site Scripting) attacks. Sanitizing user input or preferring textContent ensures safer rendering.

Modern Libraries and the DOM

Modern frontend frameworks abstract away much of the manual DOM manipulation:**React**: Uses Virtual DOM to compute and apply the minimal necessary updates.**Vue/Svelte**: Utilize

JOURNAL OF APPLIED SCIENCE AND SOCIAL SCIENCE
eISSN 2229-3113 pISSN 2229-3205
Volume 15 Issue 10, October 2025
Impact factor: 2019: 4.679 2020: 5.015 2021: 5.436, 2022: 5.242, 2023: 6.995, 2024 7.75

reactive data binding to automatically synchronize data and view.However, understanding how the real DOM works is essential, since these abstractions rely on DOM principles for rendering, event propagation, and accessibility.

1. Changing Text Content Dynamically

```
const title = document.querySelector('.title');
title.textContent = 'New Title';
```

2. Event Delegation

```
document.querySelector('#list').addEventListener('click', (e) => {
  const item = e.target.closest('.item');
  if (item) {
    console.log('Clicked:', item.textContent);
  }
});
```

3. Efficient Element Insertion with DocumentFragment

```
const frag = document.createDocumentFragment();
for (let i = 0; i < 100; i++) {
  const li = document.createElement('li');
  li.textContent = `Item ${i}`;
  frag.appendChild(li);
}
document.querySelector('#list').appendChild(frag);
```

Conclusion

DOM and JavaScript together form the foundation of modern web application development. A solid understanding of DOM structure, APIs, event handling, delegation, performance optimization, and security is essential for building efficient and scalable user interfaces. Modern libraries simplify this process but rely on these underlying concepts. Mastering DOM and JavaScript technologies empowers developers to create dynamic, secure, and high-performing web experiences.

References

1. Baxridtdinovich, H. B. (2024). PYTHON DASTURLASH TILI VA UNING DASTURIY TA'MINOT SOHASIDAGI O'RNI. *MASTERS*, *2*(12), 41-48.
2. Baxridtdinovich, H. B. (2024). NEYRON TO'RLI TARMOQLAR. *WORLD OF SCIENCE*, *7*(12), 42-48.
3. Хамроев, Б. Б. (2024). PYTHON: ОСНОВЫ НАУКИ И

JOURNAL OF APPLIED SCIENCE AND SOCIAL SCIENCE
eISSN 2229-3113 pISSN 2229-3205
Volume 15 Issue 10, October 2025
Impact factor: 2019: 4.679 2020: 5.015 2021: 5.436, 2022: 5.242, 2023: 6.995, 2024 7.75

ИННОВАЦИЙ. *MASTERS*, *2*(12), 49-56.

4. Baxridtdinovich, H. B. (2025). THE IMPORTANCE AND APPLICATION OF POLYMORPHISM IN PYTHON. *PEDAGOGIK TADQIQOTLAR JURNALI*, *3*(2), 120-123.

5. Bakhritdinovich, H. B. (2025). APPLYING ABSTRACTION IN PYTHON PROGRAMMING. *ИКРО журнал*, *15*(01), 237-241.