THE USA JOURNALS
THE AMERICAN JOURNAL OF ENGINEERING AND TECHNOLOGY (ISSN – 2689-0984)
VOLUME 06 ISSUE07

PUBLISHED DATE: - 22-07-2024
DOI: - https://doi.org/10.37547/tajet/Volume06Issue07-05 **PAGE NO.: - 40-49**

RESEARCH ARTICLE **Open Access**

METHODS FOR IMPROVING THE PERFORMANCE OF JAVASCRIPT INTERFACES

Blahodelskyi Oleksandr Serhiyovych
Designer, MySteel BV Gemert, Netherlands

Abstract
This article discusses various methods for improving the performance of JavaScript interfaces, which are critically important in the context of the rapid development of digital technologies. Effective, responsive interfaces not only attract and retain users, but also improve the position of websites in search engines and increase conversion rates. The research covers topics related to reducing the number of requests, asynchronous programming, optimizing data loading and caching. The article provides specific examples and tips, such as minimizing queries, using local variables to increase the speed of code execution, using event delegation methods and algorithmic optimization. Tools for diagnosing and fixing performance issues such as Lighthouse and Google PageSpeed Insights are also being considered. These recommendations and techniques have been tested and systematized by experienced developers to achieve the best results in improving JavaScript performance.

Keywords Interface performance, interface, JavaScript, programming, IT, information technology.

INTRODUCTION

In the context of the rapid development of digital technologies, website performance has become a crucial factor determining the success of any online business, making this topic highly relevant. A fast, responsive, and user-friendly website not only attracts and retains visitors but also contributes to higher search engine rankings, increased conversion rates, and improved user experience (UX). For software engineers and web developers, it is essential to prioritize performance optimization methods in their projects [1].

JavaScript, as the primary programming language for developing interactive user interfaces, faces several challenges, such as the lack of document reading and loading capabilities, lack of remote access, and weak typing [2]. This article discusses the reasons why improving the performance of JavaScript interfaces is necessary, as well as the approaches and methods that can be applied to achieve this goal.

1. General Characteristics

JavaScript is one of the most in-demand programming languages. Over the past few years, this high-performance language has been present in all areas, from websites and games to servers, operating systems, and even robots. Furthermore, JavaScript ranks among the top languages used on GitHub, as evidenced by data from the Stack Overflow user survey (Figure 1).

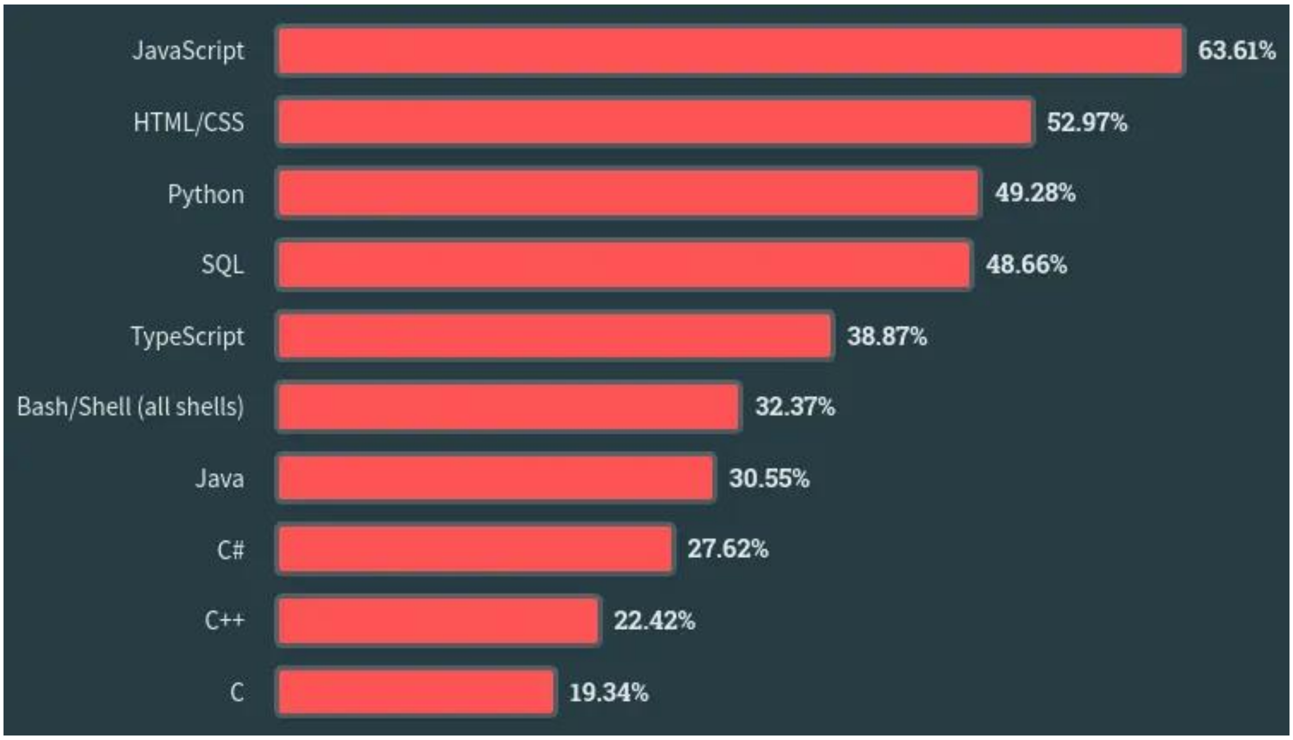

Figure 1. Top 10 programming languages according to the Stack Overflow survey [3].

With the growing popularity of JavaScript, developers bear the responsibility for its continued development and successful competition with other languages. The competition between TypeScript and JavaScript clearly demonstrates why the latter maintains its leading position.

In the face of increasing demand for both web development and mobile application creation, leading mobile application development companies and their specialists are doing everything possible to seize every opportunity to promote the growth of JavaScript [3].

2. Types of Interfaces

The term "interface" is widely used in the field of information technology and is similar in meaning to its everyday use. For example, a user interface encompasses control elements for devices such as websites, ATMs, phones, and more. For a TV remote control, the interface consists of buttons, while in a car, it includes all the levers and buttons for controlling the vehicle. Thus, an interface defines the way of interacting with a system.

Creating effective interfaces is a complex task that is often underestimated. In everyday life, we frequently encounter inconvenient interfaces, ranging from door handles to elevator controls. The more complex the system, the harder it is to create a user-friendly interface. Even in a simple example of a TV power button (with two states—on/off), control can be achieved through either two separate buttons or one button that changes its behavior depending on the current state.

In programming, the concept of an interface refers to a set of functions and their signatures (function names, the number and types of input parameters, and return values) that are independent of specific implementations. This definition aligns with the concept of an abstract data type. For example, in the context of working with points, the interface includes the functions described theoretically and implemented practically.

The relationship between abstraction and interface is as follows: abstraction describes the data we work with. Almost every web application includes the abstraction of a "user." On the Hexlet platform, for instance, there are abstractions like "course," "project," and others. The interface, on the other hand, represents a set of functions through which interaction with these data is carried out [4,5].

Below, Table 1 provides a description of a list of interfaces (objects) that can be used when developing various web applications.

Table 1. Description of Interfaces (Objects) [6,7]

Interface / Object	**Description**
Document	The Document interface represents any web page loaded in the browser and serves as an entry point to the web page's content, which is the DOM tree. The Document interface provides functions globally for the document, describing common properties and methods for any type of document.
Element	The Element interface describes methods and properties common to all kinds of elements, representing one of the objects in the Document.
Event	The Event interface represents any event that occurs in the Document Object Model (DOM). Some events are created directly by the user (e.g., mouse or keyboard events), while others are generated by the application programming interface (API), such as events indicating that an animation has ended or a video has been paused.
EventTarget	The EventTarget interface is implemented by objects that can receive and handle events, such as Element, Document, Window, and XMLHttpRequest.
Node	DOM classes form a hierarchy with the Node interface at the top, inherited by several types of DOM API objects, allowing these types to be processed similarly. The Node interface provides properties and methods for performing manipulations on the DOM tree. All objects in the document tree (including the Document object itself) inherit from the Node interface.
NonDocumentTypeChildNode	The NonDocumentTypeChildNode interface contains properties related to node objects that can have a parent object but do not fit the DocumentType interface. NonDocumentTypeChildNode is a raw interface, and no object of this type can be created; it is implemented by objects like Element and CharacterData.
ParentNode	The ParentNode object contains methods and properties common to all types of node objects (Node objects) that can have child elements. It is implemented by objects such as Element, Document, and DocumentFragment.
Storage	The Storage interface provides access to a local storage object or a session storage object for a specific domain (the source that created the storage object). It allows adding, modifying, or deleting stored data items.
WebSocket	It is the primary interface for establishing a connection to a WebSocket server and subsequently sending and receiving connection data.
Web Socket API	This is an advanced technology that allows pages to use the WebSocket protocol to open a two-way interactive communication session between the

	user's browser and a server (remote node). With this API, you can send messages to the server and receive event-driven responses without polling the server for a reply.
Web Storage API	The Web Storage API provides mechanisms for browsers to store key-value pairs in a much more intuitive way than using cookies (a small piece of data sent by a web server and stored on the user's computer).
Window	The Window interface represents a window or frame in the browser containing a DOM document.
WindowOrWorker GlobalScope	The WindowOrWorkerGlobalScope mixin describes several properties and functions common to the Window and WorkerGlobalScope interfaces.
XMLHttpRequest	XMLHttpRequest is an interface that provides functionality for exchanging data between the client and the server, allowing data retrieval via a URL without the need for a full page refresh. The XMLHttpRequest interface is widely used in asynchronous AJAX requests (Asynchronous JavaScript and XML), enabling specific page parts to be updated without disrupting user activity. Additionally, XMLHttpRequest is extensively used in the development of single-page applications (SPA).

When it comes to JavaScript performance, it needs to be considered at every stage of development. Table 2 provides a description of performance improvement options.

Table 2. Description of Performance Improvement Options [8]

Variant	Description
Reducing the number of requests	Fewer requests mean fewer transfer cycles to the server, which reduces latency.
Retrieving only necessary data	Reduce the amount of data sent over the network. Additionally, it reduces server load.
Ensuring good page load performance	Ensure that the user interface responds to user actions. For example, update the menu on the page before downloading more than 100 records.
Using asynchronous calls and templates whenever possible	Polling is a heavier burden on performance compared to using asynchronous calls or callbacks.
Caching is key	Caching further reduces server load, providing an immediate performance boost.
Preparing for more page views than you ever imagined	A target page with a large amount of data is suitable if you only have a few hits. But if you get thousands of hits, it can really impact performance.

3. Methods for Improving JavaScript Performance

To successfully obtain data, applications often make multiple internal API calls. Various middleware can be used for each function to effectively address this task. Although JavaScript is a single-threaded language, it efficiently handles asynchronous operations thanks to its event-driven execution model. Instead of using external libraries like async.js, modern JavaScript offers native constructs—Promises and async/await—for managing asynchronous operations. These mechanisms allow developers to write clean and maintainable code that handles asynchronous tasks in the event loop, preventing the main thread from blocking and improving interface performance.

```
// Asynchronous function to fetch user data
async function fetchUserData(userId) {
    try {
        // Using fetch API to request user data
        const response = await fetch(`https://api.example.com/users/${userId}`);
        if (!response.ok) {
            throw new Error(`Error: ${response.status}`);
        }
        const userData = await response.json();
        return userData;
    } catch (error) {
        console.error('Failed to fetch user data:', error);
    }
}

// Using async/await to process user data
async function processUser(userId) {
    const userData = await fetchUserData(userId);
    if (userData) {
        console.log('User data retrieved:', userData);
        // Additional user data processing logic
    }
}

// Function call
processUser(1);
```

However, even with the use of asynchronous functions, JavaScript sometimes requires external libraries to handle synchronous blocking calls, which can negatively impact performance. Therefore, it is crucial to integrate asynchronous APIs into application code. For beginners, mastering asynchronous programming can be challenging and requires attention to detail.

To enhance JavaScript performance, it is essential to write small and lightweight code, especially for mobile applications where performance is critical. Code optimization can be achieved through:

1. Gzip Compression: Gzip is a popular compression tool used to reduce the size of JavaScript files. Compressing files before sending them to the browser significantly reduces latency and improves overall application performance.

2. DOM Access Optimization: Interaction with the DOM can significantly slow down JavaScript execution. To improve performance, minimize the number of DOM accesses by caching references to browser objects or reducing the number of DOM traversal cycles. For example, updating a list of items in the DOM can be optimized using DocumentFragment:

```
function updateList(items) {
  const fragment = document.createDocumentFragment();
  items.forEach(item => {
    const li = document.createElement('li');
    li.textContent = item;
    fragment.appendChild(li);
  });
  document.getElementById('list').appendChild(fragment);
}
```

3. Caching: Caching data is one of the most effective ways to improve performance. This can be achieved using the Cache API in JavaScript or HTTP caching. Reusing cached data reduces the number of requests and speeds up the application. For example, using the Cache API to cache data:

```
caches.open('my-cache').then(cache => {
  cache.add('https://api.example.com/data').then(() => {
    console.log('Data cached successfully!');
  });
});
```

4. Memory Management and Leaks: Preventing memory leaks is essential, as they can lead to increased memory consumption by the loaded page, negatively impacting application or website performance. Tools like Chrome DevTools can be used to track memory leaks, allowing you to record timelines and analyze JavaScript performance.

5. Event Delegation: Learning event delegation is useful for improving JavaScript efficiency. This method allows a single event handler to manage events across the entire page, enhancing efficiency. Without event delegation, large web applications may face issues due to numerous event handlers. Event delegation, on the other hand, reduces memory requirements for handling events and decreases the number of bindings between code and the DOM. Example of event delegation:

```
document.addEventListener('click', function(event) {
  if (event.target.matches('.clickable')) {
        handleEvent(event.target);
    }
});
```

6. Efficient Page Loading: Ensuring efficient page loading is important for meeting user expectations of quick responses. It is unnecessary to load all site or application functions during initial page load. Techniques like Lazy Loading and asynchronous loading allow specific functions to load only when needed, such as when the user scrolls to a particular element or navigates to a new section of the site. Example of Lazy Loading for images:

```
document.addEventListener("DOMContentLoaded", function() {
    let lazyImages = [].slice.call(document.querySelectorAll("img.lazy"));
    let active = false;

    const lazyLoad = function() {
        if (active === false) {
            active = true;
            setTimeout(function() {
                lazyImages.forEach(function(lazyImage) {
                    if ((lazyImage.getBoundingClientRect().top <= window.innerHeight &&
lazyImage.getBoundingClientRect().bottom >= 0) &&
getComputedStyle(lazyImage).display !== "none") {
                        lazyImage.src = lazyImage.dataset.src;
                        lazyImage.classList.remove("lazy");

                        lazyImages = lazyImages.filter(function(image) {
                            return image !== lazyImage;
                        });

                        if (lazyImages.length === 0) {
                            document.removeEventListener("scroll", lazyLoad);
                            window.removeEventListener("resize", lazyLoad);
                            window.removeEventListener("orientationchange", lazyLoad);
                        }
                    }
                });

                active = false;
            }, 200);
        }
    };

    document.addEventListener("scroll", lazyLoad);
    window.addEventListener("resize", lazyLoad);
    window.addEventListener("orientationchange", lazyLoad);
  });
```

7. Performance Testing in Various Environments: For developing high-performance websites, it is critical to test code in different environments and on various devices. This helps identify potential performance issues and optimize the code for different usage conditions. Tools like Google's Lighthouse help evaluate performance, accessibility, progressive web applications, and other important aspects of a website.

8. Removing Unused JavaScript: Removing unused parts of JavaScript code reduces the time needed by the browser to compile and transfer the code, positively impacting load speed and overall performance [9]. This is achieved by eliminating unused JavaScript. Tools like Webpack or Rollup provide Tree Shaking capabilities that automatically remove unused code from the final bundle. Example of using Tree Shaking with Webpack:

Ensure the mode is set to production in your Webpack configuration file, which automatically activates Tree Shaking:

```
module.exports = {
  mode: 'production',
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: __dirname + '/dist'
  }
};
```

9. Performance of Built-In JavaScript Methods: It is also interesting to consider the performance of built-in object methods, which, due to low-level optimizations in engines, almost always perform faster than manually written JavaScript equivalents. The performance difference between built-in methods and hand-written equivalents is especially noticeable in Firefox, Opera, Safari, and IE. In V8 (Chrome), most built-in JavaScript methods are written in JavaScript, so the speed increase is not as significant. Figure 1 shows the performance difference based on method implementation.

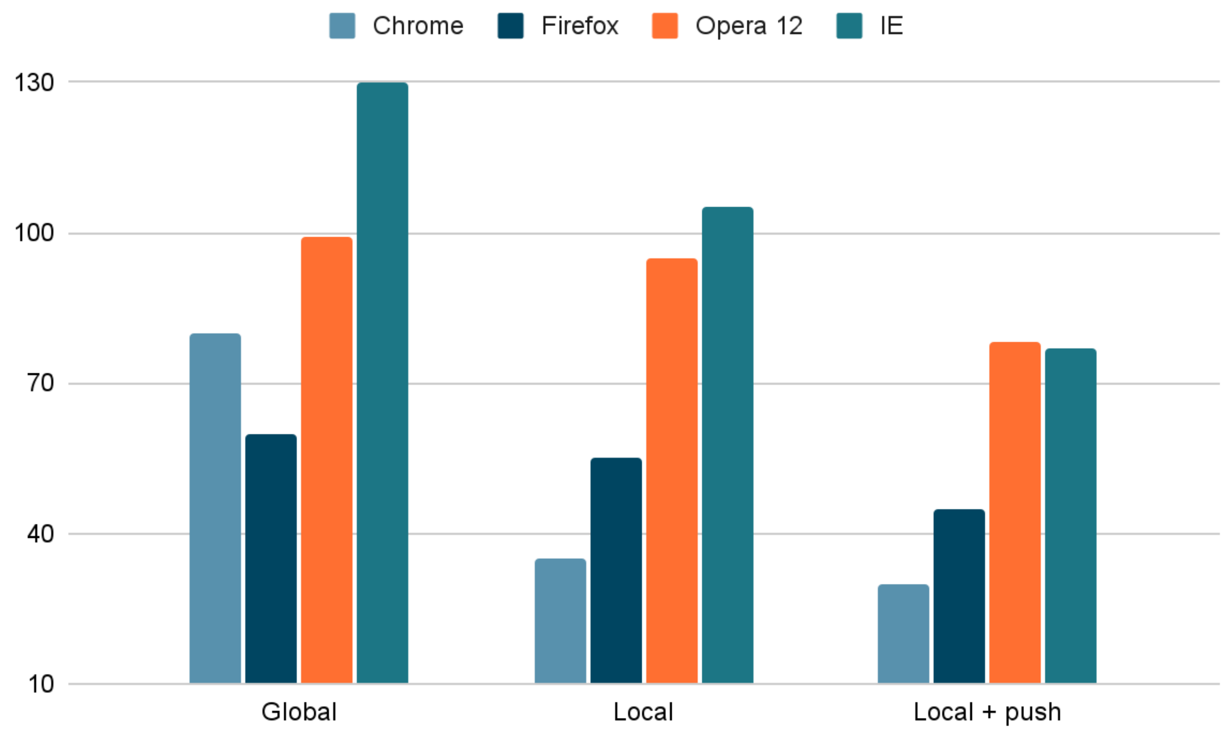

Figure 1. Performance differences based on method implementation [10].

CONCLUSION

Thus, methods for enhancing the performance of JavaScript interfaces play a crucial role in creating efficient and responsive web applications. The techniques discussed in this article, such as reducing the number of requests, utilizing asynchronous programming, and optimizing DOM access, significantly improve the user experience. Tools like Lighthouse and Google PageSpeed Insights provide developers with the ability to identify and resolve performance issues, ensuring high speed and efficiency of websites. Implementing these methods and tools not only improves user interaction with web applications but also increases the competitiveness of online businesses. Adhering to the recommendations presented in this article will enable developers to create high-performance and reliable JavaScript applications, which is an essential part of success in the modern digital environment.

REFERENCES

1. JavaScript optimization techniques for faster website loading. [Electronic resource] Access mode: https://it-dev-journal.ru/articles/metody-optimizaczii-java-script (accessed 05/22/2024).JavaScript: description, advantages and disadvantages, usage . [Electronic resource] Access mode: https://otus.ru/journal/javascript-opisanie-preimushhestva-i-nedostatki-ispolzovanie / (accessed 05/22/2024).
2. Why JavaScript remains a popular programming language in 2024. [Electronic resource] Access mode: https://tproger.ru/articles/populyarnost-javascript (accessed 05/22/2024).
3. Interface. [Electronic resource] Access mode: https://ru.hexlet.io/courses/js-data-abstraction/lessons/interface/theory_unit (accessed 05/22/2024).
4. Program at the interface level. [Electronic resource] Access mode:

https://habr.com/ru/articles/568810 / (accessed 05/22/2024).

5. JavaScript interfaces of the web API. [Electronic resource] Access mode: https://basicweb.ru/javascript/js_web_api.php (accessed 05/22/2024).
6. Interface. [Electronic resource] Access mode: https://hemantajax-2.gitbook.io/javascript-step-by-step/interface (accessed 05/22/2024).
7. JavaScript templates and performance. [Electronic resource] Access mode: https://learn.microsoft.com/ru-ru/sharepoint/dev/solution-guidance/javascript-patterns-and-performance (accessed 05/22/2024).
8. Using JavaScript Software: tips and suggestions for improving code performance. [Electronic resource] Access mode: https://allbachelor.com/2023/07/19/javascript-performance-optimization-tips-and-techniques-for-better-code-performance / (accessed 05/22/2024).
9. We improve the performance of the client side of the web application. [Electronic resource] Access mode: https://xakep.ru/2014/07/22/speedup-client-javascript / (accessed 05/22/2024).