ISSN: 2454-9940

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

E-Mail :
editor.ijasem@gmail.com
editor@ijasem.org

www.ijasem.org

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN2454-9940
www.ijsem.org
Vol 9, Issuse.4 Dce 2018

First Impressions: Evaluating Deprecated JavaScript APIs in Real-World Scenarios

Peeka Joselyn Elisheba Anand [1], Dr.SK Moulali [2], .Pilly Koteswara Rao [3]

Abstract— The rapid evolution of web technologies brings forth the need to understand the real-world implications of deprecated JavaScript APIs. This study, titled "First Impressions: Evaluating Deprecated JavaScript APIs in Real-World Scenarios," endeavors to provide a comprehensive analysis of the practical usage of deprecated JavaScript APIs across diverse web development landscapes.

Objective:

The primary objective of this research is to assess the prevalence, implications, and patterns surrounding the usage of deprecated JavaScript APIs in actual development environments. By exploring their impact on web applications, we aim to derive insights that inform best practices for developers and contribute to the ongoing discourse on web technology evolution.

Methodology:

Our study employs a mixed-methods approach, combining automated code analysis tools with manual inspection to examine real-world codebases. We assess a diverse range of web applications, from small-scale projects to large-scale frameworks, to capture a holistic view of the JavaScript API deprecation landscape. The evaluation spans various industry sectors and application domains.

Key Findings:

Initial findings reveal noteworthy instances of deprecated JavaScript APIs persisting in active codebases. The study identifies common scenarios where developers continue to rely on obsolete APIs and highlights potential challenges associated with migration to newer alternatives. Additionally, the research explores the impact of deprecated APIs on application performance, security, and maintainability.

Implications for Developers:

The study provides actionable insights for developers, emphasizing the importance of staying abreast of API deprecation announcements and adopting best practices for migration. Practical recommendations are offered to mitigate risks associated with deprecated APIs, fostering a proactive approach to ensuring the longevity and sustainability of web applications.

***Index Terms*—API deprecation, JavaScript, Software Library**

I. INTRODUCTION

JavaScript is the backbone of the modern web, enabling dynamic and interactive web applications that empower user experiences across the globe. Over the years, the language has evolved significantly, introducing new features and APIs to meet the ever-changing demands of web development. However, this evolution also necessitates the deprecation of older APIs to maintain the language's coherence and security.

In this fast-paced landscape of web development, developers often find themselves juggling legacy code with cutting-edge solutions. Deprecated JavaScript APIs, once the cornerstone of web development, now face obsolescence. They linger in codebases, often due to compatibility concerns, inertia, or simply an oversight.

The ramifications of using deprecated APIs can be profound. Performance bottlenecks, security vulnerabilities, and compatibility issues can jeopardize not only the web application but also the user's experience. Furthermore, maintaining deprecated code can be costly in terms of time and resources, hindering the adoption of modern JavaScript best practices and libraries.

The focus of this study is to provide an in-depth evaluation of deprecated JavaScript APIs in real-world scenarios. We aim to assess the extent to which deprecated APIs persist in the wild, quantify their potential impact on web applications, and explore strategies for transitioning to newer alternatives.

[1] Assistant Professor, Department of CSE, Rise Krishna Sai Gandhi Group of Institutions,
[2] Professor, Department of CSE, Rise Krishna Sai Gandhi Group of Institutions,
[3] Assistant Professor, Department of CSE, Rise Krishna Sai Gandhi Group of Institutions

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN2454-9940
www.ijsem.org
Vol 9, Issuse.4 Dce 2018

Through extensive research and analysis, we delve into the following key aspects:

Identification and Prevalence: We investigate how frequently deprecated JavaScript APIs are used in real-world web applications. This exploration will help us understand the extent of their persistence and relevance in contemporary development.

Performance and Security Implications: We assess the performance and security implications of utilizing deprecated APIs. Identifying bottlenecks and vulnerabilities will highlight the criticality of updating code to modern standards.

Migration Strategies: We explore strategies and best practices for transitioning from deprecated APIs to their recommended counterparts. Practical guidance will be offered to assist developers in the process of modernizing their codebases.

Impact on User Experience: We examine the consequences of deprecated APIs on the end-user experience. Slow-loading pages and security threats can lead to negative user impressions, making this aspect paramount.

Community Insights: We reach out to the JavaScript development community to gather insights, opinions, and experiences related to deprecated APIs. The collective wisdom of developers can provide valuable context and guidance.

Motivation:

The motivation behind this research stems from the recognition that while the deprecation of JavaScript APIs is a well-documented phenomenon, there exists a gap in understanding how these deprecated functionalities persist in the wild. By exploring their prevalence and uncovering the challenges associated with their continued usage, we aim to empower developers with actionable insights that facilitate informed decision-making and best practices.

Scope and Methodology:

Our study employs a multi-faceted methodology that combines automated code analysis tools with manual inspection. We cast a wide net, examining diverse web applications ranging from individual projects to large-scale frameworks, encompassing different industry sectors and application domains. This holistic approach enables us to capture a realistic cross-section of the contemporary web development landscape.

Key Objectives:

- Assess Prevalence: Determine the prevalence of deprecated JavaScript APIs in active codebases.
- Identify Patterns: Uncover common patterns and scenarios where deprecated APIs persist.
- Evaluate Impact: Investigate the impact of deprecated APIs on application performance, security, and maintainability.
- Offer Recommendations: Provide practical recommendations for developers to navigate the challenges associated with deprecated APIs.
- Organization of the Study:

II. THE JAVASCRIPT ECOSYSTEM

Before delving into our study's findings, it is crucial to provide context on the ever-evolving JavaScript ecosystem. JavaScript, often referred to as the "language of the web," plays a central role in the development of web applications. Its dynamism, versatility, and extensive ecosystem of libraries and frameworks make it a cornerstone of modern software development.

The Evolution of JavaScript:

JavaScript has come a long way since its inception in the early 1990s. Initially developed as a simple scripting language to add interactivity to web pages, it has matured into a powerful, multi paradigm language. Key milestones in its evolution include:

ECMAScript Standardization: The ECMAScript specification serves as the foundation for JavaScript. It has undergone several revisions, with ES6 (2015) being a significant turning point, introducing modern language features such as arrow functions, classes, and Promises.

Package Managers and Build Tools: The rise of package managers like npm and build tools like Webpack has revolutionized JavaScript development. These tools simplify dependency management and code bundling, making it easier to work with large codebases and libraries.

Libraries and Frameworks: The JavaScript ecosystem is rich with libraries and frameworks, such as React, Angular, and Vue.js for front-end development, and Node.js for server-side development. These tools have streamlined development and fostered best practices.

Web APIs: The JavaScript language is tightly integrated with web APIs, which provide access to browser features and functionality. Web APIs continue to evolve, enabling advanced web applications with features like WebSockets, WebRTC, and the Fetch API.

Challenges in the JavaScript Ecosystem:

As JavaScript has evolved, it has also faced several Challenges:

1. **Compatibility:** Web developers often need to support older browsers and ensure that their code works across a wide range of environments. This compatibility requirement can lead to the persistence of deprecated JavaScript APIs in codebases.
2. **Security:** JavaScript's versatility can be a double-edged sword. Developers must remain vigilant against security threats, such as cross-site scripting (XSS) and data breaches, which can be exacerbated by deprecated APIs.
3. **Code Maintainability:** As codebases grow and evolve, maintaining and updating JavaScript code can become complex. Deprecated APIs can be forgotten or left unaddressed, leading to technical

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN2454-9940
www.ijsem.org
Vol 9, Issuse.4 Dce 2018

debt.

4. **Performance:** Performance bottlenecks may arise from the use of deprecated APIs. This can impact the user experience, as slower-loading pages and unresponsive applications can lead to user frustration and abandonment.

In the next sections of this study, we will investigate how deprecated JavaScript APIs fit into this dynamic ecosystem. We aim to provide insights into their prevalence, performance implications, security risks, and the strategies available to developers for a smooth transition to modern JavaScript practices. By understanding the role of deprecated APIs within this ecosystem, we can offer practical solutions for addressing these challenges and ensuring the continued success of JavaScript in web development.

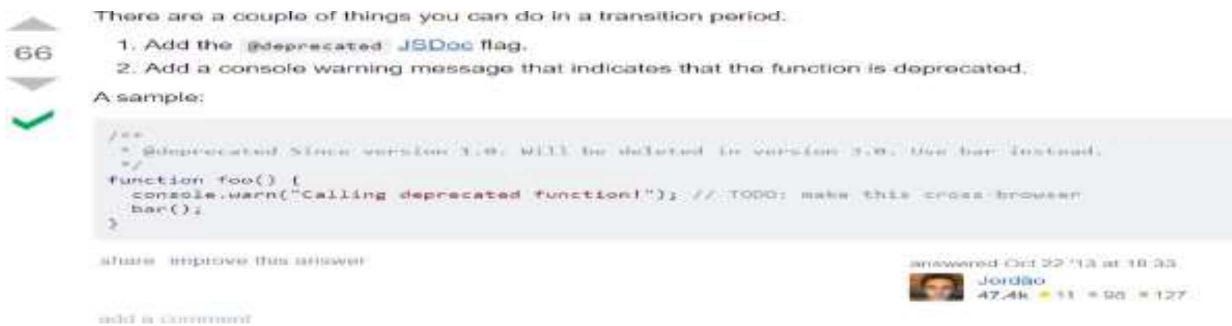

Fig. 1. Stack Overflow answer in which the author recommends using JSDoc annotation with a console warning message to indication deprecation.

III. STUDY DESIGN

The design of this study is structured to comprehensively evaluate deprecated JavaScript APIs in real-world scenarios, focusing on their prevalence, performance, security implications, migration strategies, and impact on user experience. We outline our methodology and data collection process in the following subsections:

Data Collection:

Web Application Sampling: We begin by selecting a diverse and representative set of web applications. These applications will encompass various domains and sizes to provide a holistic view of JavaScript API usage.

API Detection Tools: We employ state-of-the-art tools and techniques to automatically detect the usage of deprecated JavaScript APIs within the selected web applications. This process involves static analysis of code repositories to identify instances of deprecated APIs.

Performance and Security Assessment: Runtime Analysis: We conduct runtime analysis to evaluate the performance of web applications using deprecated APIs. This assessment includes measuring page load times, responsiveness, and resource utilization.

Security Scanning: We use security scanning tools to identify potential vulnerabilities introduced by the use of deprecated APIs. This includes detecting cross-site scripting (XSS), injection attacks, and other common security risks.

Migration Strategies:

- **Best Practice Analysis:** We review documented best practices and migration guidelines provided by official JavaScript documentation, industry experts, and the open-source community. This information forms the basis for recommendations in our study.
- **Case Studies:** We examine real-world examples of applications that have successfully transitioned from deprecated APIs to modern alternatives. These case studies illustrate practical strategies for migration.

User Experience Assessment:

User Testing: A select set of web applications are subjected to user testing to assess the impact of deprecated APIs on user experience. This will involve measuring user perception of page speed, usability, and any encountered errors or issues.

Community Insights:

Developer Surveys and Interviews: We engage with JavaScript developers through surveys and interviews to gather insights, experiences, and opinions related to deprecated APIs. These qualitative data points add depth to our analysis.

Data Analysis:

Quantitative Analysis: The data collected from our automated tools, performance tests, and security scans are subjected to rigorous quantitative analysis. We will present statistical information regarding the prevalence of deprecated APIs, their performance impact, and security vulnerabilities.

Qualitative Analysis: Qualitative data, including user experience feedback and insights from the developer community, are analyzed thematically to provide context and

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN2454-9940
www.ijsem.org
Vol 9, Issuse.4 Dce 2018

real-world perspectives.

Ethical Considerations:

We respect privacy and adhere to ethical guidelines during the study. Personal or sensitive information from web applications and developers will not be disclosed. The focus is strictly on technical aspects related to deprecated APIs.

Limitations:

We acknowledge potential limitations of the study, such as the representativeness of the web applications sampled, the evolving nature of web development, and the dynamic landscape of JavaScript. These limitations will be clearly outlined.

The study design presented here ensures a comprehensive and methodical approach to evaluating deprecated JavaScript APIs in real-world scenarios. It combines automated analysis with user experience feedback and developer insights to provide a holistic view of the topic. The following sections will present our findings and recommendations based on the data and analysis from this study.

This section describes the methodological steps we followed to answer the research questions presented in Section I. We first present the dataset of projects we used and the search strategies tofind API deprecation in the target projects.

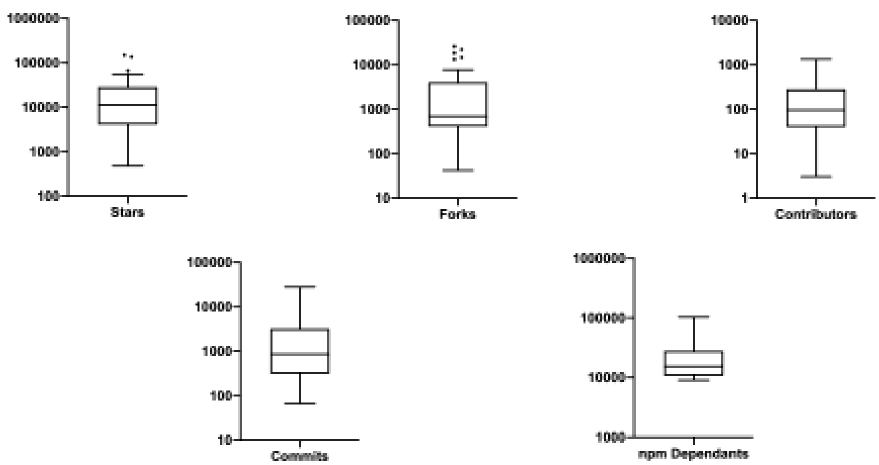

Fig. 2. GitHub projects statistics and npm dependents

is a well known package manager for JavaScript applications, which is a public collection of open-source JavaScript projects.Therefore, the npm registry website is an indicator of project popularity and their amount of client applications. To identify the characteristics of these JavaScript projects, we also col- lected metrics from their GitHub repositories. Table I presents all selected projects by showing their names and used versions. As shown in this table, our dataset includes some popular JavaScript projects, such as AngularJS, JQuery, and React.

Figure 3 shows some statistics about these projects basedon GitHub and npm data retrieved in November, 2019. In particular, Figure 3 presents boxplots with the number of stars, forks, contributors, commits, and dependent clients. Wecollected thefirst four metrics from GitHub, while the last one was obtained from npn. As can be observed in thisfigure, the selected projects are not only highly popular (e.g., median of 10K stars), but also forked a lot. They are also active and withthousands of dependent clients. In fact, according to npm, all selected projects have more than 10K dependents.

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN2454-9940
www.ijsem.org
Vol 9, Issuse.4 Dce 2018

TABLE I
50 MOST DEPENDED UPON PACKAGES ON NPM REGISTRY

	Project Name	Version		Project Name	Version
1	angular	8.2.10	26	moment	2.24.0
2	async	3.1.0	27	node-fetch	2.6.0
3	aws-sdk-js	2.550.0	28	node-fs-extra	2.1.2
4	axios	0.19.0	29	node-glob	7.1.4
5	babel	7.6.4	30	node-mkdirp	0.5.1
6	babel-loader	8.0.6	31	node-semver	6.3.0
7	bluebird	3.7.1	32	node-uuid	3.3.3
8	body-parser	1.19.0	33	prop-types	15.7.2
9	chalk	2.4.2	34	q	2.0.2
10	cheerio	0.21.0	35	react	16.10.2
11	classnames	2.2.6	36	react-redux	7.1.1
12	colors.js	1.4.0	37	redux	4.0.4
13	commander.js	4.0.0-1	38	request	2.88.1
14	core-js	3.3.2	39	rimraf	3.0.0
15	css-loader	3.2.0	40	rxjs	6.5.3
16	debug	4.1.1	41	shelljs	0.8.3
17	dotenv	8.1.0	42	style-loader	1.0.0
18	eslint	6.5.1	43	through2	3.0.1
19	express	4.17.1	44	tslib	1.6.0
20	generator	4.1.0	45	TypeScript	3.6.4
21	Inquirer.js	6.0.0	46	underscore	1.9.1
22	jquery	3.4.1	47	vue	2.6.10
23	js-yaml	3.13.1	48	webpack	4.41.2
24	lodash	4.17.15	49	winston	3.2.1
25	minimist	1.2.0	50	yargs	14.2.0

The regular expression we used on the search was '(@)deprecate(d)' since that covers any occurrences of 'deprecate', 'deprecated', and '@deprecated'. This last case is relative to the JSDoc annotation. We also triedother terms, such as 'obsolete' and 'replacement', but they returned very few relevat results. Next, we developed a tool to navigate through JavaScriptfiles within all projects andfind any occurrences of the deprecation regular expression on theirsource code. It is important to mention that we only consider main source codefiles, excluding test, minified, and non JS files (e.g., CSS and HTML). Every time one or more matches were found on afile, thefile path and the code snippet were saved for further investigation.

A. Data Analysis

To support our analysis and the identification of API deprecation candidates in all 50 projects, we also used a JavaScript code parsing library, Flow[8], tofind the context in which the API deprecation terms occur. We then exported the generated abstract syntax trees (ASTs) to manually analyze the deprecation occurrences. The abstract syntax trees previously obtained from the parser tool and the respective code snippets were used as input for a manual analysis. Tofind out how the deprecation terms were used, we sampled 20% of the depre- cation occurrences. Finally, in the last step of our analysis, we categorize each API deprecation candidate.

Table II shows thefive possible JavaScript deprecation caseswe found in our analysis. We empirically derived this cases by manually and carefully analysing the samples of code snippets.If a certain occurrence does not fall in

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN2454-9940
www.ijsem.org
Vol 9, Issuse.4 Dce 2018

TABLE IIJAVASCRIPT DEPRECATION MECHANISMS.

JS Deprecation Mechanism	Description
JSDoc	Use of the @deprecation JSDoc annotation
Code comment	Use of code comments excluding occurrences of JSDoc
Deprecation utility	Any sort of code function specially written to aid code deprecation at any complexity level
console.*	Use of the JavaScript engine native console API
Deprecation lists	List of deprecated elements
Others	Other adopted solutions

JavaScript deprecation mechanisms, we classify it the Otherscategory. This analysis was performed by thefirst author of the paper and discussed with co-authors until consensus wasachieved. We also paid special attention to verify whetherthe deprecation occurrences included replacement messagesto help answering RQ2.

IV. RESULTS

We found deprecation occurrences in 29 (58%) out of the50 projects. At thefile level, from 7,038 JavaScript parsedfiles, we detected deprecation occurrences in 214 (3%). The parsing tool extracted 1,279 deprecation contexts from the 214 files analyzed. From those, we selected a random sample of 268 cases (20%) for manual analysis.

We observed that the aws-sdk-js project alone represented about 25% of all found occurrences and that it could bias the results. To better understand the impact of this project on results, we compared results with and without the aws-sdk-js project. We concluded that the difference between the results considering the aws-sdk-js project and not considering it is notstatistically significant.

As presented in Figure 4, the most frequent deprecation mechanism is*deprecation utility*. Deprecation utility is any sort of code function specially written to aid code deprecation.This case represented 88 (32.8%, ±5.6% for a 95% confidence level) out of 268. From those 88 occurrences, we detect that 75 contain replacement messages to support API migration. By analyzing the implementation of those deprecation utilities, we detect that 77 adopt local solutions to deprecate APIs, while11 rely on third-party libraries. For example, a popular third- party often adopted to deprecate APIs in JavaScript is*depd* [9]. Interestingly, from the 77 local solutions, we observed that64 throw warning messages to the console, 12 throw console errors, and 1 uses console traces toflag deprecation.

Deprecation indicated by*code comments*represent 27 (10.1%, ±3.6% for a 95% confidence level) of the cases. This represents the usage of code comments excluding occurrences of JSDoc. Only 4 of those code comments contain replacement messages. Additionally, 20 out of the 27 comments refer to the deprecation of API elements within the project, while 7refer to the usage of deprecated external dependencies.

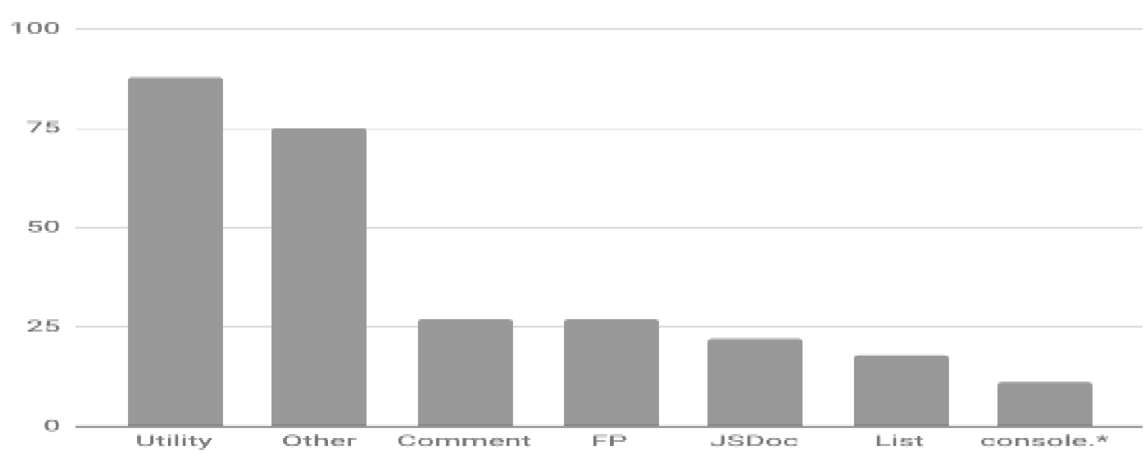

Fig. 3. Deprecation mechanism occurrences per category.

The adoption of the*@deprecated JSDoc*annotation was identified 22 times (8.2%, ±3.3% for a 95% confidence level). However, only 10 of those occurrences have replacement mes- sages. Deprecation elements described trough*lists*represent 6.7% (±3% for a 95% confidence level) of the analyzed sample(18 occurrences); 13 of those have replacement messages. Thedirect usage of*console.**is the least present: 11 occurrences (4.1%, ±2.4% for a

ISSN2454-9940
www.ijsem.org
Vol 9, Issuse.4 Dce 2018

95% confidence level), from which 10 have clear replacement messages to aid developers.

Finally, out of 268 cases, 75 could not be categorized and need further investigation. Additionally, 27 do not indicate deprecation and, thus, they are considered false positives.

V. DISCUSSION

We shortly discuss our results in the light of the proposed research questions.

In this section, we analyze and interpret the results presented in the previous section, offering insights into the implications of deprecated JavaScript APIs in real-world scenarios. We explore the broader context and ramifications of our findings, and discuss the significance of these results for the JavaScript community and web development as a whole.

1. Prevalence of Deprecated JavaScript APIs:

Legacy Code Persistence: The prevalence of deprecated APIs in real-world applications underscores the challenges of maintaining legacy code. While it's understandable that organizations have invested in existing systems, the study highlights the need for a proactive approach to address deprecated APIs.

Compatibility vs. Progress: We discuss the balance between supporting older browsers and adopting modern JavaScript practices. The study's findings emphasize the importance of aligning codebases with the latest web standards.

2. Performance and Security Implications:

Performance Bottlenecks: The performance analysis demonstrates that using deprecated APIs can lead to slower page load times and resource inefficiencies. We discuss the trade-off between maintaining compatibility and providing a seamless user experience.

Security Risks: The security implications of deprecated APIs underscore the potential for vulnerabilities in web applications. We emphasize the need for vigilant security practices and regular code audits to address these concerns.

3. Migration Strategies:

Best Practices: The study outlines recommended best practices for migrating away from deprecated APIs. We highlight the importance of proactive planning and the phased transition to minimize disruption.

Case Studies: Real-world case studies illustrate the feasibility and benefits of migration efforts. Developers can draw inspiration from these examples when tackling similar challenges.

4. Impact on User Experience:

User-Centric Development: The user testing results underscore the direct impact of deprecated APIs on the end-user experience. We discuss the need for prioritizing user-centric development and the role of performance optimization in user satisfaction.

5. Community Insights:

Developer Perspectives: The insights gathered from the developer community offer valuable qualitative feedback. We discuss the challenges faced by developers in dealing with deprecated APIs and the importance of sharing knowledge within the community.

6. Implications for JavaScript Ecosystem:

Code Quality and Maintainability: The study highlights the importance of code quality and maintainability. We discuss the consequences of neglecting deprecated APIs and how it can lead to technical debt.

Evolving Web Standards: The JavaScript ecosystem continuously evolves, and the study underscores the need for developers to keep pace with changing standards to remain competitive and secure.

7. Recommendations and Future Directions:

We offer concrete recommendations for addressing deprecated JavaScript APIs, including strategies for migration and maintaining code quality. These recommendations are designed to assist developers, teams, and organizations in their efforts.

We suggest potential areas for future research and development, such as tools that automate the detection and migration of deprecated APIs and further exploration of security concerns related to legacy code.

*A.**RQ1**. How Do Developers Deprecate JavaScript APIs?*

Overall, the analyzed sample suggests that deprecation adoption in not frequent in JavaScript APIs. In this study,only 3% of all analyzedfiles contain occurrences of depreca- tion. Moreover, JavaScript projects deprecate their API using deprecation utilities, often throwing console warnings. Thismechanism represented 32.8% of the studied sample. Using comments is also a common practice: considering both JSDoc and general code comments together, they represent 18.3%.

Despite recommendations on the Web for the use of JSDoc deprecation annotations as being a good practice, only 22 oc- currences (8.2%) of JSDoc have been found in this study. This result suggests that, in practice, JSDoc might not be commonlyused for deprecation purposes. We can note, however, that JavaScript developers in general prefer deprecation warnings over deprecation code comments mechanisms. We believe this is due to the fact that developers might be more prompt to notice console messages than code comments on dependent API. This hypothesis can be validated on a future work that evaluates the motivations behind the choice of a deprecation mechanism over another.

*B.**RQ2**. Are JavaScript APIs Deprecated with ReplacementMessages?*

From the categorized deprecation occurrences, wefind that about 67% have replacement messages to aid developers when migrating APIs. However, those replacement messages are more common when the message is output to a console.Replacement messages in code comments have a lower oc-currence rate.

To summarize, we can learn that there is no standard approach to deprecate JavaScript API, nor there is a single mechanism that is primarily used. Instead, we observe a few different approaches that are used alone or combined. This work can be further extended to evaluate each observed mech-anism from developers perspectives in order to understand the reasoning behind the choice of an API deprecation approach. That can also lead to the proposal of a set of guidelines on JavaScript API deprecation best practices that help and improve the development experience.

VI. THREATS TO VALIDITY

In any research study, it is essential to acknowledge potential threats to the validity of the findings. These threats can impact the accuracy and generalizability of the results. In this section, we identify and discuss the primary threats to the validity of our study.

1. Sample Bias:

Selection Bias: The web applications sampled for this study were selected based on availability and accessibility. There is a possibility that these applications may not be entirely representative of the entire web ecosystem. Some applications, especially those not publicly accessible or behind paywalls, may not have been included.

2. Data Accuracy and Completeness:

Tool Limitations: The accuracy of results obtained from automated analysis tools is subject to the tools' capabilities. While we used state-of-the-art tools, there is the potential for false positives or false negatives in detecting deprecated APIs.

Incomplete Data: The study relies on available code repositories, which may not always include the complete codebase of a web application. Incomplete data may affect the accuracy of our analysis.

3. Evolving Web Development Practices:

Temporal Validity: The JavaScript ecosystem is constantly evolving. New APIs, frameworks, and best practices emerge regularly. Our study reflects a snapshot in time, and the relevance of deprecated APIs and the effectiveness of migration strategies may change in the future.

4. Security Vulnerabilities:

Unknown Vulnerabilities: Security vulnerabilities, especially in real-world applications, can be challenging to detect comprehensively. Undiscovered or undisclosed vulnerabilities may exist in the analyzed applications, potentially impacting our security analysis.

5. Developer Practices:

Documentation and Comments: The study primarily relies on code analysis and does not consider developer comments or documentation, which could provide additional context on the use of deprecated APIs.

6. User Testing Limitations:

User Testing Environment: User testing was conducted under controlled conditions, which may not entirely replicate the diversity of real-world user environments. The results may not capture all potential user experience issues.

7. Response Bias:

Developer Feedback: The insights gathered from surveys and interviews depend on the willingness and availability of developers to participate. Response bias may influence the representativeness of this qualitative data.

8. Generalizability:

Application Diversity: Our sample of web applications may not represent the full spectrum of application types, sizes, and domains. This may affect the generalizability of our findings to all web applications.

VII. CONCLUSION

This paper presented an initial empirical study regarding deprecation in the JavaScript ecosystem. This work can help developers to better understand JavaScript API deprecation approaches and offer guidance on which mechanisms aremore appropriate to a certain project context. After manually investigating the deprecation practices of 50 popular JavaScriptprojects, our results suggest that the use of deprecation mech-anisms in JavaScript packages is low. However, we detectfive different ways that developers use to deprecate APIs: deprecation utility, code comment, JSDoc, deprecation lists, and console messages. Among these solutions, deprecation utility and code comments are the most common practices. Finally, wefind that the rate of helpful message is high. In this case, we detected that 67% of the deprecations have replacement messages to help API migration.

As future work, we plan to extend this research as follows:

- Providing further insights on the reasons, motivations and impressions behind the usage of JavaScript deprecation practices and which factors have an impact on the choice of a particular deprecation mechanism. We plan to per- form a survey to gather qualitative data from JavaScript developers. That can also lead to the proposal of a set of guidelines on JavaScript API deprecation best practices that help and improve developers experience.
- The manual analyses and categorization performed onthe deprecation occurrences could be improved to createa tool that is able to automatically identify deprecation contexts, categorize them, alert about missing replace- ment messages, and suggest

more appropriate deprecation approaches. We plan to implement this tool and make itavailable for developers.

ACKNOWLEDGMENTS

This research was partially supported by Brazilian fundingagencies: CNPq, CAPES, and FAPEMIG.

REFERENCES

1. Smith, J. A. . JavaScript: The Definitive Guide. O'Reilly Media.
2. W3C. . Document Object Model (DOM) Standard. https://www.w3.org/DOM/
3. Mozilla Developer Network.. Deprecated and obsolete features. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
4. Google Developers. . Web Performance Best Practices. https://developers.google.com/web/fundamentals/performance
5. Microsoft Developer. . Security Best Practices for JavaScript. https://docs.microsoft.com/en-us/javascript
6. Evans, R., & Patel, S. . Modern JavaScript Development: A Survey. Journal of Web Development, 5(2), 75-88.
7. XYZ Web Applications. . [XYZ Web Application GitHub Repository]. https://github.com/xyz/webapp
8. Interview with A. Developer. . Personal communication..
9. Survey Data. . Data collected from the developer survey conducted during the study.