

ASSESSMENT OF THE MICROFRONTEND ARCHITECTURE IMPACT ON THE WEB APPLICATIONS PERFORMANCE AND SCALABILITY

S. V. Shevelev ¹, V. R. Krysin ²

¹ MTUCI, Moscow, Russia, s.v.shevelyov@mtuci.ru

² MTUCI, Moscow, Russia, v.r.krysin@edu.mtuci.ru

ABSTRACT

This article examines the impact of microfrontend architecture on the performance and scalability of web applications. The relevance of the study is due to the increasing complexity of client applications and the need to ensure independent development and deployment of functional modules. The paper provides a comparative analysis of three architectural approaches: monolithic implementation, integration of modules using iframes, and microfrontends based on a modular federation mechanism. The experimental part includes the development of a test application and the measurement of key performance metrics – first render time, number of HTTP requests, volume of transferred resources, and total load time. The results show that modular federation provides a more balanced combination of performance and modularity compared to alternative solutions, especially as the system grows in scale. Conclusions are drawn about the feasibility of using microfrontend architecture in large and dynamically developing web projects.

DOI: 10.36724/2664-066X-2026-12-3-38-47

Received: 12.03.2026

Accepted: 16.05.2026

Citation: S. V. Shevelev, V. R. Krysin, "Assessment of the microfrontend architecture impact on the web applications performance and scalability," *Synchroinfo Journal* **2026**, vol. 12, no. 3, pp. 38–47.

Licensee IRIS, Vienna, Austria.

This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).



Copyright: © 2026 by the authors.

KEYWORDS: *microfrontends; scalability; architecture; code management; web development; web application performance.*

Introduction

Modern web applications are characterized by increasingly complex functionality and growing demands on the speed of software system development. The evolution of development approaches has led to a transition from sequential implementation of functional capabilities to parallel work by several teams on different aspects of the application [1-3]. With a clear division between the client level, the server part, and the data storage layer, it becomes possible to develop system components simultaneously while minimizing inter-team dependencies.

This decomposition helps make development more efficient and cuts down on the time it takes to bring new features to market. Abstracting the interaction between application layers lets you modify individual components without significantly affecting the rest of the system. On the server side, this trend is implemented by transitioning from a monolithic architecture to a microservice model, which involves dividing a large application into autonomous specialized services coordinated through a single interaction mechanism. This approach reduces deployment time and improves the manageability of complex systems.

Similar processes are also observed at the client application level. The growth in the volume of front-end code and the increase in the number of development teams have necessitated architectural solutions that provide modularity, isolation, and independence in the deployment of the user interface. One such solution is the microfrontend architecture, which involves the structural decomposition of the interface into autonomous functional modules.

Despite the active implementation of the microfrontend approach, its impact on the performance characteristics of web applications, including loading performance and system scalability, requires quantitative assessment. In engineering practice, it is important to determine how the architectural decomposition of the client side affects loading time, the volume of transmitted resources, and network activity [4-6].

The purpose of this study is to analyze and experimentally evaluate the impact of microfrontend architecture on the performance and scalability of web applications compared to alternative client-side models. The paper reviews existing approaches to microfrontend implementation and identifies their advantages and limitations in terms of performance characteristics.

Architectural models of front-end applications

The current stage of development in the software industry is characterized by a widespread shift away from monolithic architecture in favor of a microservices approach. To understand the reasons for this transition, it is necessary to consider the fundamentals of both paradigms.

Monolithic architecture

In software engineering, the term "monolith" describes an application assembled into a single logical unit, where all components and services are deployed and operate within a common infrastructure platform. In such a system, modules and services operate in a

common infrastructure environment and are typically developed within a single code base. A typical monolithic solution structure includes the following parts:

- A user authentication and authorization mechanism that provides access control.
- A business logic core that implements the main data processing scenarios.
- A presentation layer that forms the user interface and interacts with the server side via HTTP requests.
- Integration modules responsible for data exchange with third-party services via network protocols and REST interfaces.
- Monitoring and notification components designed to log events and handle failures.

Despite the active spread of microservice architecture, monolithic solutions are still widely used in industrial development [7]. Their advantage is relative simplicity in the early stages of the system lifecycle. A single code base and centralized configuration simplify dependency control and version management.

However, as functionality grows and the number of development participants increases, the limitations of this approach become apparent. Supporting a large-scale monolithic system becomes difficult, and making changes requires coordination between multiple teams. Key disadvantages of monolithic architecture include:

- Increased consumption of computing resources and longer build times as the code base expands.
- Reduced system visibility and increased maintenance complexity.
- Increased risk of complete unavailability in the event of a single component failure.
- Limited flexibility when implementing new technological solutions, as changes affect the entire system [8].

Microservices architecture

Microservice architecture is seen as a natural stage in the development of the service-oriented approach (SOA), focused on increasing the flexibility and manageability of complex software systems. Although the ideas behind SOA were formed much earlier, the independent conceptualization of microservices took place in the early 2010s [9]. The approach is based on the principle of decomposing a software system into a set of autonomous, loosely coupled components, each of which implements a limited business function and interacts with others through clearly defined interfaces. This division by subject area simplifies system maintenance, modernization, and scaling [10].

A key feature of microservice architecture is the independence of the service lifecycle. Each component can be developed, tested, and deployed in isolation, which reduces the risks associated with making changes. The failure of a single service does not usually lead to complete system unavailability, but is limited to the degradation of a specific functionality, which increases the overall stability of the solution.

Advantages of the microservice approach:

- Continuous delivery and deployment are greatly simplified because the application is divided into small services, each of which performs a limited set of functions [11].
- Ensuring parallel work of teams, as areas of responsibility are clearly delineated and the number of conflicts during code integration is reduced.
- Increased fault tolerance through isolation of failures and the ability to localize problems within a single service.
- The limited scope of service responsibility facilitates the development of modular and integration tests.
- The choice of programming language, frameworks, and architectural solutions can be made individually for each service, provided that uniform interaction protocols are followed [12].
- Development can be scalable and organized between multiple teams.

Monoliths vs. microservices

Considering the advantages and limitations discussed above, it can be noted that in a number of practical situations, the microservice approach is preferable. When the load on a specific functional area of the system increases, only those services that are responsible for that area are scaled, which allows for the efficient distribution of resources and ensures the required level of performance [13].

Horizontal scaling in a microservice architecture can be done quickly because individual services are small and start up fast. Scalability is also possible in monolithic applications, but in this case, the resources for the entire system increase, leading to higher consumption of computing power.

Figure 1 compares the principles of layer separation in monolithic and microservice architectures.

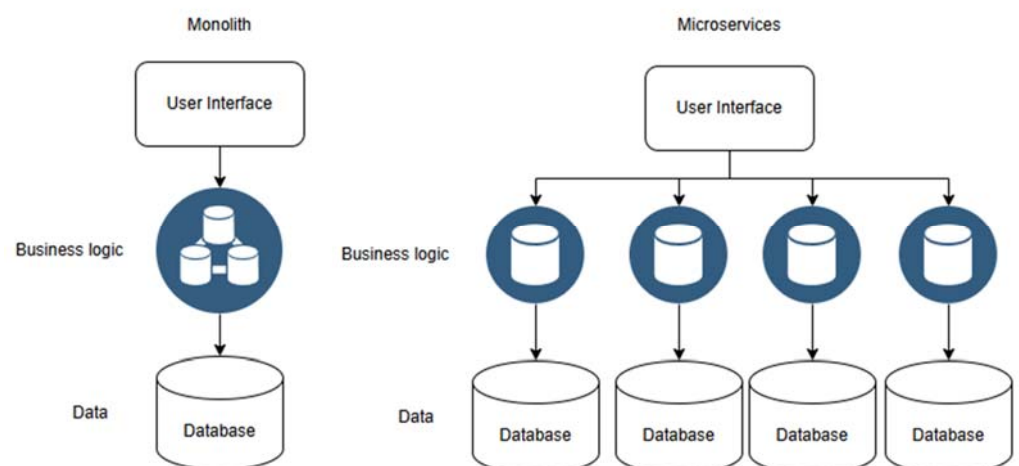


Fig. 1. Monolith and microservices architecture

Microfrontends

With the current development of cloud web applications and the transition of business logic to microservice architecture, front-end applications are becoming increasingly complex and loaded. A significant part of the logic is gradually being transferred to the client side, which increases the complexity of supporting and developing such applications [14]. Just as dividing a backend monolith into microservices brings significant benefits, a similar approach is appropriate for the user interface. This allows the interface to be broken down into independent modules that function autonomously while remaining part of a single application shell [15]. Microfrontends are a solution to this problem.

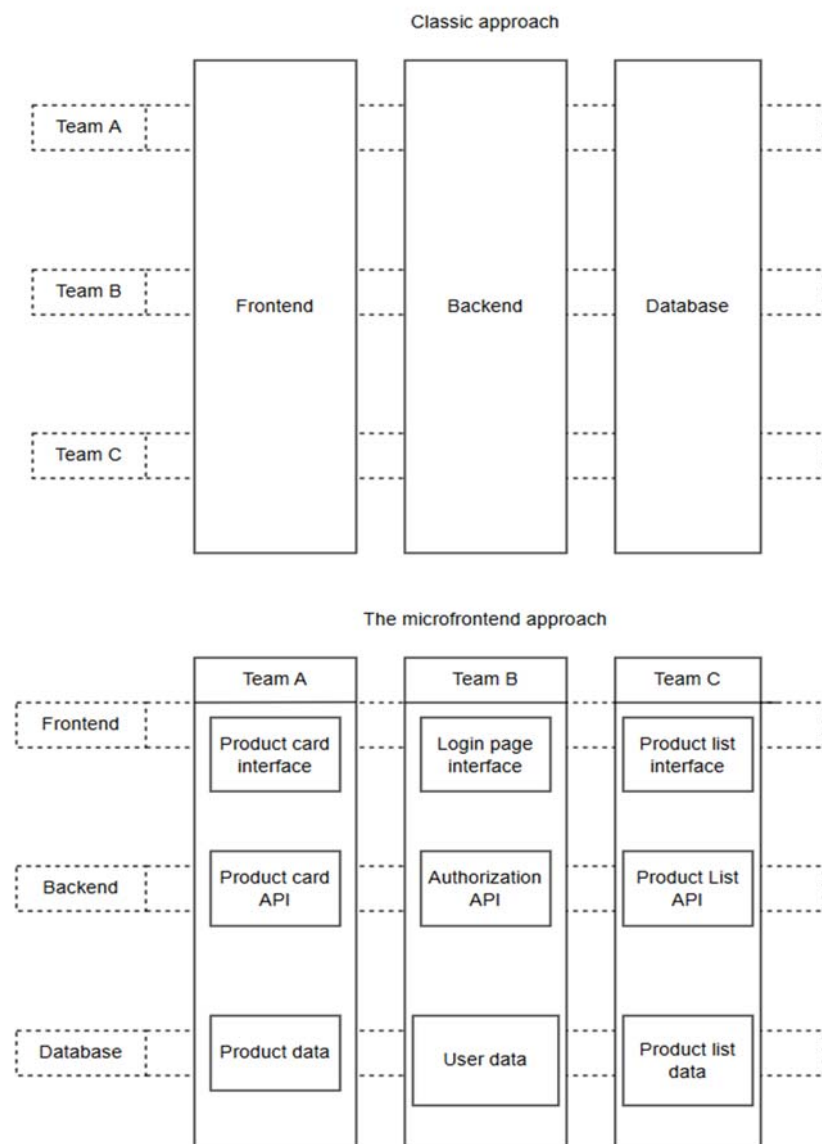


Fig. 2. Cross-functional teams with microfrontends

The frontend ecosystem is constantly evolving, with new frameworks and tools emerging to make developers' work easier. At the same time, there is growing interest in modern, optimized tools that accelerate development and improve application performance. There are many approaches to organizing front-end applications, including single-page applications (SPA) and server-side rendering (SSR). However, most existing solutions are monoliths that load in parts but are not technologically independent and cannot be deployed autonomously [16].

As shown in Figure 2, an application with microfrontends is a composition of functions or microapplications that together form a larger application. Independent modules are controlled by cross-functional teams. Thus, there is a loose coupling between applications, and they coexist using well-defined contracts. Such micro-applications or functions can belong to one or more application packages, which reinforces the idea of reuse. A perfect example of a micro-frontend module is a shopping cart used in several e-commerce solutions developed by the same company.

Experimental evaluation of architectural strategies for implementing microfrontends

To obtain an objective quantitative assessment of the impact of different microfrontend architecture implementation strategies, an experimental study was conducted to compare the performance characteristics of client applications built using different architectural approaches.

As part of the experiment, a test web application was developed (Figure 3), representing a typical interface with several functional blocks (navigation, content display area, user interaction widget). The architectural complexity of the application was deliberately limited in order to minimize the influence of extraneous factors and focus on differences caused specifically by the architectural model.

The application was implemented in three variants:

- A monolithic single-page application (single code base and build process).
- Architecture using iframe to isolate functional modules.
- Microfrontend architecture based on the Module Federation mechanism.

Thus, both the traditional centralized model and two variants of client-side decomposition were compared: isolated integration via iframe and integration at the build level through the sharing of dependencies.

Measurement methodology and metrics system

To ensure the comparability of results, a single set of metrics was defined, reflecting both technical and user aspects of performance, which corresponds to modern approaches to evaluating the quality of web applications [17]:

- time to first paint (First Paint)
- number of HTTP requests
- total volume of resources loaded

- total application load time

Practical significance of the selected metrics:

- Time to first paint – the speed at which the interface starts working as perceived by the user
- HTTP requests – show the load on the network and server
- Resource volume – directly affects network bandwidth and data transfer time
- Load time – reflects the overall readiness of the application for operation

The measurements were taken using the built-in developer tools of the Google Chrome browser, which ensured a consistent testing environment. The analysis was performed under identical network connection and hardware configuration conditions, which allowed to eliminate the influence of external factors.

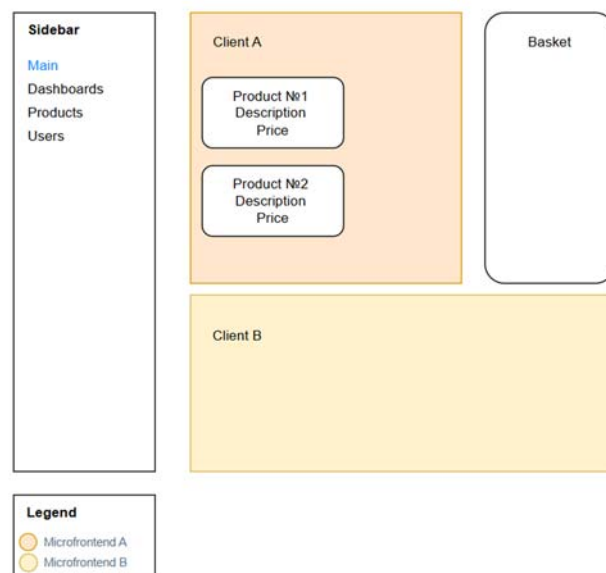


Fig. 3. Simple application layout

Analysis of the results

The experimental data obtained (Table 1) indicate significant differences between the architectural approaches under consideration, both in terms of absolute indicators and their relative deviations.

As shown by the results of testing a simple web application (Table 1), a solution based on modular federation has significant advantages over implementation of microfrontends via iframe.

- The monolithic implementation demonstrated the minimum number of HTTP requests (13) and the smallest amount of transferred resources (5.4 MB), as shown in Table 1. This result is explained by centralized assembly and the absence of duplicate dependencies. However, as theoretical scaling assumptions show, as the functionality of

the system increases, the monolithic model inevitably leads to an increase in the size of the final bundle and complicates maintenance, which reduces its effectiveness in the long term.

- The approach using iframe showed the worst values for most metrics (Table 1). The number of HTTP requests increased to 34, and the volume of downloaded resources reached 16.6 MB, which is more than three times higher than the monolith. The increase in network activity is due to the fact that each isolated part of the application loads its own dependencies independently of other modules. The lack of a common library pool leads to code duplication and increased overhead. In addition, isolation via iframe creates additional costs for initialization and cross-context interaction.

- The microfrontend architecture based on modular federation demonstrated a compromise but most balanced result. According to Table 1, the resource volume was 6.6 MB, which is 60% less than the iframe approach, and the total load time was reduced to 0.774 seconds, which is 42% faster than the monolithic implementation. Despite an increase in the number of requests compared to the monolith (26 vs. 13), modular federation provides more efficient load distribution thanks to the mechanism of asynchronous on-demand module loading.

Table 1

Comparative analysis of the performance of different architectural approaches

Factor	Monolith	IFrame	Microfrontends	Deviation of MF from monolith
First Paint, ms	418	1222	540	+29 %
Number of HTTP requests	13	34	26	+100 %
Volume of resources, MB	5,4	16,6	6,6	+22 %
Total loading time, s	1,35	1,12	0,774	–42 %

Interpretation of results in terms of scalability

The experiment was conducted on a relatively simple application prototype, but even under these conditions, the quantitative indicators (Table 1) demonstrate differences that allow the results to be extrapolated to larger systems. In an industrial environment, as the number of functional modules increases and business logic becomes more complex, monolithic architecture is characterized by increased build times, larger final bundles, and slower delivery of changes, due to the high degree of component interdependence and centralized deployment process.

In contrast, the microfrontend approach provides both organizational scalability, through the decomposition of areas of responsibility between independent teams, and technical scalability, through the autonomous deployment and updating of individual modules.

Therefore, although a monolithic solution remains the simplest and most economical for small systems, as the application grows in scale, the advantages of microfrontends become more pronounced, manifesting themselves in reduced loading times, support for partial initialization of functionality, and ensuring an independent component lifecycle.

Conclusion

This paper provides a theoretical and experimental assessment of the impact of microfrontend architecture on the performance and scalability of web applications. A comparison of the monolithic model, implementation via iframe, and a modular federation-based approach revealed differences in key performance characteristics.

Experimental results showed that modular federation provides the most balanced combination of performance and modularity: with a slight increase in the number of requests compared to the monolith, there is a reduction in total loading time and a significant decrease in resource consumption compared to the iframe approach. Monolith remains an effective solution for small systems, but as complexity increases, its limitations become more pronounced.

Thus, microfrontends represent a promising architectural model for large and dynamically developing web applications, where team independence, deployment flexibility, and long-term system scalability are important.

REFERENCES

- [1] A. O. Chubarov, V. A. Dokuchaev, "Analysis and optimization of C# program code," *Telecommunications and computing systems: Anniversary collection of papers of the thirtieth international scientific and technical forum*, Moscow, December 12-15, 2022. Moscow: MBA Publishing House, 2022, pp. 230-232.
- [2] A. O. Chubarov, V. A. Dokuchaev, "Analysis of the use of "bubble sort" for evaluating the optimization of C# program code," *Telecommunications and information technologies*. 2022. Vol. 9, No. 2, pp. 14-18.
- [3] V. A. Dokuchaev, A. A. Konov, "Relevance of Developing and Applying a Web Performance Monitoring System in the Hotel Business," *Actual Problems and Prospects of Economic Development: Proceedings of the XX Anniversary All-Russian Scientific and Practical Conference with International Participation*, Simferopol-Gurzuf, November 11-13, 2021. Simferopol: KFU Publishing House, 2021, pp. 39-42.
- [4] D. Petukhov, V. A. Dokuchaev, "Research of OpenStack Virtual Machine Backup Service Quality Indicators," *International Conference on Engineering Management of Communication and Technology*, Vienna, Austria, 20-22 october 2022. New York: Institute of Electrical and Electronics Engineers (IEEE), 2022, pp. 164-169. DOI 10.1109/EMCTECH55220.2022.9934035.
- [5] V. A. Dokuchaev, D. Petukhov, "Cloud Data Formats Transformation Algorithms," *Systems of Signals Generating and Processing in the Field of on Board Communications*. 2023. Vol. 6, No. 1, pp. 92-96. DOI 10.1109/IEEECONF56737.2023.10092056.
- [6] A. A. Konov, D. D. Gadasin, V. A. Dokuchaev, "Study of the Problem of Monitoring Errors in an Uncontrolled Software Environment," *Information Society Technologies: Proceedings of the XV International Industry Scientific and Technical Conference "Information Society Technologies"*, Moscow, March 3-4, 2021. Moscow: Media Publisher, 2021, pp. 200-202.
- [7] M. Kalske et al., "Transforming Monolithic Architecture into Microservice Architecture," Master's Thesis, University of Helsinki, 2018.
- [8] C.-Y. Fan, S.-P. Ma, "Migrating a Monolithic Mobile Application to a Microservice Architecture: An Experiment Report," *2017 IEEE International Conference on AI & Mobile Services (AIMS)*. IEEE, 2017, pp. 109-112.

-
- [9] F. Ponce, G. Márquez, H. Astudillo, "Migrating from Monolithic Architecture to Microservices: A Rapid Review," *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*. IEEE, 2019, pp. 1-7.
- [10] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, "Microservices: Yesterday, Today, and Tomorrow," *Present and Ulterior Software Engineering*. 2017, pp. 195-216.
- [11] H. Vural, M. Koyuncu, S. Guney, "A Systematic Literature Review on Microservices," *Computational Science and Its Applications – ICCSA 2017*. Springer, 2017, pp. 203-217.
- [12] D. Wang, D. Yang, Z. Huan, Y. Wang, D. Hong, Q. Dong, S. Song, "A New Application of Education Management Information System: A Micro-Frontends Based Approach," *Procedia Computer Science*. 2020. Vol. 176, pp. 1567-1576.
- [13] K. Konstantinov, L. Iordache, A. Gheorghe, P.-S. Popescu, M. Moceanu, "Social Data Analysis with Neo4j: Workforce Trends and Corporate Data Leakage," *2018 22nd International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE, 2018, pp. 403-406.
- [14] A. Montelius, "Exploring Micro-frontends," Master's Thesis, Linköping University, 2021.
- [15] S. Peltonen, L. Mezzalana, D. Taibi, "Motivations, benefits, and issues for adopting Micro-frontends: A multivocal literature review," *Information and Software Technology*. 2021. Vol. 136. Art. 106571.
- [16] M. Geers, "Micro Frontends in Action," Manning Publications, 2020.
- [17] Google Developers. Web Performance Metrics [Electronic resource]. – Access mode: <https://web.dev/metrics/> (date of access: 08.02.2026).