MICROSERVICE ARCHITECTURE IN FRONTEND DEVELOPMENT

Dudak A.
specialist degree
Tomsk State University of Control Systems and Radioelectronics
Russia, Tomsk
https://doi.org/10.5281/zenodo.14236033

Abstract
This article explores the use of microservice architecture in frontend development. It examines core principles and benefits of micro frontends, such as scalability, modularity, and flexibility, as well as the challenges of implementation, including integration, dependency management, and security. Special focus is given to system support and monitoring issues to ensure high performance and fault tolerance in high-growth applications. To further support scalability and reliability in dynamic market environments, the article highlights essential practices in integrating microservice-based frontends for enhanced adaptability to evolving technical and business requirements.

Keywords: microfrontends, microservices, scalability, modularity, web development.

Introduction

Modern web applications have become critically important tools for businesses, enabling user interaction across a variety of devices and platforms. As data volume, functionality, and user base grow, the need for flexible, scalable, and easily maintainable systems has emerged. One promising approach that addresses these needs is microservice architecture, where an application is structured as a set of independent services, each performing a specific task. Although microservice architecture was initially applied in backend development, its principles are now successfully integrated into frontend development, creating so-called micro frontends.

Micro frontends allow the frontend to be divided into autonomous modules that can be developed and maintained independently. Unlike traditional monolithic architectures, micro frontends offer a higher level of modularity, which is especially valuable for large teams working on extensive applications. This separation accelerates the development process, enables updates of individual components without reworking the entire application, and allows for more flexible system scaling. The application of micro frontends makes the system more resilient to potential failures and adaptable to changing business requirements.

The purpose of this article is to analyze the advantages and limitations of microservice architecture in web development, as well as to identify specific constraints and features. The article will examine how micro frontends contribute to maintaining flexibility and resilience in the system, the challenges they present, and the scenarios in which they are most effective.

Main part. Fundamentals and benefits of microservice architecture and micro frontends

The rapid growth of internet users and the need to ensure the reliable operation of web services require developers to continuously improve the performance and fault tolerance of their applications. According to Statista, the total number of internet users reached 5.4 billion in 2023 (fig. 1).

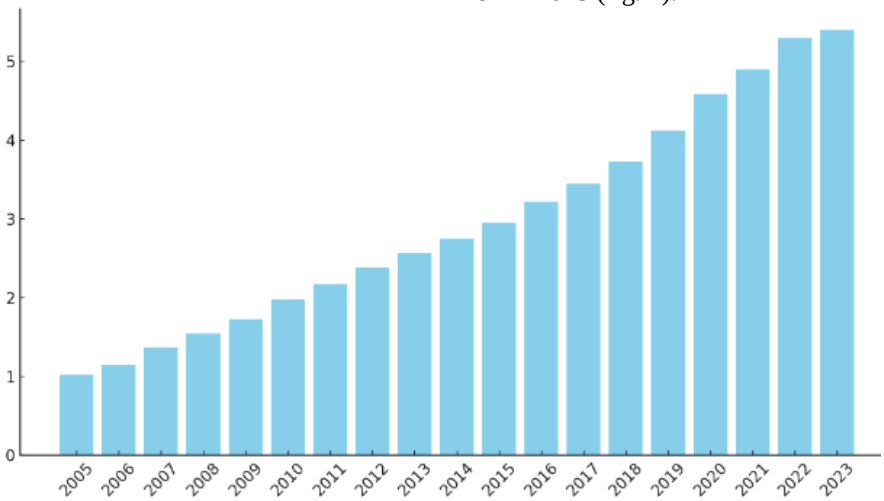

Figure 1. Number of internet users worldwide from 2005 to 2023, millions [1]

To create reliable and high-performing web resources, microservice architecture is often used. This technology represents an approach to application development where a system consists of separate, independent components that interact through well-defined interfaces. Each microservice is responsible for a specific functional area and can be developed, deployed, and scaled autonomously. In the field of frontend development, these principles have led to the concept of micro frontends, where each user interface component functions as a separate microservice with its own dependencies and infrastructure [2].

One of the main advantages of the microservice approach is its **scalability.** Unlike monolithic applications, where scaling requires increasing resources for the entire system, microservice architecture allows only those parts of the application that need it to be scaled. Specifically, in conditions of significant user growth and high load, micro frontends enable scaling of only critical services (e.g., search systems or shopping carts in an online store) without increasing the load on other parts of the interface [3].

The technology provides a **high level of modularity**, enabling the creation of independent components that can be reused in other projects. Thanks to this principle, developer teams can work in parallel on different parts of the system without interfering with each other. This is especially important for distributed teams, where developers may be in different time zones or regions. It also allows such teams to work on different components of the system simultaneously, minimizing interdependence. In the context of remote work and global team distribution, this becomes an essential factor for project success. Each team is responsible for its functional area, reducing the likelihood of conflicts and code duplication.

The architecture offers developers a high degree of flexibility in choosing technologies and approaches. Different micro frontends can use various technologies, programming languages, and even frameworks. For example, one part of the interface may be written in React, another in Vue or Angular, allowing optimal tools to be selected for specific tasks.

This flexibility also extends to the deployment process. Since each microservice is autonomous, teams can deploy new versions of individual micro frontends independently of each other, avoiding delays associated with synchronous updates across the entire system. This is particularly valuable in a dynamic market where the speed of product release can become a competitive advantage.

An example of successful micro frontend implementation is the American company **Netflix**, which has adopted microservice architecture for its user interfaces. Here, each micro frontend is responsible for a specific part of the platform, such as recommendations, film descriptions, or viewing. These micro frontends can be independently updated and scaled according to load, increasing system resilience and enhancing user experience [4].

Challenges and limitations of microservice architecture

Implementing microservice architecture in frontend development unlocks many possibilities, but it also comes with various challenges and limitations, requiring careful planning and strategic approaches. Understanding these challenges helps developer teams prepare for potential difficulties and develop strategies to use micro frontends effectively in projects (table 1).

Table 1.

Advantages and disadvantages of micro frontends [5]

Aspect	Advantages	Disadvantages
Scalability	Ability to scale individual modules as needed without affecting other parts of the system.	Increased network requests between micro frontends can negatively impact performance.
Modularity	High level of module independence enables easy updating and reusing components across projects.	Developing a dependency management system between modules can be complex and requires extra effort.
Development speed	Different teams can work on their micro frontends in parallel, speeding up development and deployment.	Due to integration and testing complexities, overall development speed may decrease during initial adoption.
Integration	Flexibility in choosing technologies and tools allows optimal solutions for each team or functionality.	Integrating components requires additional effort and often involves special frameworks (e.g., Single-SPA, Module Federation).
Security	Centralized access control and data protection for individual modules using technologies like OAuth and JWT.	Increased interaction points between micro frontends raise the risk of data leaks and require extra measures for authentication and authorization.
Dependency management	Allows independent library and framework updates for each micro frontend, reducing compatibility risks during updates.	Versioning and compatibility issues among modules, especially in large applications with many independent micro frontends.

Dividing an application into independent micro frontends **increases integration and management** complexity, as each component may have its dependencies and require coordinated interaction with other system parts. In a monolithic approach, interaction between interface parts is relatively straightforward because all modules reside in a single application. In micro frontend architecture, each component can have its dependencies and libraries, necessitating the development of an additional layer of integration and synchronization.

One way to address this issue is by **using specialized frameworks** like Single-SPA or Module Federation in Webpack (a module bundler for JavaScript). Single-SPA enables each micro frontend to operate independently, allowing them to be loaded and executed autonomously. This approach allows adding, updating, or removing individual micro frontends without the

need to recompile or restart the entire application, accelerating development and deployment. The Module Federation tool enables component sharing across micro frontends, allowing multiple teams to work independently on different system parts. It also helps manage library versions and prevents conflicts, ensuring compatibility among different parts of the application.

Microservice architecture can also **increase overhead for interactions between modules**. Since each micro frontend operates independently, requests between them may lead to delays and performance degradation across the system. For instance, for large interfaces with multiple interacting micro frontends, developing additional caching and request optimization mechanisms may be necessary to minimize latency [6]. Addressing performance issues requires managing network requests between microservices and efficiently using caching. These measures can significantly reduce server load and provide acceptable response times for users, but they also increase development complexity and require additional resources.

Security becomes more complex when using micro frontends. Each micro frontend interacts with other components and may access data that requires protection. In the case of data leakage or authorization failure, the entire system can be put at risk.

Ensuring security in a microservice architecture requires a centralized approach to user authorization and authentication. One solution could be using technologies like OAuth and JSON Web Token (JWT), which allow centralized authorization to prevent unauthorized access to microservices. However, security issues remain relevant, especially in distributed teams and when using third-party services.

Support and maintenance of microservice architecture

One of the most critical tasks in maintaining a microservice architecture is monitoring and diagnosing performance. Given the multiple microservices that operate independently, it's essential to monitor each component's performance and manage their interactions to avoid bottlenecks. Tools like Prometheus and Grafana are often used for this purpose [7].

Prometheus is an open-source monitoring system designed for metric collection and processing. In a microservice architecture, Prometheus tracks each component's status by collecting data on resource usage, latency, request count, and errors. This data can be used for graphing, trend analysis, and identifying bottlenecks.

Grafana complements Prometheus by providing powerful visualization capabilities for the collected metrics. With Grafana, customizable dashboards display real-time load and component statuses, enabling rapid detection and response to emerging issues, which is especially useful for monitoring micro frontends where each component may have specific metrics. Both tools support creating alerts based on metrics, allowing teams to respond quickly to potential failures.

Monitoring and diagnostics are essential for identifying and resolving performance issues, especially under heavy loads. Effective use of these tools helps prevent outages and maintain high-quality user service.

A key element in microservice architecture is fault tolerance, as the failure of one component should not cause the entire system to crash. To ensure fault tolerance in micro frontends, replication, resilience patterns, and distributed logging and tracing are often applied.

Service replication across multiple servers or nodes distributes load and prevents data loss if one node fails. This is crucial for high-traffic systems where even brief downtimes can lead to serious consequences.

In microservice architecture, patterns like Circuit Breaker and Retry are used. The Circuit Breaker pattern prevents cascading failures by disconnecting from a component if it frequently returns errors, protecting other microservices from being overloaded by repeated requests. The Retry pattern automatically retries service requests if it's temporarily unavailable, helping restore connectivity without user intervention.

With micro frontends, where each component may be managed by different teams and developed in various programming languages, centralized log management is crucial. Using tools like the ELK Stack (Elasticsearch, Logstash, Kibana) and Jaeger for distributed tracing enables quick identification of problem sources and diagnostics. These tools help track request chains across various microservices and detect failures in any system component.

Maintaining micro frontends requires managing versions and dependencies between components. When changing one microservice, compatibility with others must be ensured, which can be challenging. In a large system where each component may update frequently, this creates additional complexities [8].

To address compatibility among different versions as each microservice evolves independently, API versioning is used, enabling individual microservices to update without requiring all components to update. This is achieved by creating API versions that can coexist, supporting multiple components.

Version management and dependency management also involve integrating microservices into a CI/CD pipeline, automating deployment and testing processes. Tools like Jenkins, GitLab CI, or CircleCI are typically used to automatically test and deploy new versions of micro frontends. Integration of these tools minimizes update errors, as each new version undergoes automated testing before reaching production.

Canary testing and A/B testing should also be mentioned. These approaches allow deploying a new microservice version for a limited group of users to test its stability and performance in real-world conditions. Canary deployment helps detect errors and production issues early on, before the update becomes available to all users. This significantly reduces risks and allows evaluating the impact of changes on system performance.

Conclusion

Microservice architecture in frontend development, despite its complexities, is a powerful tool for creating scalable, flexible, and resilient systems, especially for applications experiencing rapid growth and meeting complex modern business demands. Dividing the user interface into micro frontends allows distributed teams to develop, test, and deploy independent modules in parallel, accelerating update rollouts and

minimizing the risk of dependency-related errors between application parts.

Ultimately, micro frontends are especially valuable for large projects and distributed teams, allowing approaches that ensure system adaptability to growing and changing requirements. A balanced approach to managing the complexities and benefits of micro frontends will enable the creation of a productive and flexible application that can withstand high loads and changing market conditions.

References:

1. Number of internet users worldwide from 2005 to 2023 // Statista // URL: https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/ (date of application: 31.10.2024).

2. Muhammad H. From Monolith to Microservice: Measuring Architecture Maintainability // International Journal of Advanced Computer Science and Applications (IJACSA). – 2023. – Vol. 14. – № 5. – P. 857-866.

3. Aluev A. Scalable web applications: a cost-effectiveness study using microservice architecture // Cold Science. – 2024. – № 8. – P. 32-38.

4. Amareen S., Dector O.S., Dado A., Bosu A. GraphQL Adoption and Challenges: Community-Driven Insights from StackOverflow Discussions // arXiv preprint arXiv:2408.08363. – 2024.

5. Hidayat D.C., Atmaja I.K.J., Sarasvananda I.B.G. Analysis and Comparison of Micro Frontend and Monolithic Architecture for Web Applications // Jurnal Galaksi. – 2024. – Vol. 1. – № 2. – P. 92-100.

6. Sidorov D. A comparative analysis of component-based architectures in web design for scalable applications // Cold Science. – 2024. – № 8. – P. 24-31.

7. Rafik T. On the maintenance support for microservice-based systems through the specification and the detection of microservice antipatterns // Journal of Systems and Software. – 2023. – Vol. 204.

8. Mozharovskii E. Evaluating retrieval-augmented generation (RAG) techniques in enhancing LMS for coding tasks // Universum: tekhnicheskie nauki : elektron. nauchn. zhurn. – 2024. – Vol. 6(123). URL: https://7universum.com/ru/tech/archive/item/17730 (date of application: 31.10.2024).