

OPTIMIZATION OF THE SOFTWARE DEVELOPMENT CYCLE USING MICRO FRONTEND ARCHITECTURE WITHIN LARGE ORGANIZATIONS

Dženita Vejsilović^{1,3}, Nermin Sarajlić²

Abstract: In today's fast-paced software industry, large organizations are under constant pressure to deliver features faster, scale development teams efficiently, and maintain high code quality. Traditional monolithic frontend architectures often become bottlenecks in the development and delivery of complex applications. The paper explores micro frontend architecture as a modern approach to optimizing software development processes, particularly within large and distributed teams. Micro frontend architecture decomposes the user interface into smaller, independent modules developed, tested, and deployed by separate teams. This approach fosters team autonomy, enables parallel development, simplifies maintenance, and allows for the use of diverse technologies within a single application. Through an in-depth analysis of its benefits and challenges, and a case study involving the implementation of a simple Online Shop application, this demonstrates how micro frontends can reduce complexity, increase efficiency, and accelerate development cycles. The paper demonstrates that while the implementation of micro frontends requires careful planning and organizational discipline, it offers a powerful solution for optimizing software development processes in large-scale environments.

Keywords: Micro frontends, large-scale environments, optimisation, software development process

INTRODUCTION

Digital technologies have become an essential part of modern businesses, especially within large companies which need advanced software for their daily operations. Software solutions that were adequate a few years ago, often no longer meet business needs, which requires their modernization and functionality expansion to support the company's further growth.

The core of any quality software solution is its architecture, which defines the way the software is created and how it functions. Over time, it has been shown that maintaining an outdated architecture can slow down the development process and company's flexibility, especially when the number of development team members increases. That is why it is important to enable teamwork by dividing tasks into smaller units, where each team develops its own part of the application with minimal need for constant coordination with other teams.

Initial application development usually involves the use of a monolithic architecture, where the entire software solution is unified by a single application that is responsible for database management, data processing, and the presentation layer that the end user sees.

The first step towards improving and scaling applications was the multi-tier architecture model (Figure 1). This

model allows application layers to be developed and scaled independently of each other. One part of the application (backend) is responsible for communicating with the database and data processing, while the other part (frontend) manages the presentation layer. This model has brought numerous advantages to the software development process. However, as software complexity grows, this architecture can make the daily work and progress of different teams more difficult, requiring frequent communication and coordination.

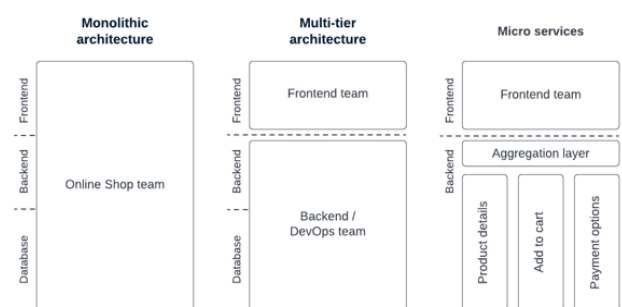


Figure 1: Overview of the monolithic, multi-tier and micro service architecture

To address scaling issues, the backend has been improved through the concepts of microservice architecture. In this architecture, parts of the data processing system are divided into independent services, however, there is still a limitation because the frontend is most often implemented as a monolithic application, which prevents its scaling and

¹ Glide Tech d.o.o, Bosnia and Herzegovina

² Faculty of Electrical Engineering, Bosnia and Herzegovina

³ Correspondence email: dzenita.vejsilovic@fet.ba

© 2026 Author(s). This is an open access article licensed under the Creative Commons Attribution License 4.0. (<http://creativecommons.org/licenses/by/4.0/>).

limits the work of multiple teams on the user interface. A solution to these challenges is the micro frontend (MFE) architecture, which aims to divide the application into smaller parts based on related functionality, allowing organizations to distribute work across teams and therefore accelerate software delivery (Figure 2). [6]

This architectural approach has gained significant attention in both industry and academia as organizations seek to address the limitations of monolithic frontend systems [12][13].

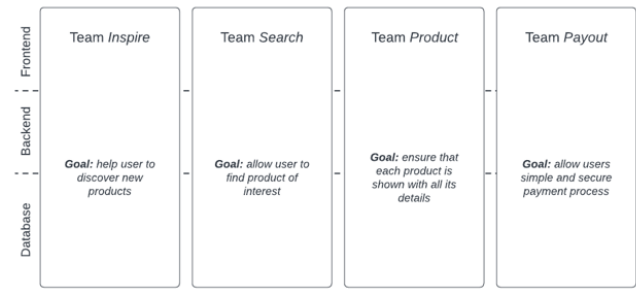


Figure 2: Different system functionalities organized based on the concept of micro frontend architecture [6]

Before choosing a software architecture, it is necessary to analyse the technical requirements, organizational and team structure, project life cycle, and delivery timeframe.

Micro frontend architecture brings several advantages such as faster software upgrades, simpler code solutions, the ability to independently develop individual functionalities, and the formation of autonomous teams that operate independently. The teams decide on the implementation method themselves and focus on the overall user experience. As a result, development is accelerated, and the quality of the software product is improved.

The aim of this paper is to investigate how micro frontend architecture contributes to the improvement of software development in large organizations and how it affects the internal organization and structure of development teams. Special focus is placed on improving the delivery process and team flexibility through this modern architecture.

1. MICRO FRONTENDS ARCHITECTURE

1.1. Micro frontend applications

Micro frontend architecture is a new type of architecture based on the concepts of microservice architecture (Figure 3). This architecture can be implemented in a way where frontend application is divided into smaller units, which is then combined with any backend architecture (monolithic or micro services) or throughout end-to-end concept where each micro frontend has its own backend.

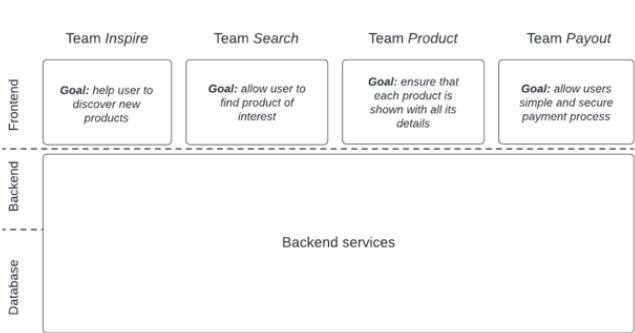


Figure 3: The concept of micro frontend architecture

Like microservice architecture, micro frontends simplify business logic and reduces the complexity of the code solution. However, they also bring certain challenges, such as the need for process automation, real-time monitoring, additional communication between teams, etc.

The theoretical foundations of micro frontends build upon established principles of software modularity and separation of concerns [14].

1.2. MVP and monolithic architecture

Many of today's websites are developed by small teams using monolithic architectures, where all the parts of code are connected and dependent on each other. In these applications, the initial effort is invested in creating reusable components, and over time the code base becomes more complex and more difficult to maintain. As projects grow, monolithic architecture slows down the delivery process and increases the risk of introducing errors into production. These are the reasons why many organizations are moving to a more modular approach through microservices.

When starting a work on a new project that offers an online service, the first iteration of application development should provide an answer to the key question: can the business idea succeed or not? The process begins with selecting technologies and services needed to create the first version of the application, with the preference given to the tools that are already familiar to team members. Eliminating unknown factors (such as technology) allows the team to focus entirely on building a minimum viable product (MVP). This approach allows for rapid collection of feedback directly from end users, which is crucial for validating the business idea and further directing development [2]. The MVP approach is well-documented in software engineering literature as an effective strategy for reducing time-to-market and validating product-market fit [15]. MVP applications mainly consist of one database that communicates with a monolithic backend application, and a single-page application (SPA) responsible for the final data presentation to the user.

1.3. Transition to microservices

When the MVP software implemented through a monolithic architecture proves to be successful, the next step is to ensure the continued application growth and to expand the technical team.

The use of microservices (Figure 4), transfers the app to a system that offers greater agility and reduced risk when developing features. Each microservice can be independently developed, tested, and deployed, which allows teams to respond more quickly to changes and market demands. [2]

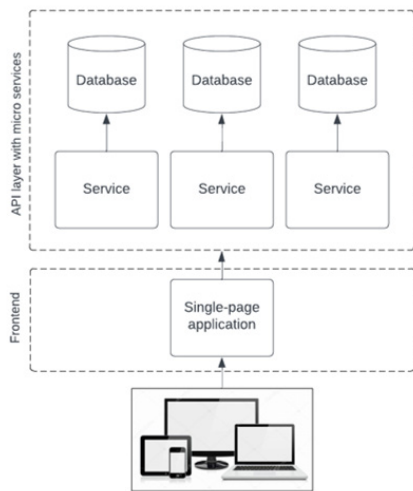


Figure 4: Microservice architecture with a single-page application [2]

With a microservice architecture, team members can independently make decisions about which database their service will use and according to which schema the data will be organized in the database. The benefits of microservice architecture have been extensively studied, with research highlighting improvements in scalability, fault isolation, and team autonomy [16].

The introduction of microservices has achieved significant improvements when it comes to scaling the API layer and the data layer, however, the frontend layer, which also requires scaling and development, still represents one large monolith [12]. This shortcoming opens the need for new approaches, such as micro frontend architecture, to enable full scaling and independent development of all parts of the application.

1.4. Micro frontend architecture

When it comes to the front-end part of the application, scaling options have been limited for a long time. Until a few years ago, the standard solution was creating

large backend applications where all the important logic was executed on the server, while the monolithic client application mainly served only to display data to the user.

In the last few years, users started expecting better user experience, more options and faster interactions. The need for responsive, feature-rich interfaces has driven the evolution of frontend architectures toward more modular approaches [17].

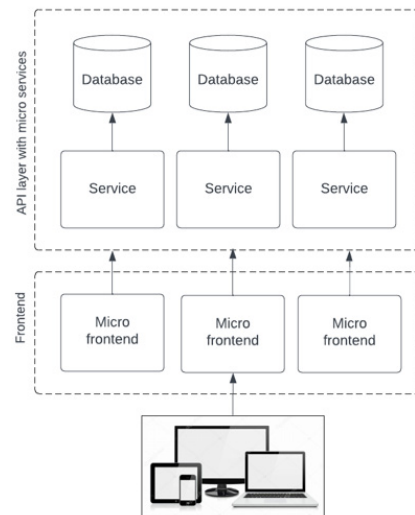


Figure 5: Micro services combined with micro frontends [2]

Micro frontend architecture (Figure 5) enables fast and efficient way of adding new functionalities to the frontend applications, while keeping pace with dynamic business development. This approach increases development speed and flexibility and allows teams to focus on specific parts of the application without affecting the overall system, therefore reducing risk and facilitating maintenance.

2. IMPLEMENTING ONLINE SHOP APPLICATION

Online Shop application represents a simple but functional system for selling products, which currently offers the following functionalities:

- Products overview - Users can see a list of all products available on the platform and filter products according to certain criteria (Figure 6).
- User registration and login - New users can create an account, while existing users can log into the system to access additional application features (Figure 7).
- Products management - After logging in, users are allowed to edit and delete their own products. They can also add new products. (Figure 8).

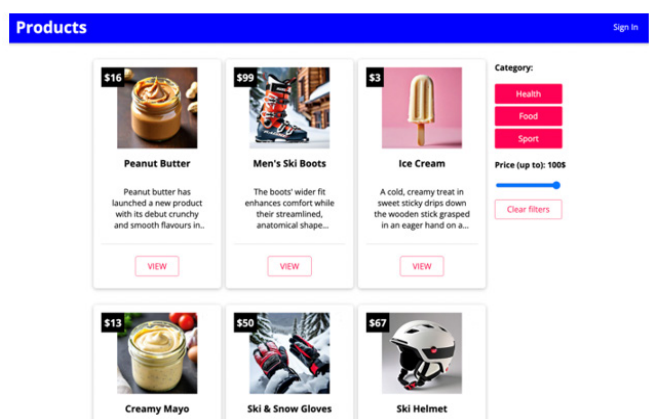


Figure 6: Homepage of the Online Shop application - products display and options for filtering results

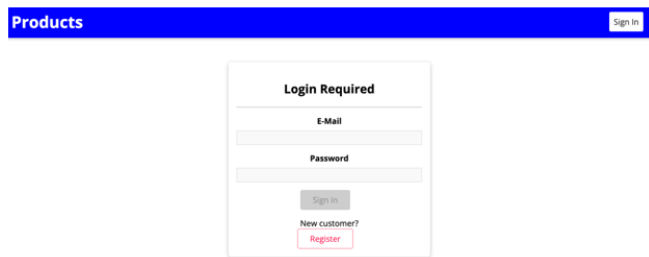


Figure 7: Page for creating a user account and logging into the application

In addition to these basic functionalities, the application can later be further expanded with advanced options such as integrating online payments, viewing sales statistics, rating and commenting on products, and sending notifications to users about the status of their products and purchases.

In order for these functionalities to be implemented quickly and efficiently in the future, it is necessary to start investing in project architecture improvements.

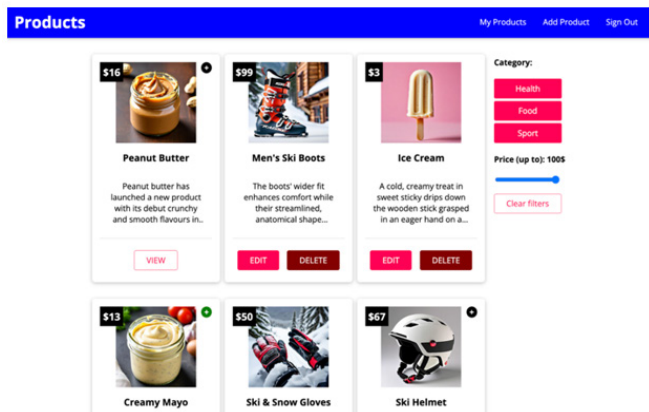


Figure 8: Homepage of the application after user has logged in

2.1. MVP version of Online Shop application

MVP of the Online Shop application was implemented throughout multi-tier architecture (one application for backend, and one for frontend).

The backend of the Online Shop application was developed using the Node.js programming language and the Express library, enabling efficient management of API requests, authentication and data processing through an asynchronous work model.

MongoDB database is used due to its flexibility, as it allows dynamic management of large amounts of data without the need for a fixed structure.

The frontend of the Online Shop application monolith is implemented as a Single Page Application (SPA) using the React.js framework, which enables fast and dynamic user interaction, reducing the need to load new pages. React.js brings modularity and allows component reuse, which makes application development and maintenance easier.

2.1.1. Technology stack selection and alternatives

The technology choices for the Online Shop application were made based on several key considerations: development speed, team expertise, ecosystem maturity, and alignment with micro frontend architectural principles.

Node.js and Express Framework

The backend was implemented using Node.js with the Express framework due to its non-blocking, event-driven architecture, which is well-suited for handling multiple concurrent requests typical in e-commerce applications. The JavaScript-based stack also enables code sharing between frontend and backend, reducing context switching for developers. Alternative backend technologies could include:

- Java with Spring Boot – which would provide stronger typing and enterprise-grade tooling, but it is more suitable for organizations with existing Java expertise
- .NET Core with ASP.NET - would be appropriate for organizations invested in the Microsoft ecosystem

MongoDB Database

MongoDB was selected as the database solution due to its schema flexibility, which is particularly valuable during the MVP phase when data models may evolve rapidly. The document-oriented structure aligns well with JSON-based API responses. Alternative database choices could include:

- PostgreSQL or MySQL - relational databases that would provide stronger data consistency guarantees and ACID compliance
- DynamoDB or other cloud-native databases - would offer managed scalability for cloud deployments

React.js Frontend Framework

React.js was chosen for the frontend implementation due to its component-based architecture, which naturally aligns with micro frontend principles. The virtual DOM

provides efficient rendering, and the extensive ecosystem offers numerous libraries for common functionality. Alternative frontend frameworks include:

- Angular - would provide a more opinionated, full-featured framework with built-in solutions for routing, forms, and HTTP communication
- Vue.js - would offer a gentler learning curve and flexible integration options

Impact of technology stack on architecture

It is important to note that the core micro frontend architectural principles demonstrated in this paper are technology-agnostic. Changing the technology stack would primarily affect implementation details of the composition layer (how micro frontends are loaded and integrated) as well as build and deployment pipelines.

However, the fundamental architectural decisions - vertical split by business domain, client-side composition and routing, and communication patterns - would remain valid regardless of the chosen technology stack. The key architectural benefits of team autonomy, independent deployment, and parallel development would be preserved across different technology choices.

2.2. Migrating Online Shop application to micro frontends architecture

The Online Shop system was initially developed as a monolithic frontend application. As the project expanded, requiring more features and greater team involvement, it became clear that the existing architecture needed improvement. To address these growing demands, a micro frontend architecture was introduced, enabling the user interface to be divided into smaller, independent modules.

This architectural shift allows multiple teams to work in parallel on different parts of the application, increasing overall efficiency, accelerating the development of new features, and reducing technical risks.

There are many ways to set up the architecture of a micro frontend application. To choose the best architecture for a project some architectural decisions must be made up front as they directly affect other decisions that need to be made, such as how to connect different pages, how to get the final page displayed to the user, and how micro frontends will communicate.

These decisions are covered by the micro frontend decisions framework, which has 4 key areas:

- Defining the micro frontends in the new architecture;
- Composing (connecting) the micro frontends;
- Routing micro frontends;
- Communication between micro frontends.

2.3. Establishing micro frontends

The process of identifying micro frontends begins with the decision on how micro frontends will exist on the page from a technical side (Figure 9) - as multiple micro frontends on one page (horizontal split) or as one micro frontend per page (vertical split). [2]

With a horizontal split, multiple micro frontends will be displayed to a user on the same page. This solution provides the flexibility to use the same micro frontends on several different pages, but it also requires more discipline and management to avoid creating too many micro frontends within one project. With vertical split, each team is responsible for their own business domain, such as authentication or catalogue, and each page represents one micro frontend.

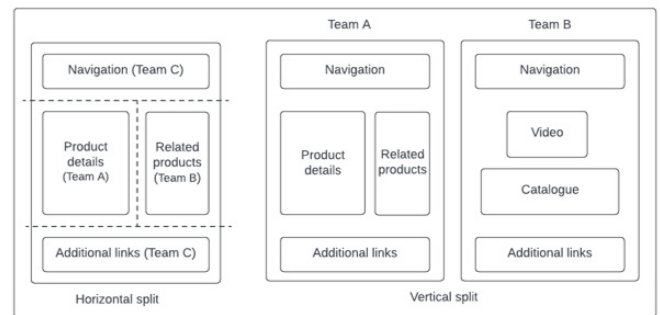


Figure 9: Overview of the horizontal and vertical split within the micro frontend architecture [2]

Since the initial version of the application was implemented as a Single Page Application (SPA) and its individual pages represent different areas of interest for a user, a vertical micro frontend architecture is a natural and efficient choice. In an SPA, each page or feature typically has unique requirements and serves a specific user category or interaction type, making vertical segmentation particularly suitable. With this approach, each page or functionality can be developed and maintained as an independent micro frontend with its own components, logic, and styling. Apart from navigation, the application has no elements that are reused across pages, so vertical split provides great flexibility for development and testing.

To analyse the individual aspects of this split through a decisions framework, the first decision that must be made is how to divide the application into micro frontends, that is, define business domain for each micro frontend.

As shown in Figure 10, the new application architecture consists of a main container application, responsible for loading navigation and integrating various micro frontends, along with three independent micro frontend applications:

- Auth (Authentication) - This micro frontend manages all user authentication processes, including account creation and login. It will also host any future functionality related to user account management.

- Marketing - The Marketing micro frontend is dedicated to displaying the product catalogue and enabling users to filter products by different criteria. In the future, it may include promotional content or special offers designed to engage users and encourage product exploration.
- Core - This micro frontend handles product management features available exclusively to logged-in users. It enables users to add, edit, or delete products within their personal account.

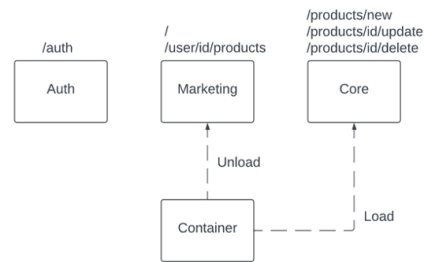


Figure 11: Client-side composition and routing

2.5. Communication between micro frontends

In theory, micro frontends should not communicate because each micro frontend should contain all the information needed for its operation. In practice, this is not always possible, especially when there are multiple micro frontends on a single page.

There are several options that can be used to notify one micro frontend that a certain event has occurred in another micro frontend within the vertical split:

- Query strings - In most cases, communication between micro frontends is performed using URL query string parameters. This method of communication between micro frontends is simple but not secure if sensitive data such as login details or other user information needs to be passed.
- Web storage - Using web storage is another efficient way for micro frontends to communicate and share data. This approach uses browser's built-in storage mechanisms.

Since the Online Shop application must store user login information, web storage is used for communication between micro frontends.

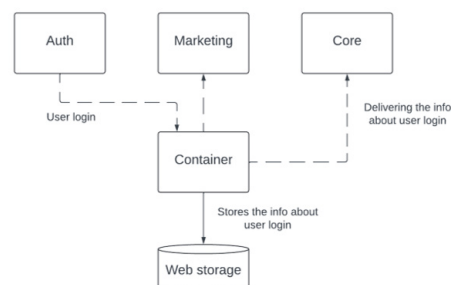


Figure 12: Communication between micro frontends using web storage and main/container application

2.4. Micro frontends composition and routing

There are different ways to compose and route micro frontend applications and they can take place on the server side, CDN-level and client side.

Although in theory any composition option can be used, all known techniques for vertical implementation of micro frontend architecture use client-side composition. This means that the main application will be responsible for loading and unloading the micro frontend and must ensure there is only one micro frontend loaded on the page, at the time. After client-side composition is chosen, the micro frontends decision framework helps to easily make the decision that routing should also be on the client side. Two types of routing will be used, global and local. Global will be responsible for loading different micro frontends (depending on the application URL), while local will be responsible for displaying different functionalities within one URL or one micro frontend.

When the user navigates from homepage to product editing page, he is directed to another URL, therefore the main application (Container) will remove Marketing and load Core MFE.

2.6. Implementing new feature within Online Shop application

After analysing users activity, it was decided to start developing the functionality for adding products to the cart. The main requirements include:

- displaying the add product button on each product card with visual confirmation of addition;
- an option to view the cart in the main menu;

- a cart page with an overview of added products and the ability to remove them, as well as options to continue shopping and proceed to payment.

The next step is to analyse which parts of this functionality should be implemented within which micro frontend.

Displaying the option to add products to the cart naturally fits into the responsibilities of the Marketing micro frontend, which is responsible for displaying products. In this case, part of the functionality is being added to an existing page, so it was easy to determine which micro frontend it belongs to.

Since the main application is responsible for displaying navigation, the team responsible for the main application will add a new link to the navigation that will serve to navigate the user to the page for displaying the cart's content.

The page for displaying the content of the cart is a completely new page, and it can be decided to assign it to either the Marketing or the Core micro frontend. This part of the functionality is seen as something that will be very important for further development of a business, therefore should be the responsibility of the Core micro frontend (and the team).

Since one micro frontend will be responsible for adding products to the cart and the other for displaying all products added to the cart, web storage will again be used for communication between micro frontends (to store the information about which products are in the cart).

After implementing the new requirement, the Online Shop application looks like in the Figure 13.

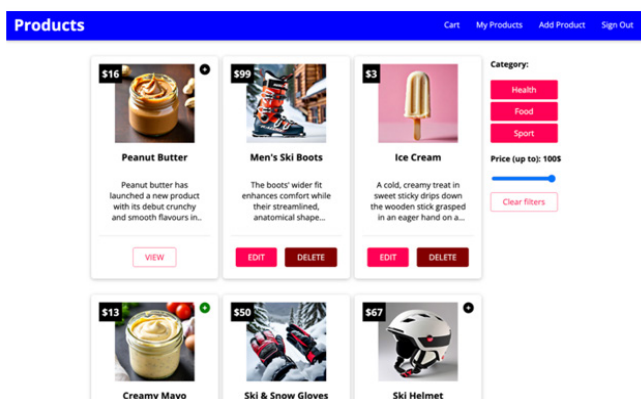


Figure 13: Online Shop - homepage after implementing the feature to add products to the cart



Figure 14: Online Shop - page for displaying the status of the cart

3. EFFICIENCY ANALYSIS OF MICRO FRONTEND ARCHITECTURE

This chapter will analyse and compare the time required to implement a new functionality for Online Shop application using two different architectures - monolithic and micro frontend. The focus of the analysis will be on implementing option to add products to the cart (described in Chapter 2), with an estimate required to develop, test, and deliver this functionality within both architectural approaches.

Every piece of software, regardless of the architecture type, must go through the same basic development phases to be delivered to the production environment, and the analysis will be done according to these phases:

- Planning and Analysis - research on the best practices and user needs, to assess the value of the new functionality.
- Requirements Definition - feature details are developed through sketches and technical consultations with developers.
- Design - a visual representation of the feature is created, which is then being used by developers.
- Development - the functionality is divided into smaller tasks that are being worked on and deployed to a test environment.
- Testing - new and existing functionalities are tested to ensure system stability.
- Software Delivery - after successful testing, the new software version is deployed to end users.
- Maintenance - functionality usage is monitored, and user support is provided as needed.

3.1. Estimation methodology

The time estimates presented in this analysis were derived using a combination of expert judgment and historical data from similar feature implementations within the development team. The estimation process followed these steps:

- Task decomposition - each development phase was broken down into specific activities (e.g., component creation, API integration, styling, unit testing).
- Historical benchmarking - previous implementations of comparable features were analysed to establish baseline effort estimates.
- Team consultation - estimates were reviewed and validated by experienced developers familiar with both architectural approaches.
- Adjustment for architecture - additional time was allocated for monolithic architecture to account for regression testing, coordination overhead, and deployment dependencies.

The smallest measurement unit used is 0.5 days (4 hours), representing a half-day of focused development work. This granularity was chosen to provide meaningful differentiation between tasks while acknowledging that some activities require less than half a day of effort.

- Assumptions:
- Team members have equivalent skill levels and familiarity with the respective architectures.
 - Development environment and tooling are properly configured and functioning.
 - There are no major technical blockers or external dependencies that could cause delays.
 - Code review and approval processes follow standard timelines.

The analysis provides insights into the relative efficiency gains achievable through micro frontend architecture, particularly for organizations managing multiple concurrent feature developments across distributed teams, while acknowledging that results may vary based on specific organizational context and application complexity.

3.2. Comparison of development through monolithic and MFE architecture

The implementation of functionality within a monolithic architecture happens sequentially through development phases and often requires greater involvement in all phases due to nature of the architecture itself. In monolithic systems, each change requires a review and testing of the entire code solution due to the tightly coupled components (called regression testing). This interdependence extends the time required for planning, development, testing, and delivery. A micro frontend architecture, on the other hand, provides greater flexibility. Teams work in parallel on different functionalities, which reduces dependencies and speeds up the development cycle. Planning and analysis are more efficient because all teams focus only on a specific module. The development phase is shortened because the code is developed and tested in isolation. Testing is also more efficient because errors can be identified and corrected at the module level without the need for extensive regression testing of the entire application.

Table I shows the implementation time of the add-to-cart functionality through both architectural approaches.

As mentioned in 3.1., for the ease of analysis, the smallest measurement unit used in estimating implementation is 0.5d, or half a day, although estimates for certain parts are even less than half a day, which is also highlighted in the table.

The analysis shows that the implementation time in a monolithic architecture is longer than in a micro frontend architecture. This difference becomes especially noticeable within more complex functionalities, in the planning, development and testing phases. While a monolithic architecture requires extensive planning and coordination due to the interdependence of all parts of the application, a micro frontend architecture allows for

Table I: Execution time for individual phases of software development within monolithic and micro frontend architecture

	Monolithic architecture	Micro frontend architecture		
		Marketing team	Core team	Auth team
Planning and Analysis	1d	<0.5d	<0.5d	0d
Reequirement Definition and Design	1d	<0.5d	<0.5d	0d
Development	2.5d	1d	1.5d	0d
Testing	1d	<0.5d	<0.5d	0d
Delivery	0.5d	<0.5d	<0.5d	0d
Maintanance	Starts on day 6	Starts on day 4		

the independent development of each component, which reduces implementation time and speeds up delivery. These findings align with research on modular software architectures, which has consistently demonstrated that reduced coupling between components leads to improved development velocity and team productivity [18].

In the case of implementing the functionality of adding products to the cart within a monolithic architecture, the process was as follows:

- Planning, analysis and requirements definition - 2 days
- Development phase – 2.5 days
- Testing of the entire application – 1 day
- Software delivery – 0.5 days

The total time required to deliver this functionality to users within a monolithic architecture is 6 working days. On the other hand, in micro frontend architecture, the time required for each individual phase is similar, but the key difference is the parallel work of autonomous teams. Each micro frontend team works on its own segment of the application independently of the others, which allows faster implementation and delivery.

Although a simple example is analysed here, and therefore the difference in duration between these two architectural approaches is small, analysis clearly shows that when using micro frontend architecture, users receive new functionality on the fourth day, while with monolithic architecture they wait until the sixth day, which is 33% faster. In more complex systems, the advantage of micro frontend architecture becomes even bigger.

Figure 15 gives an overview of the mentioned software development phases by day, for both monolithic and micro frontend architectures, which visually shows sequential or parallel development within different architectures.

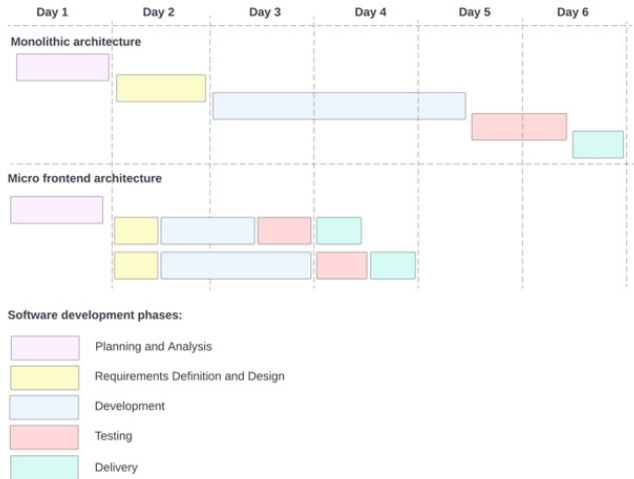


Figure 15: Execution time by phases - monolithic vs micro frontend architecture

In addition to time savings, micro frontend architecture brings several strategic advantages, especially in large organizations that manage complex software solutions. Greater flexibility and faster reaction to changing user requirements enable more efficient development and system scalability.

Unlike monolithic systems, which require extensive coordination and slow down delivery, micro frontends allow independent teams to work in parallel, reducing dependencies and accelerating implementation, testing, and upgrades. This modular approach also simplifies maintenance, as updates and fixes can be applied to individual modules without affecting the entire system. Teams can adopt different technologies tailored to their needs, supporting innovation and easier resource scaling.

4. CHALLENGES WHILE USING MFE ARCHITECTURE

This chapter examines how micro frontends introduce complexity into certain concepts due to the nature of this type of architecture. While micro frontend architecture offers significant benefits, it also introduces specific challenges that must be carefully managed. These challenges have been documented in both academic literature and industry practice reports [19][20].

4.1. Redundancy

When using MFE architecture, each team develops its application within a selected set of technologies, which introduces redundancy because everyone is working on the same or similar things, such as maintaining the application server, implementing a solution for continuous software delivery, and adding/duplicating the same functionalities on the client side (JavaScript and CSS).

4.2. Application performance

Performance optimization in micro frontend architectures has been the subject of recent research, with studies proposing various strategies for code sharing and lazy loading to mitigate bundle size issues [21]. Micro frontends require more JavaScript and CSS code because each of them implements the same or similar functionalities separately, so that they can work independently of other micro frontends, which ultimately means more frontend code to load on the client side, directly affecting the performance of the application.

4.3. Consistency of user experience

Maintaining a consistent user experience of an application across different micro frontends can be difficult, especially if different teams are responsible for each micro frontend. It is necessary to agree in advance on the rules regarding the use of certain libraries and how to apply styles.

4.4. Coordination between teams

In large teams and organizations, clearly defined conventions for exchanging data between micro frontends are essential. Without them, compatibility issues can arise, which can slow down the development process and increase maintenance costs.

4.5. Heterogeneity

One of the biggest advantages of the micro frontend architecture is that it allows teams to choose the technologies and tools they will use within their project. However, if the same or similar technologies are used across teams, it is easier to exchange best practices, get help, or move developers from one team to another if necessary.

4.6. Data integrity

If each micro frontend has its own separate backend system, it is necessary to use data replication to ensure data consistency. Also, different caching strategies or caching times can affect data integrity, so additional care must be taken.

5. IMPACT ON TEAMS STRUCTURE

In addition to the technical advantages, this architecture also has a significant impact on the team structure. For this reason, it is important to consider the technical aspects, but also to understand how this architecture contributes to the organization of work within a company. The relationship between software architecture and organizational structure has been extensively studied, most notably through Conway's

Law, which states that organizations design systems that mirror their communication structures [9][10]. The inverse Conway's law suggests that organizations can deliberately design their architecture to promote desired team structures and communication patterns [11]. Research has shown that aligning team structure with architectural boundaries can significantly improve development efficiency [22].

In the case of the Online Shop application, the application is divided into three micro frontends, and thus three development teams:

- Auth - a team responsible for user login and registration,
- Marketing - a team focused on product display and options for finding the desired product,
- Core – a team responsible for product management and the main application (Container).

Each of these teams work on its respective segments of the application according to the software development process described earlier.

Ideally, an optimal architecture adapted to the project context would be designed and teams would be organized around it. However, in practice, this is often not possible. If an organization cannot change its structure, this fact must be a key factor when choosing the appropriate architecture.

In the OnlineShop project example, the starting assumption was that people could be assigned to teams after defining the project architecture. The chosen structure was not the only possible option - for example, there had been a limit of having only two teams, the project would most likely have been split into two micro frontends to simplify implementation and communication flows.

6. CONCLUSION

The concept of micro frontend architecture marks a significant advancement in the design and development of modern web applications. By extending microservice principles to the frontend, it enables teams to develop, deploy, and scale individual parts of an application independently. This modular approach enhances scalability, shortens development cycles, and improves maintainability. Dividing the frontend into smaller, self-contained units allows teams to choose the technologies and libraries that best fit their specific needs, fostering innovation and flexibility.

As demonstrated in this paper, micro frontends influence not only technical implementation but also organizational structure and development practices. Their adoption aligns naturally with decentralized, cross-functional teams empowered to own specific parts of the application from

concept to delivery. This shift increases productivity, strengthens team ownership, and reduces dependencies that often slow down development.

For large organizations, micro frontends enable parallel workflows, allowing teams to develop, test, and deploy features independently. This autonomy accelerates delivery and improves reliability, as errors can be isolated to specific modules without affecting the entire system.

However, this architecture also introduces challenges. Ensuring a consistent user experience, managing communication between micro frontends, and avoiding performance issues require careful planning. Without strong governance, there is a risk of fragmentation in code quality, design standards, and best practices. Effective coordination and shared guidelines are essential to maintain coherence across teams.

Micro frontends are particularly valuable for large, distributed organizations, offering a structured way to manage cross-functional teams while minimizing inter-team dependencies. Still, their adoption must be approached thoughtfully, considering team expertise, project complexity, and long-term maintenance.

When implemented well, MFE architecture can improve agility, adaptability, and development efficiency, which highlights its importance in the evolution of modern software systems.

REFERENCES

- [1] Micro Frontends - extending the microservice idea to frontend development, Available: <https://micro-frontends.org/> [22.10.2024.]
- [2] Building Micro-Frontends, by Luca Mezzalana, O'Reilly Media Inc., 2021.
- [3] The Strengths and Benefits of Micro Frontends, Available: <https://www.toptal.com/front-end/micro-frontends-strengths-benefits> [04.11.2024.]
- [4] The Future of Micro Frontends, Available: <https://betterprogramming.pub/the-future-of-micro-frontends-2f527f97d506> [23.10.2024.]
- [5] Micro Frontends Architecture and Best Practices, Available: <https://www.xenonstack.com/insights/micro-frontend-architecture> [23.10.2024.]
- [6] Micro Frontends in Action, by Michael Geers, Manning Publications Co., 2020.
- [7] What is the Software Development Life Cycle (SDLC), Available: <https://theproductmanager.com/topics/software-development-life-cycle/> [23.09.2024.]
- [8] <https://www.udemy.com/> - Microfrontends with React: A Complete Developer's Guide
- [9] Conway's law, Available: https://en.wikipedia.org/wiki/Conway%27s_law [24.06.2025.]

- [10] How do committees invent?, by Melvin E. Conway, Available: <https://www.melconway.com/Home/pdf/committees.pdf> [15.04.2025.]
- [11] Inverse-Conway-Maneuver: How to speed up product development teams successfully, Available: <https://www.thoughtworks.com/insights/blog/customer-experience/inverse-conway-maneuver-product-development-teams> [16.05.2025.]
- [12] Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review, by S. Peltonen, L. Mezzalana, and D. Taibi, Information and Software Technology, 2021.
- [13] Micro-frontends: application of microservices to web front-ends, by A. Pavlenko, N. Askarbekuly, S. Megha, and M. Mazzara, Journal of Internet Services and Information Security, 2020.
- [14] On the criteria to be used in decomposing systems into modules, by D. L. Parnas, Communications of the ACM, 1972.
- [15] The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, by E. Ries, Crown Business, 2011. Available: <https://www.amazon.com/Lean-Startup-Entrepreneurs-Continuous-Innovation/dp/0307887898> [13.03.2026.]
- [16] Building Microservices: Designing Fine-Grained Systems, by S. Newman, O'Reilly Media, 2015.
- [17] Learning JavaScript Design Patterns: A JavaScript and jQuery Developer's Guide, by A. Osmani, O'Reilly Media, 2017.
- [18] Exploring the structure of complex software designs: An empirical study of open source and proprietary code, by A. MacCormack, J. Rusnak, and C. Y. Baldwin, Management Science, Available: <https://pubsonline.informs.org/doi/10.1287/mnsc.1060.0552> [16.03.2026.]
- [19] The pains and gains of microservices: A systematic grey literature review, by J. Soldani, D. A. Tamburri, and W. J. Van Den Heuvel, Journal of Systems and Software, 2018.
- [20] Design, monitoring, and testing of microservices systems: The practitioners' perspective, by M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, Journal of Systems and Software, 2021.
- [21] Performance analysis of microservice design patterns, by A. Akbulut and H. G. Perros, I Computing, Available: <https://ieeexplore.ieee.org/document/8888945> [19.03.2026.]
- [22] Identification of coordination requirements: implications for the Design of collaboration and awareness tools, by M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work (CSCW), 2006.

BIOGRAPHY

Dženita Vejsilović is an experienced software engineer with 10 years in the IT industry. She holds a degree in Electrical Engineering (graduated in 2016.) and she received her M.Sc degree in Computer Science in 2025., focusing on Micro-Frontend Architecture. At Glide Publishing Platform, she has progressed from Software Developer to Head of Content Engineering. Her expertise is in customer success area which includes headless CMS, API design, SEO, and client-oriented technical consulting.

Nermin Sarajlić graduated in 1987 and received his M.Sc. degree in 1997 from the Faculty of Electrical Engineering and Mechanical Engineering, respectively, and his Ph.D. degree in 2002 from the Faculty of Electrical Engineering of the University of Tuzla, Bosnia and Herzegovina. His field of interest is the calculation of the coupled electromagnetic-temperature fields, cryptography, and crypto analysis.