

Micro-Frontend Architecture in Software Development: A Systematic Mapping Study

Giovanni Cunha de Amorim^a and Edna Dias Canedo^b
University of *Ilia-DF, Brazil*
giovanni.amorim.ti@gmail.com, ednacanedo@unb.br

Keywords: Micro-Frontend Architecture, Software Architecture, Web Development, Systematic Mapping Study.

Abstract: **Context:** The integration of new technologies into monolithic frontend projects poses significant challenges, including code redundancy, inconsistency, and limited scalability. The micro-frontend architecture has emerged as a promising solution, offering a more modular and independent approach to frontend development. **Goal:** This study aims to explore the impacts and challenges of adopting micro-frontend architecture in software projects. **Method:** We conducted a systematic mapping study to identify common architectural patterns and strategies used in micro-frontends. **Results:** Our findings underscore the potential of micro-frontend architecture in modernizing frontend development, particularly for large, complex projects. However, successful implementation depends on the project's scale and requires careful methodological and technological planning.

1 INTRODUCTION

The growing interest of large organizations in adopting micro-frontend architectures highlights a significant shift in software development practices (SPA, 2022), driven by the pursuit of more flexible and effective approaches to building frontend applications (Simões et al., 2023). However, this strategy can lead to a decrease in operational efficiency. To mitigate this issue, micro-frontends enable the integration of the entire application portfolio through a single gateway, addressing the need for rapid application evolution (Moraes et al., 2024; Rappl and Schöttner, 2021).


By enabling independent development and deployment of application modules, this approach enhances scalability and flexibility in software projects (Gashi et al., 2024; Tilak et al., 2020). However, its adoption also brings challenges, such as duplicated functionalities leading to code redundancy, difficulties in ensuring a consistent user experience, increased complexity due to the use of diverse technologies, and a growing volume of frontend code resulting from multiple dependencies (Geers, 2020).


Taibi and Mezzalira (Taibi and Mezzalira, 2022) identified four crucial decisions to consider before embarking on a micro-frontends project. The first is

the choice between a “Horizontal Split”, and a “Vertical Split”, which determines whether teams will collaborate on different frontends within the same view (horizontal) or each team will manage a distinct view (vertical). The second decision, “Composition Side”, involves choosing between client-side composition, where micro-frontends are dynamically loaded in the browser, and server-side composition. The third decision focuses on “Routing”, which concerns directing user requests to the correct micro-frontend. Lastly, the fourth decision, “Communication between Micro-frontends”, addresses how to enable communication between different applications, either through custom events or shared services.

Although the concept has gained popularity in recent years, more research is needed to explore its practical applications in software development environments. This study aims to fill this gap by systematically identifying and analyzing the key factors, practices, and challenges reported in the literature regarding the implementation of micro-frontends.

Given the challenges associated with large monoliths and their drawbacks, such as code redundancy, consistency issues, heterogeneity, and excessive code (Geers, 2020; Lewis and Fowler, 2014), as well as architectural decisions related to composition, routing, and communication (Taibi and Mezzalira, 2022), this study aims to explore the impacts and challenges of adopting micro-frontends in the development of com-

^a  <https://orcid.org/0009-0001-7933-1552>

^b  <https://orcid.org/0000-0002-2159-339X>

plex web applications. To achieve this, we conducted a Systematic Mapping Study (SMS) to identify the impacts and challenges of implementing the micro-frontends architecture.

Thus, we highlight the following contribution directions: the identification of design and architecture patterns specific to the implementation of micro-frontends, aiming to promote consistency, modularity, and reuse of front-end components in a distributed ecosystem; the identification of practices and tools that facilitate continuous integration and continuous delivery (CI/CD) of micro-frontends, enabling teams to implement updates quickly and safely. The results of this work highlight the positive impact of micro-frontends on software projects, the architectural patterns used, and the challenges faced during adoption, contributing to the validation of the SMS.

2 RELATED WORK

Experiments comparing frontend architectures connected to microservices-based backends were conducted by (Harms et al., 2017). Their findings indicate that Micro-Frontends offer good testability and modifiability, though with variations in performance and interface consistency. This study emphasized the practical application of Micro-Frontends in hybrid architectures but did not explore specific composition or splitting strategies, which are gaps our research seeks to fill.

A multi-platform web application using Micro-Frontends and microservices to ensure a consistent user experience across different devices was developed by Mena et al. (Mena et al., 2019). Their primary focus was on fulfilling the application's functional requirements, highlighting that Micro-Frontends helped isolate the development of distinct components and minimize codebase conflicts. Our research complements this approach by investigating architectural patterns that support component isolation in a more structured and comparative manner.

A Multivocal Literature Review to investigate the motivations behind companies adopting Micro-Frontends was conducted by Peltonen et al. (Peltonen et al., 2021). The study analyzed 173 publications, 43 of which addressed motivations, benefits, and challenges. The results showed that companies adopted Micro-Frontends to increase team autonomy and reduce frontend complexity, benefits similar to those promised by microservices in the backend. However, the authors identified drawbacks such as increased application payload size and code duplication. While the study highlights general motivations

and impacts, it does not address specific architectural patterns, which is the main focus of our research.

Pavlenko et al. (Pavlenko et al., 2020) reported the creation of a React-based single-page application using Micro-Frontends. While the architecture introduced greater complexity, the authors found it suitable for applications with extensive frontend logic and larger development teams. Although the study detailed operational challenges, it did not address the systematization or comparison of different architectural patterns, which is a key aspect of our analysis.

Männistö et al. (Männistö et al., 2023) described the migration of a monolithic application to a Micro-Frontends architecture using a framework-less approach based on Web Components. The study highlighted that even small teams can benefit from Micro-Frontends, particularly in terms of configurability, and that web standards-based APIs offer viable alternatives to JavaScript frameworks. While the authors focused on cost benefits and client-specific configurability, our research delves into the analysis of specific architectural strategies and their applicability to diverse development scenarios.

3 RESEARCH METHOD

In this work, we conducted a *Systematic Mapping Study (SMS)* to identify the impacts and challenges associated with the implementation of the micro-frontends architecture. This approach enabled us to systematically review existing literature, categorize key findings, and synthesize insights relevant to both academic and industry perspectives. The goal of this study is to provide insights into the architectural views and patterns associated with micro-frontends and understand how these elements resonate in the development environment. It is important to emphasize that our scope is strictly limited to aspects related to the implementation of the micro-frontends architecture, without delving into specific technical development issues such as programming languages and frameworks. To achieve the objectives of the SMS, we formulated three research questions to guide our investigation, described as follows:

- RQ.1: What architectural patterns are used in the implementation of micro-frontends?
- RQ.2: What are the impacts of adopting a micro-frontends architecture?
- RQ.3: What challenges do software development teams encounter when adopting micro-frontends?

Search String. The PICOC terms (Wohlin et al., 2012) were used as a guide to define the inclu-

sion and exclusion criteria, although they were not directly applied in the construction of the search string. The Population was defined by terms related to micro-frontends, software industry, development teams, and micro-frontend architecture adoption. The Intervention focused on terms like implementation, patterns, and adoption. The Outcome was shaped by terms such as impacts, challenges, team coordination, continuous integration, continuous delivery, and dependencies. Finally, the Context involved software development, implementation, and frontend paradigms. The search string was iteratively adjusted based on preliminary results, ensuring greater precision in identifying relevant articles for the study.

We selected five digital databases (ACM Digital Library, IEEE Xplore, ScienceDirect, Scopus e Springer Link) to apply the search string. The choice of these platforms was based on their relevance in Software Engineering research (Brereton et al., 2007). The search string used in the systematic mapping study was: (“micro-frontend” OR “micro frontend” OR “microfrontend”). Studies published before 2016, as the micro-frontend technology became widely adopted primarily from that year onward.

Selection Criteria. To ensure a rigorous and well-defined selection process, we adopted specific inclusion and exclusion criteria, as summarized in Table 1. The inclusion criteria focus on identifying studies that provide comprehensive insights into the technologies, challenges, and organizational impacts associated with micro-frontends. Conversely, the exclusion criteria were designed to filter out studies that lack relevance or clarity regarding the adoption and implementation of micro-frontends.

Table 1: Selection Criteria.

Type	Criteria
Inclusion	<p><i>IC1.</i> Technologies or methodologies used in the implementation of micro-frontends.</p> <p><i>IC2.</i> Challenges related to the implementation of micro-frontends.</p> <p><i>IC3.</i> Impacts on team organization when implementing technologies and methods related to micro-frontends.</p>
Exclusion	<p><i>EC1.</i> Studies that do not explicitly address the adoption of micro-frontend architecture, or that lack a clear focus.</p> <p><i>EC2.</i> Studies published in languages not understood by the authors (Portuguese and English).</p>

Execution. To conduct the review, we used the Parsifal tool. Figure 1 presents the number of studies selected at each stage of the review. A total of 279 studies were initially identified, distributed across the following databases: 14 from the ACM Digital Li-

brary, 150 from IEEE Xplore, 27 from ScienceDirect, 48 from Scopus, and 40 from Springer Link. During the Duplicate Removal Phase, 110 duplicate studies were identified and removed. This left a final set of 169 studies for analysis. Then, inclusion, exclusion, and quality assessment criteria were applied to the remaining studies through a review of titles and abstracts.

Although the initially selected studies addressed topics related to the search string, 119 studies were excluded for not meeting the established selection criteria or not being relevant to the context of this paper, resulting in a total of 50 studies for analysis in the next phase.

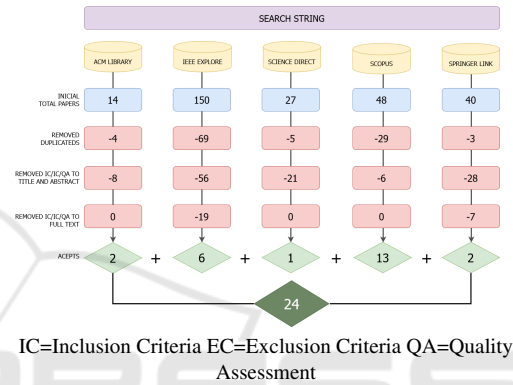


Figure 1: Selected Studies for Data Extraction.

Finally, we conducted a full reading of the selected studies. During this stage, we applied all the established selection criteria to a total of 24 studies, including 2 from the ACM Digital Library, 6 from IEEE Xplore, 1 from ScienceDirect, 13 from Scopus, and 2 from Springer Link.

3.1 Data Extraction

In the data extraction phase, our goal was to ensure the completeness of the SMS. To achieve this, we systematically identified, evaluated, and interpreted all the selected studies (Table 2). The data extracted from these studies served as the foundation for both quantitative and qualitative analyses. This process went beyond merely classifying the studies; it involved a detailed evaluation to support the subsequent phases of the investigation. We have made the Supplementary Material available on Zenodo¹.

¹<https://zenodo.org/records/14834868>

Table 2: Data Extraction.

Study	Year	#Cite	Type	Conference/Journal	Score	Ref.
PS1	2021	21	J	Information and Software Technology	2.0	(Peltonen et al., 2021)
PS2	2020	0	J	Journal of Internet Services and Information Security	1.5	(Pavlenko et al., 2020)
PS3	2022	1	C	AICMHI 2022	2.0	(Mohammed et al., 2022)
PS4	2023	0	C	CLOSER 2023	1.5	(Abdelfattah and Cerný, 2023)
PS5	2019	11	J	IOP Publishing Ltd	1.5	(Yang et al., 2019)
PS6	2023	0	C	Politechnica University of Bucharest	2.0	(Petcu et al., 2023)
PS7	2020	5	C	ETFA 2020	1.0	(Shakil and Zoitl, 2020)
PS8	2021	0	C	ETFA 2021	2.0	(Lorenz et al., 2021)
PS9	2023	0	J	International Journal of Web Engineering and Technology	1.5	(Wanjala, 2022)
PS10	2022	0	J	Journal of Information Processing	2.0	(Nishizu and Kamina, 2022)
PS11	2021	0	C	Budapest Tech Polytechnical Institution	2.0	(Pölöskei and Bub, 2021)
PS12	2022	1	C	ICTI 2022	1.5	(Stefanovska and Trajkovic, 2022)
PS13	2021	1	C	ICITCS 2021	1.5	(Noppadol and Limpiyakorn, 2021)
PS14	2022	0	C	SummerSOC 2022	2.0	(Bühler et al., 2022)
PS15	2021	11	J	SN Computer Science	2.0	(Sorgalla et al., 2021)
PS16	2020	4	C	IEEE-HYDCON 2020	1.5	(Tilak et al., 2020)
PS17	2023	1	C	ICSA 2023	1.0	(Männistö et al., 2023)
PS18	2023	1	C	AICT 2023	2.0	(Perlin et al., 2023)
PS19	2019	17	J	IEEE Access Journal	2.0	(Mena et al., 2019)
PS20	2022	1	C	ACM SIGSOFT Softw. Eng. Notes	2.0	(Taibi and Mezzalana, 2022)
PS21	2023	1	C	Web3D 2023	1.0	(Simões et al., 2023)
PS22	2020	9	J	KES-2020	2.0	(Wang et al., 2020)
PS23	2023	0	C	ISCSIC-2023	2.0	(Zhang et al., 2023)
PS24	2024	0	C	MECO	2.0	(Gashi et al., 2024)

4 SMS RESULTS

The results of our Systematic Mapping Study (SMS), addressing the three research questions (RQs) that guided our investigation. The analysis provides insights into the architectural patterns employed in micro-frontend implementations (RQ.1), the impacts of adopting this approach in software development projects (RQ.2), and the challenges faced by development teams during its adoption (RQ.3). The findings are synthesized from relevant studies and categorized to highlight key trends, benefits, and obstacles associated with micro-frontends.

4.1 RQ.1. What Architectural Patterns Are Used in the Implementation of Micro-Frontends?

To address RQ.1, we used the data from the selected studies presented in Table 2. In these studies, architectural views and patterns used in the implementation of micro-frontends were identified, including technologies such as frameworks, programming languages, and tools for development and deployment. Due to the breadth of RQ.1, we divided it into two topics: architectural views and architectural patterns.

Architectural Views. The architectural views (Figure 2) identified in the studies illustrate the strategies adopted by companies to implement micro-frontend architecture, as follows:

- **API View:** Addressed in 20 studies (PS1, PS2, PS3, PS4, PS5, PS8, PS9, PS10, PS11, PS12, PS13, PS14, PS15, PS16, PS17, PS19, PS20, PS21, PS23, and PS24), this view demonstrates the role of APIs as a single entry point for backend requests, allowing micro-frontends to remain independent and loosely coupled, facilitating communication between components, and optimizing development and maintenance.
- **Single-SPA View:** Mentioned in 18 studies (PS1, PS2, PS3, PS4, PS5, PS6, PS7, PS9, PS10, PS11, PS12, PS13, PS17, PS18, PS20, PS21, PS22, and PS24), this view supports the scalability and modularity of the application, allowing for the addition of new micro-frontends without impacting existing parts of the application.
- **Web Components:** Discussed in 13 studies (PS2, PS5, PS6, PS8, PS10, PS11, PS12, PS14, PS17, PS19, PS20, PS22, and PS24), the implementation in the studies used technologies such as Custom Elements, HTML Templates, Shadow DOM, HTML, and JavaScript without the need for a specific library or framework.

- **App Container and Module Federation Views:** The App Container view (containerization) was referenced in 7 studies (PS1, PS2, PS4, PS7, PS11, PS12, PS22). Similarly, Module Federation, addressed in seven studies (PS3, PS6, PS12, PS18, PS20, PS21, and PS24), these studies explored how different parts of an application can be loaded on demand without requiring a full system recompilation.
- **DevOps Views:** DevOps was discussed in 4 studies (PS1, PS2, PS11, and PS15), noted for its efficiency in continuous integration and delivery (CI/CD).
- **BFF Views:** The BFF (Backend for Frontend) view appeared in 3 studies (PS3, PS4, and PS11), highlighting that different parts of the user interface can communicate with specific backends, ensuring targeted and efficient communication.

These findings highlight the various architectural views employed in micro-frontend implementation, emphasizing their significance in promoting modularity, scalability, and effective communication.

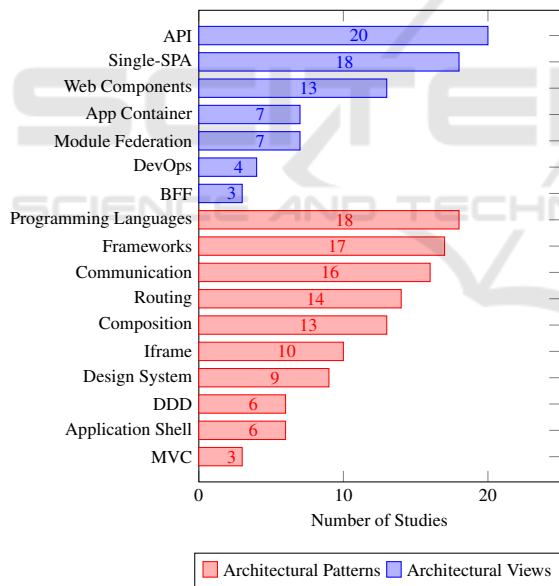


Figure 2: Architectural Views and Architectural Patterns.

Architectural Patterns. During the analysis of the selected studies, we identified the most employed architectural patterns in the implementations of micro-frontends architecture (Figure 2). These patterns represent the strategies adopted by development teams to address routine challenges, such as choosing programming languages, frameworks, libraries, and specific decisions regarding micro-application architecture, such as composition and routing.

- **Programming Languages:** Mentioned in 18 studies, JavaScript, TypeScript, HTML, and CSS are widely used in the frontend layer. The studies addressing these languages include PS1, PS2, PS5, PS6, PS7, PS8, PS9, PS10, PS11, PS12, PS13, PS14, PS17, PS18, PS20, PS21, PS22, and PS24. These languages are often used in conjunction with frameworks such as Angular, React, and Vue.
- **Frameworks:** The choice of frameworks Angular, React and Vue was the most discussed in studies PS1, PS2, PS3, PS5, PS7, PS8, PS9, PS10, PS11, PS12, PS13, PS14, PS18, PS19, PS20, PS21, and PS22. These 17 studies highlighted the flexibility provided by micro-frontends, allowing teams to adopt their preferred technologies for developing components independently.
- **Communication Pattern:** Referenced in studies PS1, PS4, PS5, PS6, PS7, PS8, PS12, PS13, PS14, PS15, PS17, PS18, PS19, PS20, PS22 and PS24, this pattern promotes more efficient integration between micro-applications. It improves communication between components, particularly through dynamic routing, offering more effective control and management of component state.
- **Routing Pattern:** Mentioned in studies PS1, PS2, PS5, PS6, PS9, PS10, PS11, PS12, PS19, PS20, PS21, PS22, PS23, and PS24, this pattern improves the efficiency of the application's initial load by dynamically loading micro-frontends. It can be managed on the server side, reducing load time and enhancing the user experience (Mezzalana, 2021). The pattern is associated with deployment independence, team autonomy, and ease of code maintenance, although study PS9 noted increased operational complexity when used with SPAs and the *Static Assets* pattern.
- **Composition Pattern:** Analyzed in studies PS1, PS2, PS3, PS6, PS11, PS12, PS13, PS14, PS15, PS18, PS19, PS20, and PS24, this pattern includes Client-Side Composition, Server-Side Composition, and Edge-Side Composition. Composition is associated with increased application scalability and operational complexity.
- **Iframe Pattern:** Highlighted in studies PS1, PS2, PS5, PS6, PS11, PS12, PS19, PS20, PS22, and PS24, this pattern simplifies the loading of multiple applications within the browser, promoting independence and agility in team deployments. However, it may lead to potential CSS style conflicts and code redundancy.
- **Design System:** Mentioned in studies PS5, PS6, PS8, PS9, PS15, PS17, PS18, PS20, and PS24,

this pattern addresses code reuse, interface standardization, and reduction of complexity in front-end application development. Study PS24, although not explicitly using the term Design System, refers to it as Component Library and User Experience (UX), contextualizing it with design patterns.

- **DDD Pattern:** Mentioned in studies PS1, PS12, PS15, PS21, PS22, and PS24, this pattern aligns business strategies with technology to improve operational efficiency. The studies discuss how DDD, in the context of Micro-Frontends, helps unify product and development teams by associating each micro-frontend with a specific business domain, ensuring better alignment between technology and business needs while supporting scalable and adaptable architectures.
- **Application Shell:** Referenced in studies PS1, PS11, PS16, PS17, PS20, and PS24, this pattern mounts and unmounts a micro-frontend, improving performance by quickly loading the basic structure of the application for a faster initial response.
- **Model-View-Controller (MVC):** Addressed in studies PS2, PS9, and PS22, this pattern improves source code organization and maintenance efficiency.

The identified architectural patterns reflect diverse strategies and tools employed by development teams to tackle common challenges in micro-frontend architectures.

4.1.1 Comparative Analysis of Micro-Frontend Architectural Patterns

In this section, we present a comparative analysis of the main architectural patterns for micro-frontends: *Split Horizontal*, *Split Vertical*, *Client-Side Composition*, *Edge-Side Composition*, and *Server-Side Composition*. Although other patterns can be applied in different software architecture contexts, we focus on those most relevant to this approach. We will then explore the differences between the splitting approaches and composition methods, which have a significant impact on the implementation of micro-frontend-based solutions.

Horizontal Split. In the Horizontal Split pattern, different teams work on distinct functionalities within a single page or view, sharing the same rendering area on the front end. Each team is responsible for a portion of the application's layout, facilitating collaboration across teams working on different functionalities (Figure 3).

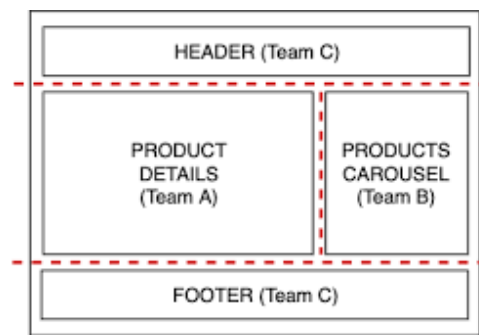


Figure 3: Horizontal Split (Mezzalira, 2021).

This model is more suitable for small teams or applications with simple requirements, where communication among members is more fluid. In Study PS12, the authors present use cases that apply this approach and also conclude that Split Horizontal can introduce communication overhead between teams, which may hinder coordination and impact the user experience if not managed efficiently.

Vertical Split. In the Vertical Split pattern, teams are responsible for entire sections of the system, such as a login screen or a user dashboard, having full control over the behavior and appearance of that section. This approach aligns with the DDD principle, as discussed for example in Study PS24, where each team is responsible for a specific business domain, such as authentication. By structuring teams around business capabilities, this pattern enhances domain expertise and improves development efficiency. The main advantage of this pattern is the greater autonomy and control teams have over their work areas, facilitating responsibility isolation and system maintenance, as shown in Figure 4.

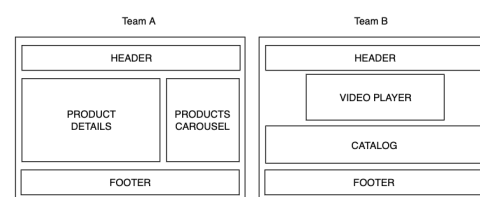


Figure 4: Vertical Split (Mezzalira, 2021).

However, coordinating functionalities that involve data or actions across different sections can be more challenging, increasing system complexity. Additionally, there is a higher likelihood of functionality duplication between teams. Split Vertical is better suited for larger systems with multiple independent functionalities that require greater team autonomy.

Client-Side Composition. This approach involves assembling components on the client side, meaning in the user's browser. The micro-frontend is dynamically loaded when the page is accessed as show in Figure 5, reducing server costs as composition is handled by the client and improving user experience with asynchronous and responsive loading.

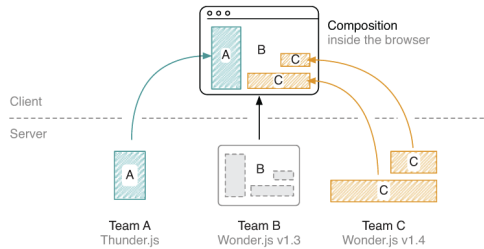


Figure 5: Client-Side Composition (Geers, 2020).

In Study PS11, it is added that lazy-loading is necessary for satisfactory client-side performance, ensuring that only the required modules are loaded when needed. This approach involves assembling components on the client side, meaning in the user's browser. The micro-frontend is dynamically loaded when the page is accessed, reducing server costs as composition is handled by the client and improves user experience with asynchronous and responsive loading. However, this approach also increases client-side complexity, resulting in higher maintenance costs within the browser. If too many micro-frontends are loaded simultaneously, performance degradation may occur. Reusability, another fundamental aspect, enables some frontend functionalities to be served directly across various platforms.

Server-Side Composition. This pattern are composed on the server, which generates a fully assembled page, including micro-frontends ready to be rendered in the browser. This model reduces client-side complexity since composition happens on the server, providing better performance control by distributing the workload between the client and the server.

Figure 6 shows the server-side approach, where page composition occurs on the server before being sent to the client (browser). The server assembles the micro-frontends to build the complete page and sends a fully integrated version to the client. Study PS19 highlights that this approach is a proven method for implementing microservices mechanisms in the frontend, such as API gateway patterns, circuit breakers, and service discovery (client-side and server-side). These mechanisms enhance scalability, resilience, and flexibility in distributed systems with micro-frontends. However, this approach can increase

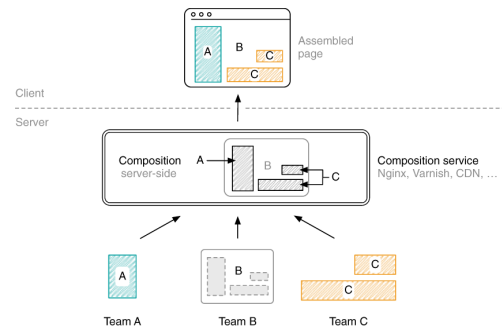


Figure 6: Server-Side Composition (Geers, 2020).

server load, potentially affecting performance in high-demand systems, and provides less flexibility for dynamic content changes.

Edge-Side Composition. Edge-Side composition, as described by Geers (Geers, 2020), involves combining Micro-Frontends on the server or an intermediary point like a CDN, which is a variation of server-side composition but occurs closer to the user at the “edge” of the network. This approach improves performance by offloading composition from the browser, enabling efficient caching and faster content delivery.

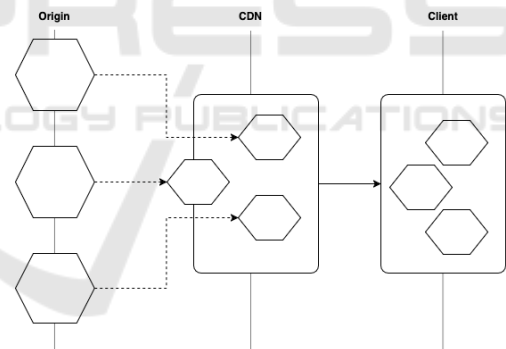


Figure 7: Edge-Side Composition - Adapted from (Mezzalana, 2021).

Edge-Side Composition allows for selective caching strategies, where frequently accessed components can be stored at edge locations, reducing the need for repeated requests to the origin server. This is particularly beneficial for globally distributed users, minimizing load times. Study PS20 discusses how Edge Side facilitate the assembly of views at the CDN level, as shown in Figure 7, using placeholders replaced with valid HTML during processing. In the context of Micro-Frontends, this approach enables faster content delivery by distributing composition tasks closer to the user. Since composition occurs before reaching the client, real-time personal-

ization and user-specific adaptations may be limited, requiring additional client-side logic to mitigate these restrictions.

4.2 RQ.2. What Are the Impacts of Adopting a Micro-Frontends Architecture?

The implementation of micro-frontends presents a perspective of bringing benefits such as scalability, flexibility, and the ability for independent development. However, it is important to analyze the negative impacts that may arise, notably regarding the complexity in configuration and coordination of the architecture. Figure 8 presents the number of studies that addressed both the positive and negative impacts of adopting the micro-frontends architecture, categorizing them for a holistic analysis. It is important to note that we grouped the strategies of Decoupling, Scalability, Productivity, Flexibility, and Performance into the “Application Efficiency” strategy, and the strategies Components, Conflicts, and Code Redundancy into the “Reusability” strategy (Figure 8).

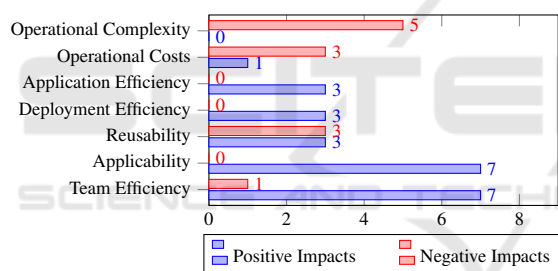


Figure 8: Positive and Negative Impacts.

Positive Impacts. The *Team Efficiency* (Figure 8) resulting from greater independence, scalability, and faster deliveries stood out as one of the most mentioned impacts by the authors of studies (PS1, PS8, PS9, PS10, PS11, PS12, and PS22). This impact was highlighted across multiple studies, emphasizing how micro-frontends improve development efficiency by enabling teams to work more autonomously and deliver results more rapidly.

Additionally, the *Applicability* of micro-frontends was extensively studied. Studies PS3, PS4, PS6, PS14, PS16, PS19, and PS21 developed prototypes and use cases in various business scenarios where the micro-frontends approach was successfully adopted. These studies demonstrated positive outcomes in industries such as the Human-Machine Interface (HMI) (PS14), industrial plants (PS3), and Internet of Things (IoT) (PS16). Moreover, Study PS6 presented benefits in the effective implementation of chatbots, and

PS21 reported the successful construction of platforms designed for organizing functionalities associated with microservices and micro-frontends.

The *Reusability* of components and plugins (PS2, PS5, and PS17) was also identified as an influential practice in enhancing team velocity and ensuring interface standardization, thus accelerating development cycles. In parallel, studies PS10, PS12, and PS22 highlighted positive impacts on *Deployment Efficiency*, allowing for independent implementations, risk reduction, facilitated rollback, and continuous updates. These benefits contribute to a more agile development cycle and a more resilient application to changes.

Furthermore, Study PS20 highlighted positive points related to the decoupling and scalability of applications, which led to cost reduction and improved user experience, further illustrating the advantages of adopting micro-frontends. Studies PS1 and PS15 also mentioned improvements in *Application Efficiency* as positive outcomes of the approach.

Finally, one particular study, PS21, highlighted the improved application performance as a positive outcome, which in turn led to a reduction in *Operational Costs*. This shift offered greater flexibility and customization, allowing for a more efficient use of resources. Manufacturers were able to leverage external expertise and specialized solutions, optimizing their processes while maintaining cost efficiency through a pay-as-you-go model.

Negative Impacts. The negative impacts associated with the adoption of the micro-frontends are related to technical and operational complexities, including the implementation of pipelines, and reusability, where challenges such as conflict resolution and code redundancy become evident (Figure 8). The studies PS2, PS10, PS12, PS20 and PS21 highlighted the increase in *Operational Complexity* as a negative impact resulting from the adoption of the micro-frontends architecture. For example, the study PS12 compared two implementation strategies of micro-frontends using the Iframes and Module Federation patterns. The results showed that, in a decentralized architecture, scaling the release and deployment processes becomes essential to support multiple applications. This, in turn, introduces a new type of complexity. This increase in complexity was attributed, in part, to the considerable effort required in configuring environments, as mentioned by study PS20. This complexity can result in a steep learning curve and require efficient configuration management practices.

Regarding *Team Efficiency*, while some studies highlight potential improvements in software devel-

opment team management—such as increased autonomy, which allows teams to focus on their specific domain and make independent architecture and implementation decisions—others reveal significant challenges. Study PS17, for instance, demonstrated that while micro-frontend architecture provided benefits in rewriting the Resident Portal application, it also introduced complexities that impacted efficiency. These challenges included increased coordination overhead, the need for standardized communication between teams, and potential inconsistencies in user experience due to decentralized development.

Furthermore, although sources such as Geers (Geers, 2020) suggest that micro-frontend architecture is ideal for medium to large-scale applications, its adoption in smaller organizations, as observed in study PS1, led to an increase in payload size and the number of requests to servers, thereby raising operational costs. Additionally, studies PS6 and PS17 also highlight that the necessity of maintaining multiple independent frontends results in duplicated efforts and higher operational burdens. Collectively, these studies indicate that while micro-frontend architecture offers advantages, it also introduces trade-offs that can significantly increase *Operational Costs*, particularly for teams with limited resources.

Regarding *Reusability*, the negative impact was emphasized by the presence of redundancy and conflicts in the source code, as evidenced in studies PS5, PS11, and PS13. These issues highlight the importance of adopting effective coding and reuse techniques. In this context, study PS6 suggests that componentization, along with the use of libraries and plugins, can help mitigate redundancy and minimize conflicts in the code.

4.3 RQ.3. What Challenges Do Software Development Teams Encounter when Adopting Micro-Frontends?

According to the selected studies, the challenges faced by software development teams when adopting micro-frontends are related to Team Coordination, Environment Configuration, Application Efficiency, and Reusability.

Challenges in Team Coordination. As different teams may be developing independent micro-frontends, coordination between these teams can become complex. It is essential to establish effective communication and collaboration strategies to prevent conflicts and ensure software cohesion. Only

study PS7 reported team coordination as a challenge, likely due to the project's small scale.

Challenges in Environment Configuration. Configuring environments to support the continuous execution and integration of multiple micro-frontends can be challenging. Environment configuration for micro-frontends involves both technical complexity and operational costs, as presented in Figure 8. These challenges were also reported in studies PS7, PS9, PS12, PS15, and PS18. Furthermore, studies PS1, PS3, PS5, PS13, PS16, and PS15 emphasized the importance of a strategic and proactive approach to overcoming orchestration challenges, considering project-specific factors such as requirements, technologies involved (Docker and Kubernetes), and intended scalability.

Challenges in Application Efficiency. This challenge includes aspects inherent to software development, used to achieve results such as decoupling, scalability, productivity, flexibility, and performance. The aspects include state management, communication patterns, navigation and routing, composition, and user experience, including developer experience. These challenges were identified in studies PS1, PS10, PS15, PS18, PS19, PS20, and PS21.

Challenges in Reusability. Studies PS22, PS8, and PS13 identified challenges related to the complexity of code and component reuse, including difficulty in efficiently managing CSS style conflicts and code redundancy (PS13). These challenges highlight the inherent difficulty in integrating various frontends and teams working on a single system, even when divided among different teams and micro-applications, as the system must ultimately appear seamless for a better user experience. One solution to mitigating style conflicts and avoiding code duplication is the adoption of a Design System (studies PS2, PS3, PS10, PS11, PS12, PS13, PS15, PS18, and PS21).

5 DISCUSSION

Micro-frontends have emerged as a promising architectural approach to address the challenges of modern software development frontend, particularly in fostering agility, scalability, and independent deployment of frontend components. This systematic mapping study (SMS) synthesized a broad range of literature to examine the architectural patterns, implementation

strategies, impacts, and challenges associated with micro-frontends.

The analysis of architectural views shows a strategic approach by organizations to enhance modularity and scalability while optimizing communication between components. The widespread adoption of API views underscores the role of APIs as pivotal in enabling independent micro-frontends, facilitating seamless backend communication and supporting efficient development and maintenance practices across diverse frameworks and technologies. Similarly, views such as Single-SPA and Web Components illustrate the flexibility and modularity achievable without strict dependencies on specific frameworks, promoting adaptability and simplifying the integration of new features. Containerization and orchestration views, alongside emerging concepts such as Module Federation, further illustrate evolving practices aimed at maximizing code reuse and operational efficiency. Overall, these architectural visions underscore the adaptability and strategic value of micro-frontend architectures in modern software development landscapes.

In parallel, the study of architectural patterns reveals a comprehensive landscape of strategies adopted by development teams. These patterns, ranging from framework choices (such as Angular, React, and Vue) to communication and composition patterns, underscore the flexibility and complexity inherent in micro-application development. Key patterns such as IFrame for deployment agility, Design System for interface standardization, and Routing for enhanced user experience illustrate the diverse approaches to managing frontend complexity and improving application scalability. These patterns may facilitate the integration, scalability, and maintainability of applications, while also highlighting the complexities involved. Understanding these patterns and their implications helps in making informed decisions during the adoption and implementation of micro-frontends, ensuring a balanced approach between innovation and operational efficiency.

The adoption of micro-frontends offers significant benefits to software development projects. Improved team efficiency, highlighted by autonomous development teams making independent architectural decisions, enhances productivity and responsiveness to market demands. Deployment efficiency is also notable, with micro-frontends enabling incremental updates, reduced deployment risks, and faster time-to-market. Furthermore, the scalability and flexibility afforded by micro-frontends cater to diverse application requirements, supporting the dynamic scaling of components based on user demand. This adaptability

is particularly advantageous in sectors requiring rapid iteration and deployment cycles, such as e-commerce and content management systems.

Despite its benefits, the adoption of micro-frontends introduces certain challenges. Technical complexity arises from managing multiple frontend technologies, necessitating reliable configuration management and infrastructure orchestration. Challenges in ensuring consistent user experiences across micro-frontends, especially in handling state management and interface coherence, require careful architectural planning and design. Moreover, concerns over code reusability and integration complexities highlight the importance of establishing clear development standards and enforcing best practices. Addressing these challenges requires ongoing refinement of architectural strategies, investment in developer training, and adoption of tools that streamline collaboration and code management.

6 THREATS TO VALIDITY

Construction Validity. We ensured clarity in definitions of concepts and terms used in the primary studies, focusing on views and patterns in micro-frontends architecture. We recognized the threat of conceptual divergences, such as different interpretations of component communication and variations in strategies such as Single-SPA or Module Federation. To mitigate this, we included definitions aligned with widely accepted practices, analyzed each study in its business context, and mapped relationships between perspectives and results in section 7.

Internal Validity. To address selection bias, we established inclusion criteria representing different implementation approaches. We also discussed confounding factors and emphasized the observational nature of included studies, which limits causal inference. Reusability and conceptual divergence can affect consistency in defining practices, but addressing divergences during analysis can enhance validity. Operational complexity and costs, influenced by implementation diversity, also impact results. We assessed the relationship between these factors to mitigate validity threats. Efficiency aspects such as decoupling, scalability, and performance vary by project size and complexity, requiring contextualized evaluation. Deployment strategies and team efficiency are critical in validating conclusions.

External Validity. Given the modular nature of micro-frontends, results from specific contexts may

not apply universally. The diversity in implementations means that findings from one context may not be directly applicable to others. We addressed this by transparently discussing the results and considering the variety of approaches used, which improved the external validity of our conclusions. The selected studies covered different scenarios, from large enterprise applications to smaller, agile setups, enhancing the sturdiness and representativeness of our findings. This approach allowed for more generalized insights that can guide best practices across various use cases.

7 CONCLUSIONS

In this study, we explored fundamental strategies and guidelines to address the challenges and maximize the benefits of adopting micro-frontends through a Systematic Mapping Study. The results emphasize the need for strategies to enhance the benefits and mitigate challenges associated with this emerging architecture. Based on the findings of this research and the identified gaps, a promising area for future exploration involves conducting more in-depth studies on specific aspects of micro-frontends architecture. This includes detailed investigations into communication strategies between micro-frontends, integration patterns, and state management techniques, aiming for a deeper understanding of their practical implications and potential improvements.

Additionally, expanding research through empirical studies to validate and deepen this work's conclusions is essential. Collecting quantitative and qualitative data in real software development environments will provide valuable insights into the performance, scalability, and maintainability of micro-frontends in different contexts. A systematic comparison with similar architectures, such as microservices, will also help assess their advantages and disadvantages, guiding decisions on adopting and evolving micro-frontends.

Looking ahead, further research is needed to address emerging trends and challenges. Exploring advanced composition strategies, enhancing tooling support, and optimizing performance and scalability are key areas of focus. Empirical studies evaluating real-world implementations and comparative analyses across architectures will offer deeper insights into the long-term viability of micro-frontends.

REFERENCES

- Abdelfattah, A. S. and Cerný, T. (2023). Filling the gaps in microservice frontend communication: Case for new frontend patterns. In van Steen, M. and Pahl, C., editors, *Proceedings of the 13th International Conference on Cloud Computing and Services Science, CLOSER 2023, Prague, Czech Republic, April 26-28, 2023*, pages 184–193. SCITEPRESS.
- Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.*, 80(4):571–583.
- Bühler, F., Barzen, J., Harzenetter, L., Leymann, F., and Wundrack, P. (2022). Combining the best of two worlds: Microservices and micro frontends as basis for a new plugin architecture. In Barzen, J., Leymann, F., and Dustdar, S., editors, *Service-Oriented Computing - 16th Symposium and Summer School, SummerSOC 2022, Hersonissos, Crete, Greece, July 3-9, 2022, Revised Selected Papers*, volume 1603 of *Communications in Computer and Information Science*, pages 3–23. Springer.
- Gashi, E., Hyseni, D., Shabani, I., and Çiço, B. (2024). The advantages of micro-frontend architecture for developing web application. In *2024 13th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–5.
- Geers, M. (2020). *Micro Frontends in Action*. Manning Publications.
- Harms, H., Rogowski, C., and Iacono, L. L. (2017). Guidelines for adopting frontend architectures and patterns in microservices-based systems. In Bodden, E., Schäfer, W., van Deursen, A., and Zisman, A., editors, *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 902–907. ACM.
- Lewis, J. and Fowler, M. (2014). Microservices. Accessed on January 6, 2024.
- Lorenz, J., Lohse, C., and Urbas, L. (2021). Microfrontends as opportunity for process orchestration layer architecture in modular process plants. In *26th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2021, Vasteras, Sweden, September 7-10, 2021*, pages 1–4. IEEE.
- Männistö, T., Tuovinen, A., and Raatikainen, M. (2023). Experiences on a frameworkless micro-frontend architecture in a small organization. In *20th International Conference on Software Architecture, ICSA 2023 - Companion, L'Aquila, Italy, March 13-17, 2023*, pages 61–67. IEEE.
- Mena, M., Corral, A., Iribarne, L., and Criado, J. (2019). A progressive web application based on microservices combining geospatial data and the internet of things. *IEEE Access*, 7:104577–104590.
- Mezzalana, L. (2021). *Building Micro-Frontends*. O'Reilly Media.
- Mohammed, S., Fiaidhi, J., Sawyer, D., and Lamouchie, M. (2022). Developing a graphql SOAP conversa-

- tional micro frontends for the problem oriented medical record (QL4POMR). In *Proceedings of the 6th International Conference on Medical and Health Informatics, ICMHI 2022, Virtual Event, Japan, May 13-15, 2022*, pages 52–60. ACM.
- Moraes, F. R., Campos, G. N., de Almeida, N. R., and Affonso, F. J. (2024). Micro frontend-based development: Concepts, motivations, implementation principles, and an experience report. In Filipe, J., Smialek, M., Brodsky, A., and Hammoudi, S., editors, *Proceedings of the 26th International Conference on Enterprise Information Systems, ICEIS 2024, Angers, France, April 28-30, 2024, Volume 2*, pages 175–184, <https://doi.org/10.5220/0012627300003690>. SCITEPRESS.
- Nishizu, Y. and Kamina, T. (2022). Implementing micro frontends using signal-based web components. *J. Inf. Process.*, 30:505–512.
- Noppadol, N. and Limpiyakorn, Y. (2021). Application of micro-frontends to legal search engine web development. In Kim, H. and Kim, K. J., editors, *IT Convergence and Security*, pages 165–173, Singapore. Springer Singapore.
- Pavlenko, A., Askarbekuly, N., Megha, S., and Mazzara, M. (2020). Micro-frontends: application of microservices to web front-ends. *J. Internet Serv. Inf. Secur.*, 10(2):49–66.
- Peltonen, S., Mezzalira, L., and Taibi, D. (2021). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Inf. Softw. Technol.*, 136:106571.
- Perlin, R., Ebling, D., Maran, V., Descovi, G., and Machado, A. (2023). An approach to follow microservices principles in frontend. In *2023 IEEE 17th International Conference on Application of Information and Communication Technologies (AICT)*, pages 1–6.
- Petcu, A., Frunzete, M., and Stoichescu, D. (2023). Benefits, challenges, and performance analysis of a scalable web architecture based on micro-frontends. *UPB Scientific Bulletin, Series C: Electrical Engineering and Computer Science*, 85(3):319–334. cited By 0.
- Pölöskei, I. and Bub, U. (2021). Enterprise-level migration to micro frontends in a multi-vendor environment. *Acta Polytechnica Hungarica*, 18(8):7 – 25. Cited by: 4; All Open Access, Bronze Open Access.
- Rappl, F. and Schöttner, L. (2021). *The Art of Micro Frontends: Build websites using compositional UIs that grow naturally as your application scales*. Packt Publishing.
- Shakil, M. and Zoitl, A. (2020). Towards a modular architecture for industrial hmis. In *25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2020, Vienna, Austria, September 8-11, 2020*, pages 1267–1270. IEEE.
- Simões, B., del Puy Carretero, M., Santiago, J. M., Segovia, S. M., and Alcaín, N. (2023). Twinark: A unified framework for digital twins based on micro-frontends, micro-services, and web 3d. In Posada, J., Moreno, A., Jaspe, A., Muñoz-Pandiella, I., Stricker, D., Mouton, C., Mohammed, A., and Elizalde, A., editors, *The 28th International ACM Conference on 3D Web Technology, Web3D 2023, San Sebastian, Spain, October 9-11, 2023*, pages 9:1–9:10. ACM.
- Sorgalla, J., Wizenty, P., Rademacher, F., Sachweh, S., and Zündorf, A. (2021). Applying model-driven engineering to stimulate the adoption of devops processes in small and medium-sized development organizations. *SN Comput. Sci.*, 2(6):459.
- SPA, S. (2022). Single spa - getting started overview. Accessed on January 6, 2024.
- Stefanovska, E. and Trajkovik, V. (2022). Evaluating micro frontend approaches for code reusability. In Zdravkova, K. and Basnarkov, L., editors, *ICT Innovations 2022. Reshaping the Future Towards a New Normal - 14th International Conference, ICT Innovations 2022, Skopje, Macedonia, September 29 - October 1, 2022, Proceedings*, volume 1740 of *Communications in Computer and Information Science*, pages 93–106. Springer.
- Taibi, D. and Mezzalira, L. (2022). Micro-frontends: Principles, implementations, and pitfalls. *ACM SIGSOFT Softw. Eng. Notes*, 47(4):25–29.
- Tilak, P. Y., Yadav, V., Dharmendra, S. D., and Bolloju, N. (2020). A platform for enhancing application developer productivity using microservices and micro-frontends. In *2020 IEEE-HYDCON*, pages 1–4.
- Wang, D., Yang, D., Zhou, H., Wang, Y., Hong, D., Dong, Q., and Song, S. (2020). A novel application of educational management information system based on micro frontends. In Cristani, M., Toro, C., Zanni-Merk, C., Howlett, R. J., and Jain, L. C., editors, *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES-2020, Virtual Event, 16-18 September 2020*, volume 176 of *Procedia Computer Science*, pages 1567–1576. Elsevier.
- Wanjala, S. T. (2022). A framework for implementing micro frontend architecture. *Int. J. Web Eng. Technol.*, 17(4):337–352.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Systematic Literature Reviews*, chapter 4, pages 45–54. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Yang, C., Liu, C., and Su, Z. (2019). Research and application of micro frontends. *IOP Conference Series: Materials Science and Engineering*, 490(6):062082.
- Zhang, C., Zhang, D., Zhou, H., Wang, X., Lv, L., Zhang, R., Xie, Y., Liu, H., Li, Y., Jia, D., Huang, Y., and Jiang, T. (2023). Application business information interaction bus based on micro frontend framework. In *2023 7th International Symposium on Computer Science and Intelligent Control (ISCSIC)*, pages 306–310, Los Alamitos, CA, USA. IEEE Computer Society.