Exploring Micro Frontends: A Case Study Application in E-Commerce

RicardoHideki Hangai Kojo, Renato Cordeiro Ferreira, Luiz Real O. Corte Real Alfredo Goldman, Renato Cordeiro, and Thatiane de Oliveira Rosa

[1] University Of São Paulo, São Paulo, SP, Brazil
[2] Federal Institute of Tocantins, Paraíso do Tocantins, TO, Brazil
guerraem@gmail.com ricardokojo@alumni.usp.br
{gold,renatocf}@ime.usp.br {thatiane}@ifto.edu.br

Abstract. In the micro frontends architectural style, the frontend is divided into smaller components, which can range from a simple button to an entire page. The goal is to improve scalability, resilience, and team independence, albeit at the cost of increased complexity and infrastructure demands. This article seeks to understand when it is worth adopting micro frontends, particularly in the context of industry. To achieve this, we conducted an investigation into the state of the art of micro frontends, based on both academic and gray literature. We then implemented this architectural style in a marketplace company for handmade products, which already uses microservices, and finally evaluated the implementation through a semi-open questionnaire with the developers. At the studied marketplace company, the need for architectural change arose due to the tight coupling between their main system (a Java monolith) and a dedicated frontend system. Additionally, there were deprecated technologies and poor developer experience. To address these issues, the micro frontends architecture was adopted, along with the API Gateway and Backend for Frontend patterns, and technologies such as Svelte and Fastify. Although the adoption of micro frontends was successful, it was not strictly necessary to meet the company's needs. According to the analysis of the mixed questionnaire responses, other alternatives, such as a monolithic frontend, could have achieved comparable results. What made adopting micro frontends the most convenient choice in the company's context was the monolith strangulation and microservices adoption, which facilitated implementation through infrastructure reuse and knowledge sharing between teams.

Key words: software architecture, micro frontends, study case, experimental software engineering

1 Introduction

The microservices architecture, which emerged around 2012 (Lewis; Fowler, 2014), has grown in popularity in both academia and industry, being adopted by companies such as Amazon, Netflix, and Uber (Richardson, 2020). This architecture consists of breaking down an application into small, independent services that work together. The loose coupling between these services brings benefits such as maintainability, scalability, and resilience, key characteristics of large systems that need to be reliable, always available, and minimally prone to failure. However, this approach focuses solely on the backend of the application, leaving the frontend aside (Peltonen et al., 2021).

In this context, the micro frontend architecture has emerged, aiming to apply microservices concepts to the presentation layer of applications (Peltonen et al., 2021). In this approach, the frontend system is divided into small parts, which may range from a simple button to a complete page. These parts are composed either on the client side (e.g., in the user's browser), the edge side (e.g., a CDN or Content Delivery Network), or the server side (e.g., a cloud server), resulting in a unified and cohesive output (Mezzalira, 2020).

There is significant overlap between the benefits and challenges of microservices and micro frontends. Therefore, it is reasonable to consider adopting both architectures, leveraging existing knowledge and infrastructure. However, the concept of micro frontends is relatively recent — introduced in 2016 (Geers, 2017) — and there is a lack of academic studies on the subject. Unlike microservices, which benefit from well-established technologies and design patterns, micro frontends still have limited tool support. Consequently, numerous opportunities remain to be explored.

Thus, this study aims to identify when it is worthwhile to adopt the micro frontend architecture, particularly in the context of the software industry. To achieve this, both theoretical and practical activities were carried out.

To achieve this objective, the first step was to understand the state of the art of this architecture through a review of both academic literature, such as articles and books, and grey literature, including blog posts and recorded talks. Through a critical analysis, it was possible to identify the most comprehensive sources of knowledge, the main authors on the topic, and the existing gaps in current materials. In addition to the literature review, the micro frontends architecture was implemented at Company A (a fictitious name), a marketplace platform that already used microservices. This objective also included documenting and evaluating the implementation: understanding the reasons behind the architectural change, outlining the development history, including the new systems, and explaining the rationale for the decisions made. Finally, the impacts of adopting micro frontends were analyzed through a questionnaire applied to the involved team members.

2 Background

This section aims to present the essential concepts necessary for a comprehensive understanding of this study, assuming a basic knowledge of software architecture and architectural styles, especially micro frontends.

2.1 Software Architecture and Architectural Styles

Software architecture can be defined as *"the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both"* (Bass et al., 2013). It encompasses the foundations, properties, rules, and constraints that guide system design.

The purpose of defining a software architecture is to facilitate aspects such as development, deployment, operation, and maintenance (Martin, 2019), in addition to supporting the analysis of qualitative attributes such as scalability, resilience, security, and maintainability. Architectural decisions are therefore essential to the system's evolution. To identify the most suitable architecture for addressing specific quality attributes, various architectural styles and patterns are adopted (Richards, 2015).

An architectural style prescribes a constrained set of elements and relationships that can be used to define a system's overall structure (Richardson, 2018a). A system that conforms to such constraints can be considered an instance of that architectural style. It is important to note that a single application may incorporate multiple styles.

When choosing a specific architectural style, it is crucial to consider the motivations behind the decision, its expected benefits, and potential challenges. Architectural decisions tend to have significant implications, and as Brooks (1995) famously stated, there are no silver bullets in software development; no single architectural style offers universal solutions or only advantages.

This study explores the monolithic, microservices-based, and micro frontend architectural styles, which are particularly relevant for understanding the proposed research.

2.2 Characteristics of the Monolithic and Microservices-based Architectural Styles

In a monolithic architecture, the entire application is structured in a single layer containing all necessary logic for full system functionality, typically resulting in a single executable artifact. This architecture is simple and straightforward, making it a common choice for new projects. However, as the team and codebase grow, technical and organizational issues may emerge, such as maintenance difficulties, increased bug count, slow feature development, and scalability limitations. In such contexts, a microservices architecture may serve as a viable alternative (Richardson, 2018a).

According to Lewis and Fowler (2014), *"the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies"*.

In the literature, microservices are often described as an alternative to monolithic systems that face organizational and technical limitations. The transition from monolith to microservices can follow the Strangler Fig Pattern (Fowler, 2004), in which existing functionalities are gradually extracted into independent services until the monolith becomes obsolete and is replaced entirely. This process is gradual, as a full rewrite of the monolith is usually infeasible due to its size and complexity.

2.3 Micro Frontend Architecture

Micro frontends are *"an architectural style where independently deliverable frontend applications are composed into a greater whole"* (Jackson, 2019). The application is broken down into smaller and simpler parts that can be developed, tested, and deployed independently, while preserving a unified user experience.

According to Peltonen et al. (2021), *"micro-frontends extends the microservice architecture idea and many principles from microservices apply to Micro-Frontends"* Therefore, it is common to implement both architectures in parallel. Since backend and frontend are distinct domains, existing distributed system knowledge and infrastructure can be leveraged without conflict between services. For these reasons, micro frontends are considered a natural next step after the adoption of microservices (Yang et al., 2019).

Peltonen et al. (2021) and Geers (2020) identify several motivations for adopting a micro frontend architecture. These include the increasing complexity of frontend systems, growth of the codebase, and organizational challenges. Scalability is also a recurring concern, particularly regarding the need for independent deployments and teams, faster feature delivery, innovation enablement, and reduced onboarding time for new developers. In some cases, adoption is also influenced by market trends, with companies embracing microservices and micro frontends to align with industry best practices.

Furthermore, micro frontends offer benefits similar to those of microservices. These include technological flexibility, allowing each part of the system to use the most appropriate tools, and the ability to form cross-functional teams focused on specific business domains. The architecture promotes team and project autonomy, facilitating development, testing, and continuous delivery. It also supports scalability both at the application and organizational levels. Another relevant aspect is resilience: failures in a micro frontend affect only that service, avoiding the full-system outages common in monolithic architectures.

Regarding the challenges of adopting micro frontends, the most significant include the complexity introduced by the distributed nature of the architecture, which involves multiple projects and diverse technologies. Ensuring visual consistency across services is also critical, as the final product must appear cohesive and uniform to the user. Governance poses another challenge, especially when different teams share responsibility for the same service or operate within the same business domain. Finally, performance concerns must be addressed, due to potential code duplication, shared dependencies, and the volume of data transferred during interface composition (Peltonen et al., 2021; Geers, 2020).

Before implementing micro frontends, it is necessary to define which approach will be adopted. There are two main approaches: the horizontal approach, in which each micro frontend represents a portion of the page; and the vertical approach, in which each micro frontend corresponds to a business context or domain. In the latter, a single micro frontend may contain parts of a page, an entire page, or even multiple pages (Mezzalira, 2020).

Another important aspect to consider when implementing a micro frontend architecture is the adoption of architectural patterns. In this context, the most relevant patterns are API Gateway and BFF (Backend for Frontend).

The API Gateway is a service that acts as the entry point for other services. All requests made by clients — such as a mobile app or a desktop website — go through the gateway, which, in addition to routing them to the appropriate services, can also handle authorization checks and protocol translation. This pattern is particularly useful in microservices-based architectures, where multiple clients access various services. Communication with each service can occur through different protocols, such as HTTP and gRPC (Richardson, 2018b).

The Backend for Frontend (BFF) pattern is a variation of the API Gateway (Richardson, 2018b), in which each client type has its own dedicated gateway. This allows for the isolation of specific client needs and enables a team to take responsibility for both the client application and its corresponding BFF.

3 Architecture Overview of Company A

As an e-commerce platform, the frontend is a key element for the success of Company A (fictitious name). The website must be performative, user-friendly, and accessible across both desktop and mobile devices. Before adopting micro frontends, Company A was undergoing the strangler pattern process for its monolithic system, referred to here as Marketplace. While some backend functionalities had already been migrated to independent microservices, the frontend remained centralized in the monolith. Additionally, some UI components were served by Aquarelle (fictitious name), a separate project created to meet specific demands.

Marketplace, launched in 2012, was responsible for both backend and frontend services and operated as the single entry point for all systems, thus becoming a single point of failure. Built using Java, JSP, Sass, and internally developed JavaScript libraries, the monolithic architecture enabled rapid early-stage development. However, over time, it became a bottleneck due to increasing complexity, long deployment times, and architectural erosion. As part of the strangler pattern, core functionalities such as product search, shipping calculations, and category management were moved to microservices. Aquarelle, based on NodeJS, emerged to support a new chat feature between buyers and sellers, requiring a reactive interface capable of displaying dynamic backend data, such as order status and available user actions. Since modern frameworks were not available in early 2016, the solution involved developing internal libraries.

Despite its initial success, Aquarelle faced technical and architectural challenges. One of its key libraries was deprecated soon after launch, leading to difficulties in maintenance, a lack of documentation, and reduced developer interest due to outdated technologies. Moreover, development constraints, such as limited experience with NodeJS and insufficient time to decouple features from the monolith, resulted in a high degree of coupling with Marketplace. The monolith remained responsible for authentication, data orchestration, and formatting, forcing frontend developers to engage with backend code. This rigid architecture, coupled with the use of deprecated technologies, ultimately led to the decision to adopt a micro frontend architecture.

4 Implementing Micro Frontends at Company A

Due to the issues discussed in the previous section, a plan was devised for the implementation of a new frontend architecture based on the adoption of modern technologies. The chosen architecture was micro frontends, which facilitates the isolation of responsibilities, allows interfaces to be divided into smaller parts, and enables the use of different technologies. To support this, it was necessary to create a service that would act as an API Gateway, so that the monolith would no longer serve as the central entry point and would be accessed only when needed. Based on this plan, the projects Fortelle, Veloura, Nuvella, and Brindle (fictitious names) were created, each playing a strategic role in the construction of the new architecture.

Fortelle is the implementation of the API Gateway pattern, serving as the entry point for Company A's systems. All user requests pass through it, where routing, validations, and security functions for other services are handled. This implementation was essential to enable the micro frontends to start taking over the responsibility of serving user interfaces, previously centralized in the monolith. In parallel, the team began evaluating frameworks and libraries to ensure good performance and a positive developer experience.

Veloura is Company A's design system, consisting of a set of standards for building user interfaces—including colors, spacing, fonts, and icons—which reinforce the company's visual identity (Hacq, 2018). The Nuvella project was conceived as a proof of concept for the selected frontend technologies. Its development allowed the team to validate practices such as file organization, supporting libraries, and code conventions. From Nuvella, the first Brindle was created, with the goal of migrating the search page for a specific set of craft materials from the monolith to the new architecture. This page was selected because it was visually similar to the main search page but had a lower volume of traffic.

After deploying the first Brindle to production, new clones were developed to meet internal demands. All systems derived from Nuvella came to be known as Brindle projects, with additional names used to differentiate them, such as "Brindle Discovery", for example. Currently, all of Company A's micro frontends are Brindle projects. Figure 1 illustrates the basic structure of a micro frontend: user requests pass through the API Gateway and are redirected to a Brindle instance, where the BFF (Backend for Frontend) handles internal routing, orchestrates data from microservices, and forwards it to a template rendered by an open-source library developed by Company A.

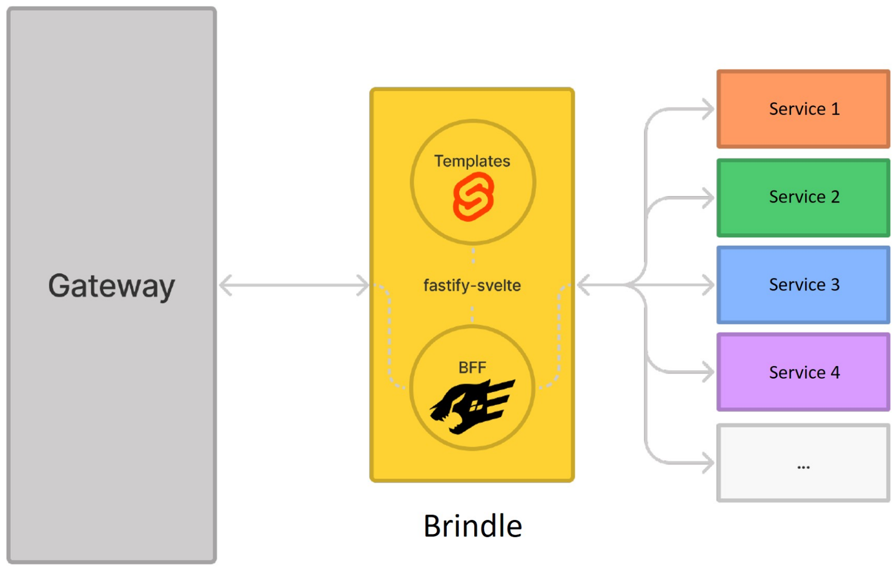

Fig. 1. Architectural schematic of a Brindle project.

With the creation of three Brindle projects, the frontend teams were able to identify common modules across services. These shared components were extracted into a new library, Brindle Core, an open-source framework for building Brindles. One of the main advantages of Brindle Core is its ability to abstract internal mechanisms that rarely change, reducing code duplication, the number of files, and the need for repeated testing in common components.

5 Current Architecture of Company A

With the conception of the mentioned projects, the company's frontend architecture underwent significant changes. The introduction of the API Gateway (Figure 2) enabled other services, besides the monolith, to handle part of the user request processing, decentralizing responsibilities and enabling the adoption of micro frontends. As a result, creating new Brindle projects became straightforward, requiring only minor additions to the API Gateway code.

The architectural transition at Company A proved successful. The adoption of vertical micro frontends allowed the migration of pages from the monolith to the Brindle projects, which utilize modern technologies and provide an improved development experience. Moreover, the new structure facilitated the creation of

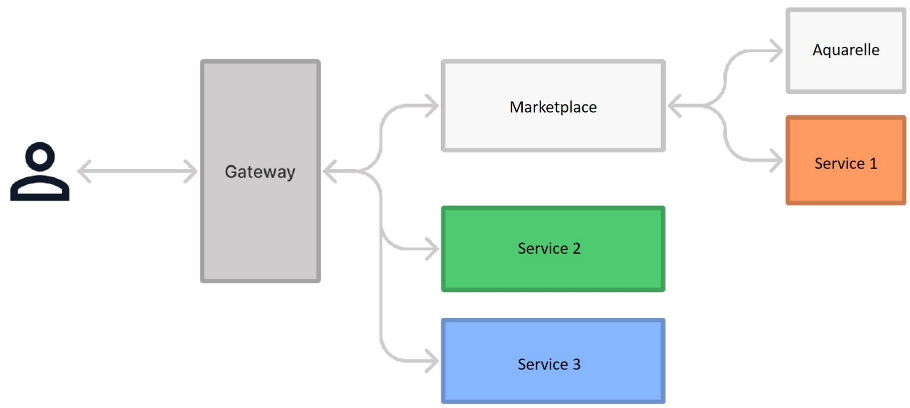

Fig. 2. Request flow to frontend after implementing micro frontends.

frontend systems for the company's new products. It is worth noting that the ongoing transformation towards microservices favored this adoption by enabling the reuse of technologies and infrastructure already in place, since micro frontends also share microservices' principles.

During this research, it was observed that the natural evolution of the architecture involves migrating more pages from the monolith to the new services, as well as creating additional micro frontends. However, there are no plans to expand the number of services, as Company A has defined a domain division based on its products. Therefore, until new products are conceived, the teams remain focused on improving existing services.

Nonetheless, a bottleneck exists: to migrate more pages to the Brindle projects, functionalities from the monolith must be converted into microservices. Since the backend teams currently lack the availability to develop these services, obtaining the necessary data for the interfaces is compromised. To mitigate this issue, backend and frontend teams must establish contracts, allowing simulation of calls to future services and advancing development. Despite adopting microservices and micro frontends, Company A is still evaluating the implementation of multidisciplinary teams, considering organizational impacts and questioning whether this practice is always beneficial, as discussed in the literature.

6 Survey with Company A Developers

To better understand the impacts resulting from the architectural change, a questionnaire was applied to Company A employees who were directly or indirectly involved with the frontend. The purpose of this section is to present the results obtained.

One of the goals of this study is to evaluate the implementation of micro frontends carried out by the company. In this sense, it is reasonable to turn to those directly involved: the developers. Therefore, a semi-open questionnaire was prepared to identify the problems faced, expectations regarding the change, and the impacts perceived by the respondents.

The questionnaire was structured to separate, as much as possible, the impacts of adopting new technologies from those resulting from the new architecture. Since these two complex transformations occurred simultaneously, it was expected that developers might have difficulty distinguishing their responses without this methodological precaution. Nineteen questions were formulated, fifteen open-ended and four multiple-choice. For the multiple-choice questions, a Likert scale was used.

These questions were distributed into five sections. The first addresses information about the respondent, such as professional experience and involvement with Company A's frontend. The second covers the architecture in place prior to the adoption of micro frontends. Sections three and four focus, respectively, on the use of new technologies and the architectural change. Finally, the fifth section aims to understand the participants' comprehension of the new architecture.

The multiple-choice questions asked respondents to consider both technological and architectural impacts across seven aspects: hiring and onboarding; deployment; development across multiple teams; project understanding; implementation of new features; testability; and development speed. These aspects were chosen due to their relevance to the company and their frequent presence in the reviewed literature.

A total of eight responses were obtained. Only one woman responded to the questionnaire. Most participants (62.5%) have more than 10 years of experience in the technology field. Half work as frontend developers. Engineering managers and frontend team leaders also participated in the study.

When asked about their involvement with the frontend, whether in development or architectural decisions, respondents rated themselves between medium and high levels. This result is consistent with expectations, considering that developers deal directly with the frontend, while engineering managers have a broader view and participate more indirectly.

When asked about the new architecture, some respondents answered "I don't know how to answer", and none selected "No impact". This indicates a lack of clarity about the perceived architectural impacts. Among the analyzed aspects, testability and the hiring/onboarding processes were most negatively affected. Regarding testability, although tests are no longer coupled to the monolith, end-to-end testing in distributed systems, such as those based on microservices and micro frontends, is inherently more complex. As for hiring and onboarding, service fragmentation can facilitate a piecemeal introduction when necessary; however, understanding the system as a whole becomes notably more difficult, requiring more adaptation time.

For the open-ended questions, responses were individually analyzed, and relevant excerpts were coded. After a full analysis, the codes were grouped into themes such as old architecture, new architecture, and micro frontends. Each theme was subdivided to better organize the emerging topics.

Regarding the old architecture, the objective was to identify its main strengths and weaknesses, as well as to verify which aspects respondents expected to see improved, that is, the focus was on expectations related to the changes. For the new architecture, the aim was to understand the perceived advantages and disadvantages following the adoption of micro frontends, with emphasis on the impacts on participants' daily activities. Lastly, the study also investigated participants' understanding of the micro frontend concept, their perception of the new architecture, and whether they recognize it as a legitimate implementation of this approach.

With respect to the old architecture, a significant number of disadvantages were noted. However, it is important to highlight that some of the points raised may be more closely related to the technologies used than to the architecture itself. Some confusion between technological and architectural impacts in the responses was expected. Among the advantages mentioned, most were associated with Aquarelle, which, despite achieving its goals, suffers from both technological and architectural limitations. The most cited disadvantage was the strong coupling between frontend systems, precisely the main reason behind the architectural change and the aspect for which improvement was most anticipated.

Regarding the new architecture, there were more mentions of advantages than disadvantages. Some points were cited in both positive and negative lights: the use of an emerging architecture attracts developers, and breaking the system into smaller services may ease onboarding. However, the inherent complexity of distributed systems can prolong the onboarding process. Moreover, although implementation speed has increased, the amount of code to be developed has also grown.

A noteworthy finding is that half of the participants do not recognize the new architecture as a micro frontend implementation. It can be concluded that two main factors contribute to this: the prevailing definitions in the literature and the lack of explicit references to the vertical slicing approach. Many definitions suggest that a single page must be split into multiple visual components for it to qualify as a micro frontend, which is not necessarily true. As with microservices, there is no rigid definition of what constitutes a "service". Therefore, it is possible to have micro frontends that do not even involve visual interfaces.

The analysis also provided evidence that, despite the successful implementation of micro frontends, this was not the only viable architectural alternative for Company A's context. A monolithic frontend, for instance, could work, provided that the interfaces were decoupled and modern technologies were used. One point that reinforces this argument is the fact that only one participant mentioned scalability, one of the main benefits of microservice-based architectures and, consequently, of micro frontends.

7 Conclusion

This article aimed to understand when it is worthwhile to adopt micro frontends in industry. Initially, a state-of-the-art study on the architecture was conducted through both academic literature and gray literature sources, such as blog posts and conference talks. Next, micro frontends were implemented at Company A, and finally, the adoption was evaluated through surveys with the company's developers. The literature on the topic remains limited, especially regarding the concept of vertical, or domain-based, micro frontends as described by Mezzalira (2020), whose simplicity facilitates implementation. Furthermore, existing definitions lack flexibility and generally tie the architecture to the decomposition of a page into components, which caused confusion within the company. Although a vertical approach was adopted, developers hesitated to classify it as a micro frontend architecture due to the absence of composition, a central focus in works such as Geers (2020). Therefore, there is a need to align the definition of the architecture more closely with that of its predecessor, microservices, and to document alternative forms such as micro frontends without user interfaces.

At Company A, all prerequisites were met, and the adoption was considered successful. The API Gateway pattern enabled a distribution of frontend responsibilities, while the BFF (Backend for Frontend) pattern supported communication with backend services. Still, adopting micro frontends was not strictly necessary, as other architectures, such as a monolithic frontend, could have achieved comparable results. What made the transition more feasible was the company's ongoing migration to microservices, which allowed teams to reuse infrastructure and share existing knowledge. Thus, while not the only possible solution, micro frontends proved to be the most convenient within that specific context. It is believed that this article may support researchers and practitioners alike by combining theory and practice, offering a historical account of implementation in an e-commerce setting, and providing a critical perspective on the literature. Additionally, it is hoped that this study contributed to architectural discussions at Company A and serves as documentation for current and future developers.

References

Len Bass, Paul Clements e Rick Kazman. Software Architecture in Practice. 3a. Addison Wesley, 2013.

M. E. Conway. "How do committees invent". In: Datamation 14.4 (1968), pgs. 28–31.

Martin Fowler. StranglerFigApplication. `https://martinfowler.com/bliki/StranglerFigApplication.html`. 2004.

Michael Geers. Micro Frontends. `https://micro-frontends.org/`. 2017

Michael Geers. Micro Frontends in Action. 1a. Manning, 2020.

Cam Jackson. Micro Frontends. `https://martinfowler.com/articles/micro-frontends.html`. 2019.

James Lewis e Martin Fowler. Microservices. `https://www.martinfowler.com/articles/microservices.html`.2014.

Luca Mezzalira. Front-End Reactive Architectures. Apress, 2018.
Luca Mezzalira. Lessons from DAZN: Scaling Your Project with Micro-Frontends. https://www.infoq.com/presentations/dazn-microfrontend/. 2020.
Luca Mezzalira. Building Micro-Frontends. O'Reilley Media, Inc., 2021.
Severi Peltonen, Luca Mezzalira e Davide Taibi. "Motivations, benefits, and issues for adopting micro-frontends: a multivocal literature review". In: Information and Software Technology 136 (2021), pg. 106571.
Mark Richards. Software Architecture Patterns. 1a. O'Reilley Media, Inc., 2015.
Chris Richardson. Microservices Patterns. 1a. Manning, 2018a.
Chris Richardson. Pattern: API Gateway Backends for Frontends. https://microservices.io/patterns/apigateway.html. 2018b.
Chris Richardson. Pattern: Monolithic Architecture. https://microservices.io/patterns/monolithic.html. 2019.
Chris Richardson. Who is using microservices. https://microservices.io/articles/whoisusingmicroservices.html. 2020.