Micro Frontends: Definition, Classifications, Challenges, Implementation Approaches and Communication Methods

João L. Castro
Universidade Federal de São Carlos
São Carlos, São Paulo, Brasil
joaocastro@estudante.ufscar.br

Daniel Lucrédio
Universidade Federal de São Carlos
São Carlos, São Paulo, Brasil
daniel.lucredio@ufscar.br

Abstract

Micro Frontend Architecture (MFA) seeks to decompose monolithic frontends into autonomous, orchestrated modules. Despite growing adoption, the field lacks a universally accepted definition and a cohesive classification framework. This Systematic Mapping Study (SMS) synthesizes current academic perspectives on MFA definitions, characteristics, classifications, challenges, implementation approaches and communication methods. Results indicate that while a consensus is absent, four key attributes (modularity, independence, microservice heritage and technological agnosticism) form the paradigm's core. This study contributes a multidimensional taxonomy based on composition, division, integration, and communication methods. Furthermore, the technical landscape is mapped through the identification of the main implementation approaches, of which five are most recurrent (Web Components, iframes, Module Federation, Single-SPA and route distribution) and several critical challenges, among which six stand out (integration, complexity, communication, performance, dependency management and User Experience (UX) consistency). This research clarifies the MFA conceptual landscape and provides a roadmap for addressing the inherent complexities of decentralized frontend architectures.

Keywords

Micro Frontends, Software Engineering, Web Development, Micro Architecture, Systematic Mapping Study

1 Introduction

Contemporary web applications require continuous availability, scalability, and fault isolation [11, 15, 34, 44]. To meet these needs, the microservice architectural style is frequently employed on the backend, promoting independence, allowing for heterogeneous technologies, and aligning development with business domains [11, 15, 19, 25, 28, 31, 34].

Many systems implement microservice backends while maintaining monolithic frontends, thereby preserving architectural limitations within the presentation layer [15, 27–31, 34]. To extend microservice benefits to the view layer, the Micro Frontend Architecture (MFA) emerged as a specialized architectural alternative [11, 15, 19, 24, 25, 29–32, 34, 44].

Despite growing interest since its introduction in 2016 [11], micro frontend definitions remain fragmented. This lack of conceptual consensus represents a significant gap, as a concise and universally accepted definition of the paradigm is still elusive [25].

Recent literature—including case studies, reviews, and mappings [25, 27, 31, 34]—offers valuable insights into MFA. However, these works do not establish a precise or unified definition for the paradigm, reinforcing the relevance of this research.

The conceptual landscape of micro frontends is fragmented by varying classification criteria. Studies such as those by Moraes et al. [25] and Gashi et al. [11] disagree on terminology and scope for module distribution, while others focus strictly on communication mechanisms (e.g., publish-subscribe) [4]. This suggests that current classifications are either too narrow or context-specific, highlighting the need for a more comprehensive architectural framework.

Furthermore, micro frontends present significant technical and managerial challenges [1, 25, 30, 32, 34]. Systematically mapping these limitations is vital for guiding future research and architectural evolution. Grounded on this necessity [34], this study analyzes these challenges to help refine current architectural practices.

This article aims to systematize MFA definitions, characteristics, classifications, challenges, implementation, and communication protocols. To fulfill these objectives, this Systematic Mapping Study (SMS) proposes a MFA definition synthesized from peer-reviewed literature, guided by the following Research Questions (RQs):

RQ1: What is MFA defined in current peer-reviewed research?

RQ2: What are the essential characteristics and principles of MFA?

RQ3: What classification models for MFA have been proposed in the literature?

RQ4: What are the main challenges associated with MFA?

RQ5: What implementation approaches for MFA are most frequently reported in the literature?

RQ6: What communication mechanisms are most commonly adopted in MFA?

While the analysis confirms that a consensual definition for micro frontends has yet to emerge, it highlights four core characteristics that underpin the architecture: **modularity**, **independence**, **microservice heritage**, and **technological agnosticism**. Expanding on these foundational traits, this study establishes a MFA taxonomy consisting of four primary classifications: **composition** (*server-side*, *edge-side*, *client-side*, and *hybrid*), **division** (*horizontal* and *vertical*), **integration** (*runtime* and *build-time*), and **communication** (*publish-subscribe*, *shared state*, and *API combination*). However, the diversity of these architectural choices often introduces significant operational hurdles; among the various challenges identified, six recurrent issues stand out: **integration and management of application parts**, **high system complexity**, **system communication**, **performance constraints**, **dependency management**, and **User Experience (UX) consistency**. To navigate these complexities, practitioners employ several technical strategies, with the five most frequent implementation methods being **Web Components**, **iframes**, **Module Federation**, **Single-SPA**,

and **route distribution**. Finally, within these implemented environments, **event emitters** have emerged as the most prevalent mechanism for facilitating decoupled communication.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 provides a detailed account of the methodology employed for data collection and analysis. The core findings and insights derived from this process are presented in Section 4. Section 5 addresses the potential limitations and threats to the validity of the study. Finally, Section 6 synthesizes the primary contributions and future work.

2 Related Work

The existing body of secondary studies offers valuable baselines for understanding MFA. For instance, Peltonen et al. [31] conducted a multivocal review across 43 sources (40 from gray literature and 3 from peer-reviewed studies) to map early benefits, such as technology independence, alongside initial architectural and organizational challenges. However, because their findings heavily rely on gray literature collected five years ago, our study focuses strictly on peer-reviewed literature to validate and update how these challenges have since evolved (RQ4). Furthermore, Santos et al. [34] conducted a systematic mapping study covering integration (Q4) and security (Q8), aligning with RQ3, RQ4, and RQ5. Regarding integration, they concluded that autonomous teams favor independent deployment using Module Federation, alongside application shells or iframes for runtime composition. On security, their findings highlight critical vulnerabilities in permission management, cross-site scripting (XSS), and inter-component communication. For standalone definitions (RQ1), Moraes et al. [25] conceptualized MFA as a "microservices-inspired architectural style [composed of] decoupled frontend applications [...] modeled around a business domain" [25, p. 177]. However, while their mapping establishes this theoretical baseline, our study goes further by evaluating how this conceptual definition translates into the practical classifications (RQ3) and architectural challenges (RQ4) reported in the literature.

While these studies provide essential viewpoints on isolated dimensions of the paradigm, the literature still benefits from an overarching synthesis. This study extends these existing foundations by establishing a unified conceptual, architectural, and taxonomic framework that connects definitions to practical challenges.

3 Methodological Approach

To structure the micro frontend domain, a SMS was conducted following the guidelines of Kitchenham [16]. Although originally designed for systematic reviews, adapting these principles ensures a transparent framework for interpreting the literature. This section details the corresponding research objectives, search sources, selection criteria, and quality assessment.

3.1 Search Approach

Primary sources were retrieved from high-impact databases: IEEE Xplore, ACM Digital Library, SpringerLink, and ScienceDirect. The mapping was managed using the "State of the Art through Systematic Review" (StArt) tool [9][1], which facilitates the systematic organization and analysis of literature.

[1]https://www.lapes.ufscar.br/resources/tools-1/start-1

To capture all spelling variations across publication titles and keywords, the search string was defined as: "**[Publication Title: micro front-end] OR [Publication Title: microfront-end] OR [Publication Title: microfrontend] OR [Publication Title: micro frontend] OR [Keywords: micro front-end] OR [Keywords: microfront-end] OR [Keywords: microfrontend] OR [Keywords: micro frontend]**". Terminology is standardized to the "micro frontend" format.

Executed on October 17, 2025, this query yielded 73 articles, comprising 54 papers from IEEE Xplore, 13 from ScienceDirect, 4 from ACM Digital Library, and 2 from SpringerLink.

3.2 Application Of Inclusion/Exclusion Criteria

The selection process relied on three exclusion criteria: languages other than English or Portuguese (EC1), gray literature (EC2), and the absence of "micro frontend" (or any spelling variation) in titles or keywords (EC3). Additionally, one inclusion criterion required software engineering relevance (IC1). Although the search string targeted MFA, EC3 provided a critical verification layer to eliminate partial database matches for individual terms.

The pipeline began with the automated removal of two duplicates via the StArt tool. Next, phase 1 (EC2) excluded four book chapters (3 IEEE, 1 ScienceDirect). Phase 2 then screened titles, keywords, and abstracts, rejecting 37 papers (28 IEEE, 9 ScienceDirect) that targeted unrelated fields (e.g., energy conversion, computer architecture, nuclear science) or represented partial keyword matches. In total, 43 papers were excluded, leaving 30 accepted primary studies for the mapping.

3.3 Quality Assessment Criteria

Post-selection, the remaining papers were quality-assessed to ensure credibility, clarity, and methodological rigor using the following Quality Assessment Questions (QAQs):

QAQ1: Are the research objectives and the study's context clearly defined and easy to follow?

QAQ2: Does the study provide a comprehensive and up-to-date literature review that justifies the research gap?

QAQ3: Does the study present a critical synthesis of the findings rather than a purely descriptive summary of the results?

QAQ4: Does the study discuss its findings in the context of the current state of practice, utilizing or referencing industry-standard frameworks and tools?

QAQ5: Are the research findings supported by empirical evidence (case studies, experiments, or Proof of Concepts (PoCs))?

Primary studies were scored against the aforementioned criteria using a three-point scale: 1 (fully addressed), 0.5 (partially), or 0 (not addressed). A 50% cutoff threshold (2.5) was applied; studies scoring below this mark were excluded from the final synthesis. Table 1 details the final scores.

The 30 studies surpassed the 50% threshold, confirming methodological rigor. While context was universally well-defined (QAQ1), varying empirical evidence (QAQ5) and analysis (QAQ3) indicate a blend of theoretical and practical approaches. High industrial alignment (QAQ4) grounds these findings in professional standards.

Table 1: Quality assessment scores of each paper.

Study	QAQ1	QAQ2	QAQ3	QAQ4	QAQ5	Total
1	1	1	0.5	1	0.5	4
2	1	1	1	1	1	5
3	1	0.5	1	1	1	4.5
4	1	0.5	1	1	1	4.5
5	1	1	1	1	0.5	4.5
6	1	1	1	0.5	0.5	4
7	1	0.5	1	1	1	4.5
8	1	0.5	0.5	1	0.5	3.5
9	1	0.5	0.5	1	0.5	3.5
10	1	0.5	1	1	1	4.5
11	1	0.5	0.5	1	1	4
12	1	0.5	1	1	1	4.5
13	1	1	1	1	0.5	4.5
14	1	1	1	1	1	5
15	1	1	1	1	1	5
16	1	1	1	1	1	5
17	1	1	1	1	1	5
18	1	1	1	1	1	5
19	1	1	1	1	1	5
20	1	1	1	1	1	5
21	1	1	1	1	1	5
22	1	0.5	0.5	1	0	3
23	1	0.5	1	1	1	4.5
24	1	0.5	1	1	1	4.5
25	1	0.5	1	1	1	4.5
26	1	1	1	1	1	5
27	1	1	0.5	1	1	4.5
28	1	1	1	1	1	5
29	1	0.5	1	0.5	1	4
30	1	1	1	1	1	5

3.4 Final Selection & Data Extraction

Data were extracted from the 30 studies—named from A1 to A30—to address the RQs; full titles and authors appear in Table 2

One notable inclusion in this mapping is the Multivocal Literature Review (MLR) conducted by Peltonen et al. [31], which bridges the gap between academic research and industrial practice. This study was retained specifically because it synthesizes "data extracted from 43 selected sources including 3 peer-reviewed academic papers and 40 gray literature sources" [31, p. 8]. Its inclusion is essential for capturing the practitioner-led experiences and real-world insights that define the current micro frontend landscape, where industrial adoption often outpaces academia.

Data analysis followed an inductive coding approach via *In Vivo* techniques [33], where codes serve as a "summative, salient, essence-capturing" attribute of the data and a "critical link" between collection and interpretation [33, p. 3]. This approach facilitated data grouping by similarity [33], allowing preliminary codes to be iteratively refined and synthesized into the broader core themes detailed in Section 4.

Codes were derived through keyword-based and contextual analysis to align the literature with the RQs and mitigate terminological inconsistencies across primary studies. Consequently, an initial pool of 189 codes (including sub-codes) was consolidated into 83 final entries after merging 106 items, with the resulting codebook detailed in Section 4.

A snowballing phase was omitted because conceptual saturation was achieved within the 30 primary studies. High-level categories stabilized near the sample's end: the final new codes emerged from study A28, while A29 and A30 merely reinforced established concepts. Consequently, a snowballing search would have yielded diminishing returns rather than novel conceptual insights.

4 Results

Publication trends (2019 – 2025) contextualize the findings before addressing the RQs. Output remained stable during 2019–2022 (2 papers annually), rising to 4 in 2023. Activity peaked in 2024 (13 papers), followed by 6 identified through mid-October 2025. This latter figure likely underrepresents total output due to search cutoffs and indexing lags. Despite a decline from the 2024 peak (1.08 articles/month), the 2025 rate (0.63) exceeds those of 2023 (0.33) and 2022 (0.17), indicating sustained interest. Future data will determine if the field is stabilizing or nearing a new peak.

4.1 RQ1: What Is MFA Defined In Current Peer-Reviewed Research?

This section addresses two dimensions of the micro frontend definition: the historical origin of the terminology and the conceptualizations proposed by the primary studies.

4.1.1 Origin Of MFA. Regarding the origin of micro frontends, nine articles offer historical context, attributing the concept to three sources: ThoughtWorks, Cam Jackson, or an unattributed 2016 emergence. Specifically, several studies [3, 4, 8, 11, 31, 46] credit a 2016 ThoughtWorks technical report, whereas others [10, 12] identify a general 2016 emergence without a specific creator. Uniquely, Capdepon et al. [5] blend these perspectives, attributing the paradigm's emergence to ThoughtWorks and its consolidated conceptualization to Cam Jackson, thus distinguishing the term's introduction from its subsequent development.

4.1.2 Consolidated Definition Of MFA. Although all studies characterize MFA, only 16 explicitly define the term [3–5, 7, 8, 10, 11, 13, 14, 18, 20, 31, 38, 39, 42, 46]. These definitions primarily focus on modular decomposition (12 studies), compositional frameworks (6), an evolutionary link with microservices (3), or end-to-end vertical integration (1).

Tilak et al. [42] define MFA as "an architectural style where independently deliverable frontend applications are composed into a greater whole" [42, p. 2], emphasizing modularity and the operational independence of each module as the fundamental characteristics of the approach.

Kunštnár and Podhorský [18] define MFA as a design approach centered on breaking down monolithic client applications into smaller, modular parts. Stepanov and Klym [39] also focus on modularity, noting that these smaller applications are more manageable. Gashi et al. [11] incorporate an organizational dimension to the definition, specifying that individual teams can be responsible for specific functionalities or interface features.

Kambala [13] and da Silva et al. [8] utilize equivalent definitions, emphasizing the architecture's capacity to enhance flexibility, scalability, and maintainability. From a similar definition, Cao et al. [4], Chen [7] and Liu et al. [20] highlight modularity alongside team and deployment independence as core pillars of the approach. Peltonen et al. [31] acknowledge the modularity and independence, but define the paradigm as an extension of microservices into the frontend domain, that same perspective being presented in Fu et al.

Table 2: List of accepted papers.

ID	Title	Authors
A1	A platform for enhancing application developer productivity using microservices and micro frontends	Tilak et al. [42]
A2	Micro Frontend Application Shell and Module Federation Architecture Implementation and Comparison	Bahiense-Junior et al. [2]
A3	Micro Frontend Architecture	Kunštnár and Podhorský [18]
A4	Dynamic Micro Frontends	Kičić et al. [17]
A5	Toward Bundler-Independent Module Federations: Enabling Typed Micro Frontend Architectures	Lando and Hasselbring [19]
A6	Assessing the Feasibility of Micro Frontend Architecture in Native Mobile App Development	Capdepon et al. [5]
A7	Challenges, Communication and Future of Micro Frontends Development and Implementation	Stepanov and Klym [39]
A8	A Multi-way Access Portal Website Construction Scheme	Bian et al. [3]
A9	Application Business Information Interaction Bus Based on Micro Frontend Framework	Zhang et al. [45]
A10	Application of LCDP and Technology in Existing Business Systems in Enterprise	Fu et al. [10]
A11	Application of Micro Frontend Technology in Network Public Opinion System	Zhao et al. [46]
A12	The advantages of Micro Frontend architecture for developing web application	Gashi et al. [11]
A13	Micro Frontends as Opportunity for Process Orchestration Layer Architecture in Modular Process Plants	Lorenz et al. [21]
A14	A Progressive Web Application Based on Microservices Combining Geospatial Data and the Internet of Things	Mena et al. [23]
A15	Experiences on a Framework-less Micro Frontend Architecture in a Small Organization	Männistö et al. [26]
A16	An Approach to Follow Microservices Principles in Frontend	Perlin et al. [32]
A17	Framework-Agnostic JavaScript Component Libraries: Benefits, Implementation Strategies and Commercialization Models	Gupta et al. [12]
A18	Micro Frontend Architecture: A Comparative Study of Startups and Large Established Companies-Suitability, Benefits, Challenges and Practical Insights	Sutharsica and Arambepola [40]
A19	Dynamic Micro Frontend Architectures for Enhancing Scalability and Performance in Multi-Tenant Web Application	Kambala [13]
A20	A Catalog of Micro Frontends Anti-Patterns	da Silva et al. [8]
A21	Micro Frontend Architecture for Robotic Systems: a Systematic Approach With Design Rationale	Silva et al. [35]
A22	Frontend Solutions for Micro Applications	Cao et al. [4]
A23	Optimization Scheme of Micro-front-End Architecture Based on Dynamic Cloud Components and Web Components	Tao et al. [41]
A24	Analysis and research on computer website front-end development technology and optimization measures	Chen [7]
A25	Intelligent Enterprise Portal Web Workbench based on Micro frontends	Liu et al. [20]
A26	Motivations, benefits and issues for adopting Micro Frontends: A Multivocal Literature Review	Peltonen et al. [31]
A27	A Novel Application of Educational Management Information System based on Micro Frontends	Wang et al. [43]
A28	Implementing Digital Twins via micro frontends, microservices and web 3D	Simões et al. [36]
A29	Micro Frontend Based Performance Improvement and Prediction for Microservices Using Machine Learning	Kaushik et al. [14]
A30	Evaluating Micro Frontend Approaches for Code Reusability	Stefanovska and Trajkovik [38]

[10]'s study. Kaushik et al. [14] do the same, but also highlight technology agnosticism as core attribute.

Capdepon et al. [5] characterize each module as a distinct, autonomous feature, prioritizing independence and modularity. Notably, among the studies, Capdepon et al. [5] uniquely identify micro frontend as an end-to-end unit encompassing both the frontend and backend—often described as a "vertical slice".

Bian et al. [3] conceptualize MFA as an assembly of discrete functions and modules, emphasizing modularity and development independence. Zhao et al. [46] and Stefanovska and Trajkovik [38] define MFA primarily through its modularity, focusing on the functional decomposition of the system. A distinguishing assertion by these authors is that this approach can lead to a significant increase in "program speed", a claim that shifts the focus from organizational agility to technical performance.

4.1.3 *Answer To RQ1.*

To address RQ1, despite the lack of universal consensus, MFA definitions in the literature focus on four conceptual dimensions (decomposition, assembly, microservice heritage, and full-stack integration), which were synthesized with the core characteristics detailed in Section 4.2 to propose the following unified definition:

> MFA is an architectural style that extends the principles of microservices to the presentation layer (frontend) by breaking down a system into smaller independently developed and deployed frontend modules with heterogeneous technologies, owned by individual teams, which are assembled to form the final application itself.

To contextualize our proposed definition, it is essential to compare it with the conceptualizations presented in related works (Section 2). Moraes et al. [25] represents the only related study that explicitly proposes a standalone definition. Their definition, however, differs from ours in some key aspects. For instance, it omits independent development and heterogeneous technologies from the core definition itself—although the authors do discuss independent development as a mandatory factor, and frame heterogeneous technologies as an optional possibility. Furthermore, Moraes et al. [25] define Micro Frontends as strictly business-driven, whereas our systematic analysis revealed this to be an occasional characteristic rather than a requirement.

In contrast, Peltonen et al. [31] and Santos et al. [34] do not synthesize a concise, single-sentence definition. While Peltonen et al. [31] rely heavily on gray literature (40 out of 43 sources), the current study explicitly consolidates findings grounded in peer-reviewed sources. Meanwhile, Santos et al. [34] do not aim to conceptually define MFA, focusing instead on specific operational facets such as negative impacts, team collaboration, and testing frameworks.

4.2 RQ2: What Are The Essential Characteristics And Principles Of MFA?

This section synthesizes the 13 MFA characteristics identified across 30 studies. Four primary attributes predominate, each appearing in over 60% of the literature, with Modularity cited in more than 95% of the analyzed papers (see Section 4.2.6).

4.2.1 Modularity. As the most prevalent characteristic, **modularity** is cited in 29 of 30 studies, with only A4 [17] omitting it.

Defined as the decomposition of complex systems into autonomous units that collectively form an application [4, 7, 8, 11, 13, 14, 18, 20, 31, 38, 39, 46], modularity serves as a macro-concept. During the re-analysis (Section 3.4), several technically distinct principles were synthesized under this umbrella as complementary facets of the same architectural goal: "low coupling" and "decoupling" [4, 7, 10, 36, 41–43, 45], with the flexibility to follow business-driven boundaries; "autonomous units" and "component independence" [5, 10, 13, 18, 39] to minimize cross-module impacts; and "code isolation" [18, 23] to enhance maintainability.

4.2.2 Independence. As the second most cited characteristic, independence is divided into two complementary dimensions. First, **independent teams / development**, which encompasses "team autonomy" [5, 40] and "independent management" [2], appearing in 26 of 30 articles (except [17, 18, 21, 45]). This dimension grants autonomy over development processes, tech stacks, and structures [3–5, 7, 10–12, 14, 23, 26, 31, 32, 35, 39, 40, 42, 43, 46], isolating internal workflow changes [2, 4, 13, 14, 31, 32, 35, 36, 38, 40]. This characteristic is intrinsically linked to modularity, as development autonomy relies on decomposing complex systems into smaller, cohesive modules [2, 4, 5, 7, 8, 10–14, 19, 20, 23, 31, 32, 35, 39–41].

Second, **independent deployment**, which incorporates "independent releases" [2], was identified in 23 of 30 studies (excluding [14, 17, 18, 21, 23, 42, 46]). It enables teams to deploy modules without cross-team coordination [2, 3, 5, 10–13, 19, 26, 31, 32, 39, 40, 45], facilitating continuous, incremental upgrades without full-system redeploys [4, 7, 8, 13, 20, 31, 35, 36, 38, 40, 41, 43, 45]. Literature frequently links both groups, as architectures facilitating autonomous development typically support individual module deployment.

4.2.3 Microservice Heritage. Characterized in 22 of 30 studies as an **extension of microservice principles** (except [2, 4, 14, 17, 20, 21, 38, 45]), micro frontends share a strong conceptual correlation with backend microservices. This architectural style significantly shaped the foundational principles of MFA [3, 5, 8, 10, 13, 18, 19, 21, 31, 32, 35, 40, 42, 43, 46] through a shared goal: decomposing monoliths into autonomous units [3, 5, 7, 11, 13, 18, 19, 26, 32, 35, 36, 39–41]. Hence, micro frontends often represent the practical implementation of microservices within the presentation layer [3, 10–13, 19, 21, 26, 32, 35, 39, 46]. Although Männistö et al. [26] and Capdepon et al. [5] propose vertical decomposition spanning both environments, the majority of the literature maintains a clear distinction between frontend and backend architectural domains.

4.2.4 Technology Agnosticism. This characteristic, identified in 19 studies (except [3, 5, 11, 19, 21, 36, 38, 39, 45, 46]), subsumes "compatibility" [10], "technological heterogeneity" [32], and "flexibility" [12, 35]. Characterized by enabling the simultaneous use of diverse technologies, this capability is inherently linked to modularity and independence (Sections 4.2.1 and 4.2.2). Decoupled modules empower teams to adopt optimal stacks for targeted business needs and integrate new technologies safely without broader systemic impacts.

4.2.5 Secondary Characteristics. The literature also highlights secondary characteristics: (i) **reduced complexity** [4, 14, 31, 32, 36, 42, 46] through decoupled modules; (ii) **reusability** [4, 7, 17, 18, 23, 32, 40] at code [7, 40] and application [17] levels; (iii) **scalability**, noted as a characteristic [2, 10, 40, 41] and benefit [5, 8, 11, 12, 14, 19, 26, 31, 35, 36, 38, 39, 42, 43, 46] for independent scaling; (iv) **domain-driven** alignment [4, 7, 17, 32, 36, 38] with functional areas; (v) **resilience** [4, 13, 18, 32] against component failures; and (vi) **configurability** [7, 23, 26] for visibility management. Other traits cited without elaboration include **evolvability** [26, 31, 43], **team integration** [7, 18], and **performance optimization** [7].

4.2.6 Answer To RQ2.

> The analysis identified 13 characteristics associated with MFA. While most of these traits were mentioned by fewer than 10 articles, four essential characteristics clearly stand out: **Modularity**, **Independence**, **Microservice heritage**, and **Technological agnosticism**. These four are particularly significant, as each was cited by at least 20 of the 30 studies analyzed, indicating that they represent the core pillars of the architecture.

4.3 RQ3: What Classification Models For MFA Have Been Proposed In The Literature?

This subsection presents the MFA classifications identified during data extraction.

4.3.1 Composition. This dimension concerns the orchestration of individual modules into a unified application. Nine studies [5, 8, 17, 18, 31, 32, 35, 38, 40] outline four methods, all recognizing server- and client-side approaches. Specifically, server-side composition integrates components on the server prior to delivery [8, 32], whereas client-side composition executes assembly within the browser. Alternatively, edge-side composition [8, 18, 31, 32, 38] utilizes intermediate Content Delivery Networks (CDNs) to decrease server overhead. Lastly, hybrid composition [40] blends multiple techniques to maximize flexibility and performance. Although Sutharsica and Arambepola [40] only details server- and client-side blending, the hybrid approach conceptually encompasses any combination of these strategies.

4.3.2 Division. A classification based on the method of distributing modules across pages [2, 11, 17, 31, 32, 36, 38]. This approach follows either a horizontal or vertical logic, distinguished primarily by the number of modules per page [11]. Horizontal architectures integrate multiple modules into a single view [11] and usually necessitates deliberate orchestration to ensure a cohesive UX [31]. Vertical architectures, in contrast, assign an entire page to a single module [11], prioritizing domain-driven logic, organizing modules around distinct business capabilities [31].

4.3.3 Integration. MFAs are also classified by their integration method [8, 13, 21, 38, 39]—either runtime or build-time (compile-time)—which defines how separate modules are made available to the system. Runtime integration maintains modules as independent units that the container application calls and assembles dynamically during execution. In contrast, build-time integration—frequently paired with server-side composition—bundles pre-built modules into the main system before deployment. This results in a single, cohesive deployment unit where modules lack independent availability.

4.3.4 Communication. Few studies classify communication methods within MFAs, causing discrepancies between general discussion and classification citation counts. Literature identifies three primary categories: **API combination** [4], which aggregates data from multiple APIs to reduce redundant requests and optimize performance via encoded URL parameters; **shared state** [4, 20], which ensures components share access to common data (e.g., window or storage objects) to facilitate synchronization despite data conflict risks [4]; and **publish-subscribe** [4, 20], which leverages an event bus for decoupled, anonymous module communication [4].

Additionally, da Silva et al. [8] expand this framework by classifying communication into three distinct directional paths: Parent-Fragment for host notifications, Fragment-Parent for fragment updates, and Fragment-Fragment for direct inter-module interactions.

4.3.5 Answer To RQ3.

The MFA taxonomy comprises four primary classification groups: **composition**, **division**, **integration**, and **communication**. Composition defines how modules are assembled into the final User Interface (UI) through *server-side*, *edge-side*, *client-side*, or *hybrid* methods. Division categorizes architectures as either *horizontal* or *vertical* based on the quantity of modules per page. Integration distinguishes between *build-time* and *runtime* orchestration, while communication analyzes inter-module and server interactions via *API combination*, *shared state*, or *publish-subscribe* patterns.

4.4 RQ4: What Are The Main Challenges Associated With MFA?

This section examines 26 identified MFA challenges. Although presented individually, these issues are often interconnected and frequently co-occur.

4.4.1 Integration And Management Of Application Parts. Orchestrating and managing disparate components is the primary MFA challenge identified in the literature [2, 4, 7, 11, 13, 17, 19, 26, 31, 32, 38, 40–42], encompassing interrelated issues such as "governance" [32], "integration techniques" [2, 17, 19, 26, 31, 40], and "component isolation" [41]. This multifaceted challenge is strictly concerned with the integration, coexistence, and assembly of system fragments within the application. It explicitly excludes system routing, which is addressed as a separate and distinct challenge.

Within this scope, determining optimal module granularity represents a critical challenge, as Gashi et al. [11] note that, absent universal standards, decomposition requires rigorous contextual analysis alongside careful backend integration. On an organizational level, this complexity is heightened by governance hurdles; Perlin et al. [32] emphasize that internal pressures often favor tightly coupled components, complicating governance by necessitating intensive coordination between independent teams. These structural and technical friction points explain why integration is so widely cited as a critical MFA hurdle [2, 17, 19, 26, 31, 40]. Indeed, a survey of 200 professionals conducted by Sutharsica and Arambepola [40] highlights the scale of this problem, reporting integration issues in 64.10% of startups and 70.83% of enterprises due to the difficulties of orchestrating disparate subprojects into a cohesive whole [2, 18, 19, 26, 31].

From an operational perspective, MFA decentralization inherently increases management overhead by requiring the coordination of concurrent modules [13]. This overhead typically peaks during release cycles, where managing independent modules demands rigorous operational and resource coordination [7]. Furthermore, the ecosystem requires multiple servers, independent pipelines, and specialized monitoring [38, 42], meaning that scaling Continuous Integration and Continuous Deployment (CI/CD) processes significantly escalates overall system complexity [38].

Architecturally, designing decoupled, business-driven components represents another critical hurdle, with Cao et al. [4] highlighting the difficulty of reasonably splitting the business domain to achieve weakly coupled modules. These logical boundaries are further complicated by component-level isolation and management issues [41]. To mitigate these persistent practical challenges, Tao et al. [41] propose a cloud-managed architecture to centralize module registration, versioning, and configuration, which reportedly enhances both performance and architectural flexibility. Ultimately, exploring these diverse dimensions reveals that significant challenges still persist regarding the practical integration, orchestration, and management of decentralized application parts.

4.4.2 High System Complexity. Evidence across multiple studies within the final selection suggests that MFA systems present significantly higher complexity than traditional patterns [2, 7, 12, 13, 26, 31, 32, 36, 38–40, 42]. While related to the previous challenge, this **high system complexity** encompasses the broader operational and technical burden of a decentralized ecosystem, manifesting across several distinct dimensions. Specifically, it extends beyond interconnectivity logic to include both the initial architectural setup and the sustained resources required for long-term maintenance—conceptual distinctions that, despite their differences, frequently overlap in practice.

This inherent escalation of system complexity [7, 12, 13, 26, 31, 38–40, 42] is largely attributed to system decentralization, which necessitates scaling CI/CD pipelines to manage numerous independent modules [38], making this complexity a primary architectural drawback for organizations to mitigate. Similarly, literature identifies a marked escalation in operational complexity, with Perlin et al. [32] characterizing it as the architecture's central challenge due to the inherently distributed nature of MFAs. Furthermore, while initial configuration [2, 38] and technical implementation [7] are recognized hurdles, their specific impediments remain largely unexplored, whereas the sustained effort required for ongoing development and maintenance is identified as one of the architecture's most significant challenges [36]. Ultimately, this complexity dimension reveals a fundamental architectural trade-off: while modularity provides significant advantages, it also imposes heightened complexity and integration hurdles across the system.

4.4.3 System Communication. Inter-module communication is a primary MFA challenge [2, 4, 7, 11–14, 31, 36, 38, 39, 41]. Despite the emphasis on isolation, modules often require shared state or synchronized updates; consequently, data changes in one fragment must trigger cascading updates to maintain system-wide consistency. This multifaceted issue encompasses several distinct dimensions.

One prominent dimension is the regulation of inter-module communication [4, 7, 38, 39]. Because unconstrained interactions can compromise system stability [39], treating modules as independent sub-applications requires rigorous architectural oversight to ensure communication remains performant and ergonomic [4, 7, 38]. A closely related dimension involves the standardization of communication protocols [11], where technological heterogeneity complicates the implementation of uniform patterns across diverse, independent modules. Beyond these structural and governance challenges, the literature highlights runtime anomalies as a distinct, critical dimension of this problem. For instance, infinite loop dispatch [2] involves circular update triggers—primarily affecting pub-sub models—where a module's state update cascades back to its origin, triggering cyclic dependencies that lead to performance degradation and resource exhaustion.

Beyond these runtime and stability anomalies, communication overhead represents another critical challenge [12, 31, 36], often arising from suboptimal design or architectural and organizational misalignment, which requires rigorous planning and contextual analysis to mitigate. While some studies [13, 14] identify communication as a primary hurdle in abstract terms without detailing practical implementation difficulties, specific technical choices highlight these pain points; for instance, although iframes provide robust isolation, they frequently introduce significant latency and reduce overall efficiency [41]. Ultimately, this analysis reveals that communication remains a major challenge in MFA, particularly concerning inter-module interaction, standardization, and operational effectiveness.

4.4.4 Performance. Cited in eleven analyzed studies [2, 7, 11, 12, 19, 26, 31, 36, 39–41], performance challenges encompass various technical hurdles and architectural consequences. MFA loading issues often arise from the modular assembly process, where incremental loading can diminish perceived system reliability [11]. In this context, maintaining a seamless UI remains a significant challenge, especially in dynamic environments where fragmented rendering disrupts the user journey [26]. To mitigate this, lazy loading is frequently used to optimize responsiveness and initial load times by prioritizing critical path components [2], although implementing this mechanism is itself acknowledged as a challenge in MFA [2].

Although identified as a critical hurdle [11, 19], this optimization remains underexplored regarding its specific origins and potential solutions, with such overhead being a recognized consequence of the architecture's inherent modularity [19, 40, 41]. Literature frequently reports these concerns in generic terms [12, 31, 36, 39]; while Stepanov and Klym [39] attribute this to tight module coupling, others cite inadequate vendor consideration [31, 36] or acknowledge the issue without further elaboration [12]. Lastly, although performance bottlenecks are recognized as important challenges for this architectural style, existing work [7] acknowledges the issue without exploring its underlying causes or potential solutions. Each performance challenge identified here warrants further investigation to optimize MFA and ensure that end-user requirements are met.

4.4.5 Dependency Management. Ten primary studies [2, 5, 11, 13, 14, 18, 19, 26, 31, 40] identify dependency management as a multifaceted challenge, encompassing version control and the governance of external dependencies across independent modules. The scope of this challenge is confined exclusively to version control, package dependency management across modules, and ensuring that interdependent system modules maintain operational integrity. It excludes dynamic data management, which is addressed under system communication challenges (Section 4.4.3), as well as payload inflation caused by duplicated dependencies, which is detailed in Section 4.4.7 as the increased payload size challenge.

Within this scope, version control represents a significant hurdle in MFAs [18, 19]; without rigorous planning, managing concurrent module versions can become unpredictable, potentially introducing regressive or unexpected system behaviors. Alongside this, external dependency management is also identified as a notable challenge, particularly in native mobile micro frontends, as highlighted by Capdepon et al. [5]. Although their study evaluates the feasibility and mobile-specific hurdles of this architecture, it acknowledges the difficulty of managing these external dependencies without further investigating causes or mitigation strategies.

Additionally, managing changes within individual dependencies is a critical challenge in MFAs [11], although various established web development techniques can be leveraged to address the inherent versioning and compatibility issues. In parallel, managing shared resources and dependencies is recognized as a significant hurdle, particularly for large-scale applications [2, 26], though both studies briefly address this challenge without further elaborating on the specific technical complexities or solutions. Alongside these concerns, dependency governance also emerges as a primary implementation hurdle [19, 40], with Lando and Hasselbring [19] specifying that seamless integration necessitates sophisticated tracing tools to avoid introducing inadvertent tight coupling.

Beyond the issues discussed above, other challenges within this domain have been identified in the literature. For instance, dependency synchronization is a noted difficulty, particularly when concurrent interactions between different micro frontend versions result in user interface inconsistencies [13]. Additionally, the modular nature of micro frontends often leads to redundant core packages, such as React, being bundled independently across modules [31]. Within this scope, this redundancy heightens the complexity of coordinating and aligning package versions across subprojects; its secondary consequence on application payload inflation is addressed separately in Section 4.4.7. Finally, Kaushik et al. [14] briefly address code divergence as another distinct issue related to dependency management, though without further elaboration. Considering that, proper dependency management within applications utilizing MFA is identified as a critical architectural challenge, requiring rigorous attention in future research;

4.4.6 UX Consistency. Unlike multifaceted challenges, this prominent issue—identified in ten studies [11–13, 18, 19, 31, 32, 36, 40, 42]—is treated as a singular, cohesive challenge in the literature. It exhibits high consensus across sources, without being divided into distinct sub-facets or internal categories.

Sustaining UI cohesion is a pivotal MFA challenge, requiring autonomous teams to deliver integrated modules [31, 40, 42]. Fragmented UX typically stems from decentralization, weak governance, or unstandardized styling, necessitating rigorous coordination during scaling [11, 12, 19, 31, 32, 36, 40], while versioning mismatches

and dependency conflicts frequently compound these consistency issues [13]. Although shared design systems are often employed to enforce aesthetic harmony [18], they introduce a significant architectural trade-off: the required centralized governance can directly undermine the decoupling and team autonomy central to micro frontend objectives [32, 36].

4.4.7 Secondary Challenges. In addition to the most frequently cited challenges, several less prominent ones also emerged. Although these were not categorized during the initial coding phase or subsequent analysis, they have been organized into thematic groups to maintain structural coherence and narrative flow.

Governance & Organizational Dynamics. **Team-related issues** [7, 18, 19, 31, 36, 40] address organizational factors rather than technical execution. Within this domain, effective team coordination is important to maintaining visual and system-wide consistency [7, 18, 19], a challenge typically mitigated through synchronization meetings. Furthermore, a steep learning curve and a lack of specialized expertise [36, 40] exacerbate technical failures and UI discrepancies. This organizational friction is often mirrored in the software architecture; for instance, inter-team coupling usually stems from insufficient architectural decoupling [31], while isolated codebase management fosters islands of knowledge and fragmentation [31]. In addition to it, high team autonomy introduces the risk of redundant implementation of overlapping business logic [31]. Closely tied to these dynamics are repository organization complexities [2], which arise when defining codebase structures that balance modular autonomy with shared repositories to guarantee UX consistency. Lastly, though less elaborated in the literature, achieving a consistent development experience [19] remains a challenge in MFA environments.

Software Architecture & Code Cohesion. Regarding architectural and code cohesion challenges, **code reusability** remains a problem [8, 13, 14, 19, 31, 36, 38, 42], particularly concerning component management and the prevalence of redundancy. The difficulty in developing fully agnostic components frequently induces duplication, thereby rendering several modules partially redundant. Parallel to code structure, **data management** issues [2, 8, 13, 39, 41] manifest through state management complexities and global variable pollution. The former concerns preserving data model integrity within modular environments, whereas the latter stems from excessive global variable instantiation for state handling, an issue often inherent to MFA contexts. This architectural style also complicates **type safety** [19], where a lack of synchronization between exported module interfaces and consumers can precipitate critical runtime failures. Furthermore, **configurability** difficulties [41] frequently co-occur with challenges in component isolation and cross-frame compatibility. Lastly, structural decentralization introduces difficulties in both **system routing** [2, 13, 38], owing to complex route-to-module mappings, and **static asset services** [2], which requires the orchestration of shared resources (e.g., imagery and typography) across distributed MFA ecosystems.

Infrastructure, Performance & Lifecycle. In this context, **CI/CD** [7, 17, 38] is hindered by insufficient support for rapid component evolution [17], which necessitates specialized workflows [7]. **Debugging and monitoring** [8, 19, 31] is linked to the modular nature of micro frontends; in runtime-integrated modules, debugging each module while considering the whole application context is difficult to execute [19]. Regarding **resource usage** [7, 31, 40], this refers to the resources necessary to maintain and scale systems using MFA. Because specific servers are required to provide each module individually [7, 31], this factor tends to be a crucial issue for small companies [40]. Additionally, **increased payload size** [8, 31, 36] stems from suboptimal dependency management, which leads to excessive asset downloads [31]. Lastly, **testing** [32] is noted in the studies, albeit briefly.

Platform-Specific & Cross-cutting Concerns. **Native mobile environments** [5] present several challenges. First, technological agnosticism is unfeasible because these platforms restrict mixing disparate languages (e.g., Java/Kotlin or Objective-C/Swift) [5]. Second, independent deployment is hindered by monolithic build processes; despite development autonomy, app store requirements necessitate full system rebuilds [5]. Additionally, intermittent connectivity complicates implementations that rely on remote resources. Finally, Mobile Platform Restrictions further constrain MFA adoption in native contexts. In addition, **compatibility** issues [13, 31, 41] involve legacy system integration and CDN language constraints, particularly during multi-vendor migrations or architectural transitions [31]. Regarding **system security** [7, 11, 18], concerns stem from architectural modularity, which requires systematic audits and rigorous protocols to mitigate risks from cross-domain resource sharing. Lastly, the remaining challenges were only briefly mentioned in the studies without further elaboration: **search engine optimization** [13], **accessibility** [31], and **cross-cutting issues** [36].

4.4.8 Answer To RQ4.

A total of 26 challenges associated with MFA adoption were identified throughout the literature analysis. Among these, Six primary challenges emerged as the most significant, each cited by at least 10 of the 30 analyzed studies, representing a substantial level of concern within the research community compared to the remaining 20 issues. These core challenges are: **integration and management of application parts**, **high system complexity**, **system communication**, **performance**, **dependency management**, and **UX consistency**.

4.5 RQ5: What Implementation Approaches For MFA Are Most Frequently Reported In The Literature?

Thirteen implementation approaches were identified across 29 articles, with A1 [42] as the sole exception. These patterns are not mutually exclusive, permitting hybrid implementations.

4.5.1 Web Components. This **widely adopted approach** [2, 4, 7, 8, 11–13, 17, 18, 21, 26, 35, 36, 38, 41, 43, 46] employs standardized APIs to develop reusable custom elements [22, 29].

Web Components facilitate **code reusability** via modular integration across diverse applications [11, 22, 38, 43, 46]. The suite ensures encapsulation, isolating elements and logic from the host environment [17]. Implementation relies on three pillars: **custom**

elements, **shadow DOM** (Document Object Model), and **HTML templates** [17, 22, 43, 46].

Custom elements leverage APIs to define custom tags and behaviors by extending native HTML elements [17, 22]. Registered tags subsequently function as standard HTML elements within the interface [22].

Shadow DOM attaches a discrete shadow tree to an element, isolating internal logic and structure from the primary DOM [22]. This mechanism enforces encapsulation by maintaining private internal scopes [17].

HTML templates provide inert markup fragments as reusable blueprints for custom elements [22]. These segments facilitate programmatic cloning and document insertion [17].

Web components serve as an efficient foundation for MFAs [26], particularly because they enable the integration of applications developed across heterogeneous frameworks [38].

4.5.2 Iframes. As a widely adopted approach [3, 4, 8, 11, 13, 17, 18, 20, 21, 23, 31, 35, 38, 39, 41, 43, 46], the use of inline frames (iframes) establishes a nested browsing context for external documents, thereby maintaining independence between the host and sub-applications [3, 11, 46]. This architectural isolation prevents embedded content from affecting the primary document [46] while facilitating integration with low coordination overhead through complete host-content decoupling [3]. Ultimately, the core advantage of iframes is their robust modular isolation, which is a critical requirement for MFA architectures [11, 43].

As a foundational MFA method, iframe implementation necessitates decomposing applications into discrete modules, each loaded independently into the host [38, 39], which facilitates rapid module integration [3]. Within this setup, host-iframe communication typically utilizes `postMessage` events [3]. However, despite its maturity, this communication method incurs performance overhead and latency [13, 41], alongside constraints regarding window protocols or cookie management [3]—demonstrating that, although consolidated, iframes retain notable implementation challenges.

4.5.3 Module Federation. As a widely recognized micro frontend integration approach [2, 4, 7, 8, 11–13, 17–19, 32, 36, 38–40], Module Federation relies on a central host that manages the runtime environment to dynamically load local and remote modules [2, 19]. This model allows for efficient library version management—through either sharing or isolation—while ensuring each module functions as an autonomous application [2, 18, 19].

Module Federation enables micro frontends to expose and consume shared components without build-time bundling [19]. The key benefits include **autonomous** deployment and runtime isolation [19]. Additionally, on-demand retrieval enhances architectural **adaptability** [2, 39]

While originally tied to Webpack 5 [2], Module Federation v2 is bundle-agnostic, enabling the use of heterogeneous build tools within a single MFA [19]. Furthermore, Lando and Hasselbring [19] emphasize that beyond this build independence, the updated version introduces support for WebAssembly integration and enhanced type safety.

4.5.4 Single-SPA. This framework orchestrates JavaScript micro frontends within a central host application [2, 4, 8, 10, 11, 13, 37, 40, 41, 43, 46], enabling heterogeneous technology stacks while maintaining sub-application independence [11, 41]. Frameworks like Single-SPA reduce technical complexity [10, 40] but necessitate strict adherence to specific architectural patterns [10].

Module loading follows a three-stage lifecycle: bootstrap initializes the module, mount renders it for presentation, and unmount removes it when no longer required [4].

4.5.5 Route Distribution. Loading pages from specific routes [2–5, 7, 8, 14, 17, 31, 43] defines route distribution, a method intrinsically linked to vertical split architectures (see Section 4.3.2).

The host system orchestrates the architecture by mapping modules to a global routing schema and specific screen positions [17], typically using reverse proxies or framework-specific routing.

Transclusion integrates MFA routing and composition by interpreting code based on the active route [5], allowing the host to validate URLs and dynamically render matching modules [3, 4, 8].

4.5.6 Other Implementations. Other less common methods include the **application shell** [2, 8, 17, 26, 31, 35], which uses an orchestrator and import-maps to manage module registries and lifecycles [2, 35]. Similarly, **Qiankun** [4, 7, 8, 10, 45, 46] decouples development by partitioning applications into a main program and submodules, though it requires the Universal Module Definition (UMD) format [2, 46]. In contrast, **build-time integration** [8, 13, 21, 38, 39] treats modules as standard package dependencies, compiling and bundling them together prior to deployment [38]. While critics argue that this approach compromises core MFA objectives—such as independent deployability—proponents recognize it as a pragmatically valid implementation strategy.

Self-Contained Systems (SCS) [5, 11, 26, 43] treat modules as independent vertical systems with dedicated backend and frontend logic, rather than mere presentation layers [11]. In contrast, **Wujie** [4, 10, 45] utilizes iframes and Web Components to provide native CSS/JS isolation and resource preloading [4], though it faces challenges regarding window object reliance and performance [4].

Other noted approaches include **SystemJS** [2, 45], **Package Business Capacities** (PBCs) [36], and **Garfish** [8].

4.5.7 Answer To RQ5.

> A total of 13 implementation approaches emerged from the literature. Five primary methods—**Web Components**, **iframes**, **Module Federation**, **Single-SPA** and **route distribution**—were cited by at least ten articles. Therefore, it is possible to infer that these five are the most acknowledged strategies for implementing MFA.

4.6 RQ6: What Communication Mechanisms Are Most Commonly Adopted In MFA?

Among the 30 reviewed articles, 22 specifically address system communication, from which six distinct approaches were identified. These approaches are frequently integrated into hybrid topologies to satisfy specific architectural requirements.

4.6.1 Event Emitter. This mechanism is the most cited communication pattern [2–4, 7, 8, 11, 13, 17, 20, 23, 31, 35, 36, 38, 39, 41, 43], alternatively termed *parent-child*, *custom events*, *event-driven* or *pub-sub*.

By utilizing a shared event bus and the custom events API [4, 11, 13, 39], event-driven communication decouples micro frontend modules [4]. This strategy eliminates direct dependencies, providing the state synchronization and flexibility required to uphold core architectural principles.

4.6.2 Other Communication Methods. Beyond event emitters, **API-based communication** [4, 7, 11, 13, 14, 18, 21, 26, 45] is a prevalent MFA strategy that aggregates data from multiple APIs to meet system requirements [13]. This approach typically utilizes centralized gateways to manage information exchange between modules and backend services [14].

Web storage [4, 13, 17, 31, 36, 38, 39] leverages native browser storage to share data across modules, effectively abstracting storage APIs in cross-platform environments like React Native [39]. Similarly, **global state** [2, 4, 7, 20, 39, 41] employs centralized repositories as a single source of truth for module synchronization [4], though it necessitates rigorous management.

Query parameters [13, 31, 36, 38] encode state information directly into the URL string, providing a lightweight method for transmitting data across modules [13, 31]. Similarly, **URL changes** [31, 36] utilize route transitions to trigger new views and server-side data fetches to update micro frontend state [36].

4.6.3 Answer To RQ6.

Six methods for inter-module and client-server communication were identified: **event emitter**, **API-based**, **web storage**, **global state**, **query parameters**, and **URL changes**. Among these, the **event emitter** is the prevailing mechanism, cited in 17 articles, while all other methods appeared in fewer than 10.

5 Threats To Validity

Primary internal validity threats involve data extraction bias—mitigated by an audit conducted by the second author—and the use of binary extraction. To maintain analytical rigor, the extraction process strictly captured core contributions and considered the textual context rather than superficial mentions. Regarding external validity, potential article omission via database selection was mitigated by targeting high-impact software engineering venues, while the absence of a snowballing phase is justified by the attainment of conceptual saturation (Section 3).

6 Conclusions

This research synthesizes the MFA landscape, identifying a lack of universal definition despite consensus on four pillars (RQ2): **Modularity**, **Independence**, **microservices based**, and **Technological Agnosticism**. Nine secondary characteristics, including scalability and resilience, were also identified but appear less frequently in the literature.

The architectural landscape is structured across four taxonomic axes (RQ3): **Composition** (Server-side, Edge-side, Client-side and Hybrid); **Division** (Horizontal and Vertical); **Integration** (Runtime and Build-time); **Communication** (Publish-Subscribe, Shared State and API Combination).

The primary MFA adoption challenges (RQ4) include **modules integration**, **system complexity**, and **inter-module communication**, alongside **performance**, **dependency management**, and **UX consistency**. While Section 4 lists twenty additional issues, these **six core challenges** remain the most significant barriers to architectural maturity.

Implementation (RQ5) is led by **Web Components**, **Iframes**, **Module Federation**, **Single-SPA**, and **Route Distribution**, with eight secondary methods — including *Qiankun* and *SystemJS* — also identified.

The **Event Emitter** (or publish-subscribe) mechanism emerged as the most cited communication method (RQ6), significantly outperforming five secondary methods such as Web Storage and Query Parameters.

6.1 Future Work

Based on the gaps and limitations identified in this mapping, we outline a concrete research agenda to guide future work through specific study types:

- **Empirical Validation Studies:** The proposed four-axis taxonomy, while inductively derived from the literature, must be validated against real-world MFA projects and through practitioner surveys to confirm its practical completeness and further map architectural trade-offs.
- **Controlled Experiments:** Rigorous quantitative evaluations are required to address performance concerns among competing implementation methods, such as controlled experiments comparing *Module Federation* versus *Web Components* regarding loading overhead and runtime execution.
- **Case Studies and Action Research:** Future empirical efforts should investigate how industry teams mitigate the identified core challenges in large-scale, long-lived production systems to provide prescriptive engineering guidelines.

The ultimate goal is to transition the MFA domain from descriptive cataloging toward prescriptive frameworks that balance architectural modularity with a seamless, high-performance user experience.

Artifact Availability

To ensure transparency and reproducibility, the replication package for this SMS is publicly available on Zenodo at https://doi.org/10.5281/zenodo.21412837 [6]. This specific repository version contains the selected primary sources, the quality assessment matrix, raw extracted definitions, and the complete qualitative codebook (189 initial codes refined into 83 consolidated categories). All materials are distributed under the terms of the Creative Commons Attribution 4.0 International (CC-BY-4.0) license.

Acknowledgements

This study was financed in part by CNPq (Grant nº 406941/2025-4).

References

[1] Giovanni Amorim, Larissa Rocha, Fabiana Mendes, Rodrigo Santos, and Edna Canedo. 2025. Guidelines for Adoption Micro-frontend Architecture. In *Anais do XXI Simpósio Brasileiro de Sistemas de Informação*. SBC, Porto Alegre, RS, Brasil, 713–722. doi:10.5753/sbsi.2025.246619

[2] Marcos Roberto G. Bahiense-Junior, Lanier M. Santos, Renan R. Bezerra, and Erick C. Bezerra. 2024. Micro Frontend Application Shell and Module Federation Architecture Implementation and Comparison. In *2024 IEEE International Conference on Data and Software Engineering (ICoDSE)*. IEEE, Piscataway, NJ, USA, 166–171. doi:10.1109/ICoDSE63307.2024.10829917

[3] Yan Bian, Dechao Ma, Qing Zou, and Weirui Yue. 2022. A Multi-way Access Portal Website Construction Scheme. In *2022 5th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE, Piscataway, NJ, USA, 589–592. doi:10.1109/ICAIBD55127.2022.9820236

[4] Xia-Yu Cao, Rui Kang, Qi Zhang, Hua-Feng Zhang, and Ning-Yuan Li. 2024. Frontend Solutions for Micro Applications. In *2024 5th International Conference on Mechatronics Technology and Intelligent Manufacturing (ICMTIM)*. IEEE, Piscataway, NJ, USA, 729–736. doi:10.1109/ICMTIM62047.2024.10629547

[5] Quentin Capdepon, Nicolas Hlad, Benoit Verhaeghe, and Abdelhak-Djamel Seriai. 2024. Assessing the feasibility of Micro frontend architecture in native mobile app development. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 2309–2313. doi:10.1145/3691620.3695313

[6] João Luiz de Castro and Daniel Lucredio. 2026. *Replication Package for Paper: Micro Frontends: Definition, Classifications, Challenges, Implementation Approaches and Communication Methods*. doi:10.5281/zenodo.21412837

[7] Weiwei Chen. 2025. Analysis and research on computer website front-end development technology and optimization measures. In *Proceedings of the 2025 International Conference on Digital Management and Information Technology (DMIT '25)*. Association for Computing Machinery, New York, NY, USA, 561–566. doi:10.1145/3736426.3736512

[8] Nabson Paiva Souza da Silva, Eriky Rodrigues, and Tayana Conte. 2025. A Catalog of Micro Frontends Anti-Patterns. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE, Piscataway, NJ, USA, 2151–2162. doi:10.1109/ICSE55347.2025.00079

[9] Sandra Fabbri, Cleiton Silva, Elis Hernandes, Fábio Octaviano, André Di Thommazo, and Anderson Belgamo. 2016. Improvements in the StArt tool to better support the systematic review process. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (Limerick, Ireland) *(EASE '16)*. Association for Computing Machinery, New York, NY, USA, Article 21, 5 pages. doi:10.1145/2915970.2916013

[10] Liqin Fu, Feng Qiao, Qin Lin, Hanling Yang, An Liu, Xian Liu, and Hong Li. 2024. Application of LCDP and Technology in Existing Business Systems in Enterprise. In *2024 9th International Conference on Communication, Image and Signal Processing (CCISP)*. IEEE, Piscataway, NJ, USA, 145–154. doi:10.1109/CCISP63826.2024.10765540

[11] Elvisa Gashi, Dhuratë Hyseni, Isak Shabani, and Betim Çiço. 2024. The advantages of Micro-Frontend architecture for developing web application. In *2024 13th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, Piscataway, NJ, USA, 1–5. doi:10.1109/MECO62516.2024.10577836

[12] Kishan Kesari Gupta, Prashant Awasthi, Mahaboobsubani Shaik, and Parag Ravikant Kaveri. 2024. Framework-Agnostic JavaScript Component Libraries: Benefits, Implementation Strategies, and Commercialization Models. In *2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, Piscataway, NJ, USA, 1441–1446. doi:10.1109/CICN63059.2024.10847515

[13] Gireesh Kambala. 2025. Dynamic Micro-Frontend Architectures for Enhancing Scalability and Performance in Multi-Tenant Web Application. In *2025 8th International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, Piscataway, NJ, USA, 633–639. doi:10.1109/ICOEI65986.2025.11013353

[14] Neha Kaushik, Harish Kumar, and Vinay Raj. 2024. Micro Frontend Based Performance Improvement and Prediction for Microservices Using Machine Learning. *Journal of Grid Computing* 22, 1 (2024), 44. doi:10.1007/s10723-024-09760-8

[15] Neha Kaushik, Harish Kumar, and Vinay Raj. 2025. Maintainability improvement of microservices based applications using micro frontend. *International Journal of Computers and Applications* 47, 2 (2025), 188–196. doi:10.1080/1206212X.2025.2450250

[16] Barbara Kitchenham. 2004. *Procedures for Performing Systematic Reviews*. Joint Technical Report TR/SE-0401 and 0400011T.1. Keele University and National ICT Australia Ltd., Keele, UK and NSW, Australia. ISSN: 1353-7776.

[17] Jelena Kičić, Miloš Ljubojević, and Mihajlo Savić. 2024. Dynamic Micro-Frontends. In *2024 11th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN)*. IEEE, Piscataway, NJ, USA, 1–5. doi:10.1109/IcETRAN62308.2024.10645144

[18] Vladimír Kunštnár and Patrik Podhorský. 2024. Micro Frontend Architecture. In *2024 Zooming Innovation in Consumer Technologies Conference (ZINC)*. IEEE, Piscataway, NJ, USA, 124–129. doi:10.1109/ZINC61849.2024.10579400

[19] Billy Lando and Wilhelm Hasselbring. 2025. Toward Bundler-Independent Module Federations: Enabling Typed Micro-Frontend Architectures . In *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*. IEEE Computer Society, Los Alamitos, CA, USA, 36–40. doi:10.1109/ICSA-C65153.2025.00013

[20] Yitian Liu, Wei Guo, and Qingqiang Meng. 2024. Intelligent Enterprise Portal Web Workbench based on Micro-frontends. In *Proceedings of the 2024 16th International Conference on Computer Modeling and Simulation (ICCMS '24)*. Association for Computing Machinery, New York, NY, USA, 142–146. doi:10.1145/3686812.3686834

[21] Julius Lorenz, Candy Lohse, and Leon Urbas. 2021. Micro Frontends as Opportunity for Process Orchestration Layer Architecture in Modular Process Plants. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, Piscataway, NJ, USA, 01–04. doi:10.1109/ETFA45728.2021.9613474

[22] MDN Web Docs. 2026. Web Components. https://developer.mozilla.org/en-US/docs/Web/API/Web_components. Accessed: 2026 feb 03.

[23] Manel Mena, Antonio Corral, Luis Iribarne, and Javier Criado. 2019. A Progressive Web Application Based on Microservices Combining Geospatial Data and the Internet of Things. *IEEE Access* 7 (2019), 104577–104590. doi:10.1109/ACCESS.2019.2932196

[24] Fernando Moraes and Frank Affonso. 2024. A New Integration Approach to support the Development of Build-time Micro Frontend Architecture Applications. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software*. SBC, Porto Alegre, RS, Brasil, 637–643. doi:10.5753/sbes.2024.3585

[25] Fernando Moraes, Gabriel Nagassaki Campos, Nathalia Almeida, and Frank Affonso. 2024. Micro Frontend-Based Development: Concepts, Motivations, Implementation Principles, and an Experience Report. In *Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 2: ICEIS*. INSTICC, SciTePress, Setúbal, Portugal, 175–184. doi:10.5220/0012627300003690

[26] Jouni Männistö, Antti-Pekka Tuovinen, and Mikko Raatikainen. 2023. Experiences on a Frameworkless Micro-Frontend Architecture in a Small Organization. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, Piscataway, NJ, USA, 61–67. doi:10.1109/ICSA-C57050.2023.00025

[27] Charlles Paiva Nascimento and Eder Carlos Salazar Sotto. 2020. Micro Frontend: um estudo sobre o conceito e aplicação no frontend. *Revista Interface Tecnológica* 17, 1 (ago. 2020), 153–165. doi:10.31510/infa.v17i1.798

[28] Olena Nikulina and Kyrylo Khatsko. 2023. Method of Converting the Monolithic Architecture of a Frontend Application to micro frontends. *Bulletin of National Technical University "KhPI". Series: System Analysis, Control and Information Technologies* 2, 2 (10) (Dec. 2023), 79–84. doi:10.20998/2079-0023.2023.02.12

[29] Yuma Nishizu and Tetsuo Kamina. 2022. Implementing Micro Frontends Using Signal-based Web Components. *Journal of Information Processing* 30 (2022), 505–512. doi:10.2197/ipsjjip.30.505

[30] Andrey Pavlenko, Nursultan Askarbekuly, Swati Megha, and Manuel Mazzar. 2020. Micro-frontends: application of microservices to web front-ends. *Journal of Internet Service and Information Security* 10 (2020), 49–66. https://api.semanticscholar.org/CorpusID:220099093

[31] Severi Peltonen, Luca Mezzalira, and Davide Taibi. 2021. Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review. *Information and Software Technology* 136 (2021). doi:10.1016/j.infsof.2021.106571

[32] Rodrigo Perlin, Denilson Ebling, Vinícius Maran, Glenio Descovi, and Alencar Machado. 2023. An Approach to Follow Microservices Principles in Frontend. In *2023 IEEE 17th International Conference on Application of Information and Communication Technologies (AICT)*. IEEE, Piscataway, NJ, USA, 1–6. doi:10.1109/AICT59525.2023.10313208

[33] Johnny Saldaña. 2013. *The Coding Manual for Qualitative Researchers* (2nd ed.). SAGE Publications, Los Angeles.

[34] Luiz Santos, Marcus Silva, Shexmo Santos, Fábio Rocha, and Elisrenan Barbosa da Silva. 2024. Micro Frontend: Systematic Mapping. In *Proceedings of the 20th International Conference on Web Information Systems and Technologies - WEBIST*. INSTICC, SciTePress, Setúbal, Portugal, 119–130. doi:10.5220/0013015400003825

[35] Uanderson Silva, Matheus Andrade, José Cruz, Andresa Silva, Augusto Sampaio, and Kiev Gama. 2025. Micro Frontend Architecture for Robotic Systems: a Systematic Approach With Design Rationale. In *2025 IEEE/ACM 7th International Workshop on Robotics Software Engineering (RoSE)*. IEEE, Piscataway, NJ, USA, 1–8. doi:10.1109/RoSE66716.2025.00005

[36] Bruno Simões, María del Puy Carretero, Jorge Martínez, Sebastián Muñoz, and Nieves Alcain. 2024. Implementing Digital Twins via micro-frontends, microservices, and web 3D. *Computers & Graphics* 121 (2024), 103946. doi:10.1016/j.cag.2024.103946

[37] single-SPA contributors. 2026. Getting Started - Overview | single-spa. https://single-spa.js.org/docs/getting-started-overview Accessed: 2026 feb 23.

[38] Emilija Stefanovska and Vladimir Trajkovik. 2022. Evaluating Micro Frontend Approaches for Code Reusability. In *ICT Innovations 2022. Reshaping the Future Towards a New Normal*, Katerina Zdravkova and Lasko Basnarkov (Eds.). Springer Nature, Cham, Switzerland, 93–106. doi:10.1007/978-3-031-22792-9_8

[39] Oleksandr Stepanov and Halyna Klym. 2024. Challenges, Communication and Future of Micro Frontends Development and Implementation. In *2024 14th International Conference on Dependable Systems, Services and Technologies (DESSERT)*. IEEE, Piscataway, NJ, USA, 1–5. doi:10.1109/DESSERT65323.2024.11122150

[40] Anat Sutharsica and Nimasha Arambepola. 2025. Micro-Frontend Architecture: A Comparative Study of Startups and Large Established Companies-Suitability, Benefits, Challenges, and Practical Insights. In *2025 International Research Conference on Smart Computing and Systems Engineering (SCSE)*. IEEE, Piscataway, NJ, USA, 1–6. doi:10.1109/SCSE65633.2025.11030972

[41] Sunlong Tao, Zhengtian Cui, Yanxin Li, and Qiaoliang Yang. 2024. Optimization Scheme of Micro-front-End Architecture Based on Dynamic Cloud

Components and Web Components. In *2024 10th International Conference on Computer and Communications (ICCC)*. IEEE, Piscataway, NJ, USA, 533–537. doi:10.1109/ICCC62609.2024.10942252

[42] P. Yedhu Tilak, Vaibhav Yadav, Shah Dhruv Dharmendra, and Narasimha Bolloju. 2020. A platform for enhancing application developer productivity using microservices and micro-frontends. In *2020 IEEE-HYDCON*. IEEE, Piscataway, NJ, USA, 1–4. doi:10.1109/HYDCON48903.2020.9242913

[43] Daojiang Wang, DongMing Yang, Huan Zhou, Ye Wang, Daocheng Hong, Qiwen Dong, and Shubing Song. 2020. A Novel Application of Educational Management Information System based on Micro Frontends. *Procedia Computer Science* 176 (2020), 1567–1576. doi:10.1016/j.procs.2020.09.168

[44] Caifang Yang, Chuanchang Liu, and Zhiyuan Su. 2019. Research and Application of Micro Frontends. *IOP Conference Series: Materials Science and Engineering* 490, 6 (04 2019), 062–082. doi:10.1088/1757-899X/490/6/062082

[45] Chuanyu Zhang, Dongxin Zhang, Hualiang Zhou, Xuan Wang, Liang Lv, Rui Zhang, Yehua Xie, Hongkun Liu, Yuanyuan Li, Deshun Jia, Yi Huang, and Tao Jiang. 2023. Application Business Information Interaction Bus Based on Micro Frontend Framework. In *2023 7th International Symposium on Computer Science and Intelligent Control (ISCSIC)*. IEEE, Piscataway, NJ, USA, 306–310. doi:10.1109/ISCSIC60498.2023.00070

[46] Lingling Zhao, Yao Kang, and Yonglin Wang. 2023. Application of Micro Frontend Technology in Network Public Opinion System. In *2023 IEEE 7th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Vol. 7. IEEE, Piscataway, NJ, USA, 227–230. doi:10.1109/ITOEC57671.2023.10291288