Guidelines for Adoption Micro-frontend Architecture

Giovanni Amorim
University of Brasília (UnB),
Department of Computer Science
Brasília–DF, Brazil
giovanni.amorim.ti@gmail.com

Larissa Rocha
Bahia State University
Salvador, BA, Brazil
larissabastos@uneb.br

Fabiana Freitas Mendes
Aalto University, Department of
Computer Science
Espoo, Finland
fabiana.mendes@aalto.fi

Rodrigo Pereira dos Santos
Federal University of the State of Rio
de Janeiro (UNIRIO)
Rio de Janeiro–RJ, Brazil
rps@uniriotec.br

Edna Dias Canedo
University of Brasília (UnB),
Department of Computer Science
Brasília–DF, Brazil
ednacanedo@unb.br

Abstract

Context: The rapid evolution of software development has led to challenges when integrating new technologies into monolithic frontend architectures. While microservices advanced backend modularity, frontends often face issues like code redundancy, scalability, and lack of modularity. The micro-frontend architecture emerged as a modular approach to address these challenges. **Problem:** Current monolithic frontend structures hinder agility and scalability in corporate environments, affecting both technology performance and business adaptability. This study targets challenges such as redundancy and the complexities of team coordination when scaling frontend projects. **Proposed Solution:** We developed a Guide for Adoption of Micro-Frontends (GAM), providing structured guidelines to support the effective implementation of micro-frontends in corporate contexts. **IS Theory:** This research draws upon the Technology Acceptance Model (TAM), evaluating the factors influencing the adoption of micro-frontend architecture within development teams. **Method:** We conducted a systematic mapping study (SMS) to investigate the impacts and challenges associated with adopting micro-frontends in software development. Next, we proposed the GAM and carried out a case study at Serasa Experian to assess the feasibility of the proposed guide. Additionally, we conducted a survey with development practitioners to explore their perceptions of the micro-frontend architecture and the workflows suggested by the GAM. **Results:** Results indicate that GAM is viewed positively by practitioners, assisting in the structured adoption of micro-frontends. Our case study highlights critical areas impacted by this approach, including performance, scalability, and parallel development, as well as issues with module complexity and developer experience. **Contributions and Impact on IS:** This research contributes a practical framework for the industry, addressing current gaps in micro-frontend implementation and setting a basis for further study on modular frontend architectures in Information Systems (IS). GAM provides actionable insights for practitioners, enhancing frontend modernization efforts with measurable impacts in agility and maintenance.

CCS Concepts

• **Software and its engineering** → *Software design techniques.*

Keywords

Micro-frontend, Guidelines, Architecture, Web Development

1 Introduction

The increasing adoption of micro-frontend architectures by large organizations represents a significant trend in software development [25]. This reflects the ongoing pursuit of more flexible and effective approaches to building frontend applications [22]. Pölöskei and Bub [20] noted that some organizations have developed their application portfolios by fragmenting monolithic structures into isolated components, with separate web endpoints for each service. However, managing a large monolith using this strategy can lead to reduced operational efficiency. In contrast, micro-frontends allow for the integration of an entire portfolio of distributed applications through the implementation of a single portal. As organizations face the need to evolve their applications rapidly, the micro-frontend concept has emerged as an effective response, enabling different parts of an application to be developed and deployed independently, thus promoting scalability and agility in development [29].

Micro-frontends enable independent development and deployment of different application parts, fostering scalability and agility [29]. However, their adoption brings challenges, including: 1) Code Redundancy; 2) Consistency; 3) Heterogeneity; and 4) Increased Frontend Code [5]. Code redundancy occurs because multiple teams often work on different parts of the application, leading to duplicated efforts and code [20]. Consistency issues arise when several teams are involved in developing different modules or functionalities, resulting in a lack of cohesion among the elements and factors related to user interaction with the application—commonly referred to as User Experience (UX) [25]. This inconsistency can lead to a negative perception and decreased efficiency in development [20].

Heterogeneity addresses the flexibility to choose different technology stacks for various parts of the application [10]. Geers [5] asserts that the ability to use multiple technology stacks does not necessarily mean that they should be employed indiscriminately. When all teams opt to use the same technologies, the key benefits of autonomous version updates and reduced communication overhead are still retained. However, utilizing various technologies creates heterogeneity, which in turn increases complexity in integration and maintenance, posing additional challenges for the development team. Finally, the issue of increased frontend code refers to the

need to write more JavaScript and CSS code, which can also lead to redundancy and negatively impact application performance. Davide and Mezzalira [27] proposed four key decisions that should be considered before starting a project with micro-frontends: 1) Horizontal or Vertical Division; 2) Composition Side; 3) Routing and 4) Communication between Micro-Frontends.

Given the limitations of large monoliths, such as code redundancy, consistency issues, and complex architectures [5, 6], along with key architectural decisions [10], this study explores the impacts and challenges of adopting micro-frontends in complex web applications. In addition, we propose a guideline compiling best practices for micro-frontend adoption to support practitioners and organizations in implementing the micro-frontend architecture.

The main contributions of this study include: a) *Design and Architecture Patterns*: identifying patterns to promote consistency and reuse in distributed ecosystems; b) *CI/CD Practices*: identifying tools and practices that streamline the delivery of micro-frontends; c) *Survey Insights*: capturing industry knowledge on relevant technologies; and d) *GAM*: a guideline compiling best practices for micro-frontend adoption. These findings highlight both the benefits and challenges of micro-frontends in practice.

2 Related Work

Peltonen[15] explored the adoption of micro-frontends by large companies such as DAZN, Ikea, and Starbucks, which break down monolithic front-end applications into smaller, independent micro-applications. The study aimed to map the existing knowledge on micro-frontends by understanding the motivations behind their adoption, as well as the associated benefits and challenges. The authors conducted a Multivocal Literature Review and identified 173 studies, 43 of which reported on the motivations, benefits, and challenges related to micro-frontends. The findings revealed that as both the application and development team grow, traditional architectural options become inadequate, particularly when multiple teams are required to work on the same front-end application.

Yang [33] focused on the application of micro-frontends architecture to address common challenges in content management systems (CMS) for large or medium-sized projects, where business growth can make the front-end difficult to scale and maintain. It discussed how the separation between the front-end and back-end, combined with the maturity of related technologies, offers a more efficient and scalable solution for large-scale web applications. Among the key benefits highlighted, the adoption of micro-frontends enables better scalability and easier deployment, while modularizing the application into smaller, independent components.

Nishizu [12] explored challenges in data transfer between independently developed front-end applications. The study proposed using signals in reactive programming to improve modularity and simplify data flows in micro-frontends. Traditional methods relying on events and callbacks can create complex, hard-to-maintain code. By representing web components as signals, the approach achieved more declarative and understandable data flows. A comparison with standard Web Components API and React confirmed better modularity in the signal-based version.

Stefanovska [26] investigated code reuse as one of the key benefits of the micro-frontends architecture. The authors explored how this approach facilitates code reusability, enabling better organizational structure, which in turn reduces time-to-market and increases the scalability of applications. They conducted an evaluation of existing micro-frontends architectures based on defined qualitative attributes, resulting in a technical solution. The implementation process is divided into two phases: decomposing an existing front-end application and reusing the code by integrating the decomposed components into a new micro-frontends-based application. Pölöskei [19] examined the challenges of migrating monolithic front-ends to micro-frontends, particularly in cloud-native solutions. The study highlighted issues of code reuse, duplication, and conflicts during the transition. These challenges are more prominent in complex corporate environments with multiple vendors. To address them, the authors proposed a migration method with guidelines to minimize negative impacts. They also suggested future work to enhance code reuse and conflict management strategies.

Taibi [28] identified several negative impacts, such as increased operational complexity. The study revealed that implementing micro-frontends can introduce significant complexities. The setup and management of multiple independent applications require substantial effort, resulting in a steep learning curve. This rise in operational complexity was a major challenge encountered in DAZN's experience, as detailed in the study. Additionally, the research highlighted issues related to code redundancy and conflicts. Decomposing the front-end into micro-applications can lead to code duplication and difficulties in maintaining consistency across the independent parts. Another significant negative impact identified was the complexity in configuring and managing distributed applications. The study emphasized that coordinating and integrating multiple micro-frontends can be challenging, requiring careful management of environments and synchronization across different applications.

3 Study Settings

In this work, we first conducted a Systematic Mapping Study (SMS) following the guidelines proposed by Petersen et al. [18], with the aim of identifying the impacts and challenges associated with the implementation of micro-frontends architecture. The studies selected in the SMS allowed us to identify best practices and challenges related to the adoption of micro-frontends. Based on these findings, we propose the Guide for the Adoption of Micro-Frontends (GAM) to support practitioners in implementing and adopting the micro-frontends architecture. Table 1 presents the views and architectural patterns used in the implementation of micro-frontends, along with the impacts and challenges reported by the studies selected in the SMS. The entire protocol for conducting the SMS is available on Zenodo at https://zenodo.org/records/14063897.

During the development of the Guide for the Adoption of Micro-Frontends, we proposed an iterative process outlining the phases that organizations should follow to adopt this architecture. This process is divided into four phases: Feasibility Study, Practical Implementation, Expansion, and Pilot Project. Each phase is correlated with knowledge areas from the Software Engineering Body of Knowledge (SWEBOK), as shown in Table 2. This correlation ensures that the activities conducted in each phase align with best practices in software engineering. To assess the feasibility of the

Table 1: Views and Architectural Patterns Used in Micro-frontends Implementation

ID	Views	Patterns	Impacts and challenges	Ref.
S1	API, App Container, Devops, Orchestration, Single-SPA	Frameworks (Angular, React and Vue), Languages (Javascript, HTML, CSS), DDD, Application Shell, Communication, Composition, Iframe, Routing	Team scalability, Deployment independence	[15]
S2	API, App Container, Devops, Orchestration, Single-SPA, Web Components	Frameworks(Angular and React), Languages (Javascript, HTML, CSS, Typescript), Composition, Iframe, VCS, Routing, MVC	Operational complexity, increased efficiency in deliveries	[14]
S3	API, BFF, Module Federation, Single-SPA	Framework (React), Composition	Presents the implementation of chatbots with Micro-frontend architecture	[11]
S4	API, App Container, BFF, Orchestration, Single-SPA	Communication	Data optimization, network efficiency, and resource savings	[1]
S5	API, Single-SPA, Web Component	Frameworks (Angular and React), Languages (Javascript, HTML, CSS), Design System, Communication, Iframe, Routing, CMS	Team independence, deployment efficiency, CSS conflicts, and code redundancy	[33]
S6	Module Federation, Single-SPA, Web Components	Languages (Javascript, HTML, CSS), Application Shell, Design System, Communication, Composition, Iframe, Routing	Deployment independence and team autonomy	[17]
S7	App Container, Single-SPA	Framework (Vue), Programming Languages (HTML, CSS), Communication	Development of a module in micro-frontend for HMI	[21]
S8	API, Web Components	Framework (Angular), Programming Languages (Typescript), Design System, Communication	Proposes micro-frontend for modular industrial plant processes	[7]
S9	API, Single-SPA	Frameworks (Angular and Vue), Languages (Javascript, Typescript, HTML, CSS), Design System, Routing, MVC	Prototype development, high performance, reduction of development complexity	[32]
S10	API, Single-SPA, Web Components	Frameworks (React and Vue), Languages (Javascript, CSS), Routing	Inefficiency in team management for small projects, increased operational complexity	[12]
S11	API, App Container, BFF, Devops, Single-SPA, Web Component	Frameworks (Angular, React and Vue), Languages (Javascript, HTML, CSS), Shell Application, Iframe, VCS, Routing, Composition	Implementing migration to micro-frontend and CSS conflicts	[19]
S12	API, App Container, Module Federation, Single-SPA, Web Components	Frameworks (Angular, React and Vue), Languages (Javascript, HTML, CSS), DDD, Routing, Composition, Communication, Iframe	Efficiency in code reuse	[26]
S13	API, Single-SPA	Framework (Angular), Languages (HTML, CSS), Communication, Composition	Efficiency in team management and CSS conflicts	[13]
S14	API, Web Components	Frameworks (Angular and React), Languages (Javascript, Typescript, HTML, CSS), Communication, Composition	Reusability with plugins	[2]
S15	API, DevOps	DDD, Communication, Composition, Design System	Development of a micro-frontend solution for SMEs	[24]
S16	API	Application Shell	Development of a platform to organize and identify functional requirements	[30]
S17	API, Single-SPA, Web components	Languages (Javascript, HTML), Application Shell, Design system, Communication	Development of a micro-frontend solution for small companies	[8]
S18	Module Federation, Orchestration, Single-SPA	Frameworks (Angular, React and Vue), Languages (Javascript, HTML, CSS), Design System, Communication, Composition	Efficiency in team management	[16]
S19	API, Orchestration, Web Components	Frameworks (Angular, React and Vue), Communication, Composition, Iframe, Routing	Effectiveness in implementing dynamic interfaces	[9]
S20	API, Module Federation, Orchestration, Single-SPA, Web Components	Frameworks (Angular), Languages (Javascript, HTML), Application Shell, Design System, Communication, Composition, Iframe, Routing	Increase in operational complexity	[28]
S21	API, Module Federation, Orchestration, Single-SPA	Frameworks (Angular, React and Vue), Languages (Javascript, Typescript, HTML and CSS), DDD, Routing	Greater scalability and increased operational complexity	[23]
S22	App Container, Single-SPA, Web Component	Frameworks (Angular, React and Vue), Languages (Javascript, Typescript, HTML and CSS), DDD, Communication, Iframe, VCS, Routing, MVC	Improved user experience, decoupling, application scalability, and cost reduction	[31]
S23	API	Routing	Proposes a method based on a bus for micro-frontend	[34]
S24	API, Single-SPA, Web Components, Module Federation	Languages (Javascript, Typescript, HTML and CSS), DDD, Communication, Composition, Iframe, Routing	Explores aspects of microfrontends, such as their benefits and challenges	[4]

GAM, we conducted a practical case study within a company, followed by a survey with the development team practitioners who participated in the case study. This survey aimed to gather their perceptions regarding the micro-frontend architecture and the workflows suggested by the GAM.

4 Guidelines for Adopting Micro-Frontends (GAM)

In understanding the journey of adopting micro-frontends and in addition to the strategies addressed by Luca Mezzalira [10], we must visualize the scenario comprehensively, capturing the critical nuances that shape the success of this architectural adoption. This holistic perspective, known as the “Big Picture”, offers a broad and interconnected view of the key stages involved, outlining a structured flow that guides from the Feasibility Study to the Expansion phase of the application. All details of the Guide can be viewed at https://guidemfe.github.io/gam/.

The Viability Study stage of the GAM is the phase that seeks to analyze the managerial and technical Viabilities before deciding on the adoption of the Micro-Frontends. This assessment is subdivided into two distinct moments: Technical Feasibility and Managerial Feasibility. The technical feasibility study precedes the managerial feasibility study. This approach ensures that management has sufficient artifacts and information to analyze, for example, the costs associated with the transition, contributing to an informed decision. The flow of analyses necessary for the team to decide on

Table 2: GAM Phases and SWEBOK Knowledge Areas

GAM Phases	Swebok KAs	Correlation
Feasibility Study	Software Engineering Management, and Economics	During the feasibility study, managerial and economic aspects are analyzed to assess the project's technical and viability.
Practical Implementation	Software Design, Construction, Configuration, Models and Methods	In this phase, design principles, construction practices, and configuration management are applied, adhering to software engineering models and methods to ensure efficient and controlled development.
Expansion	Software Maintenance, Software Quality, Software Engineering Models and Methods	The Expansion phase focuses on continuous maintenance, ensuring software quality and applying appropriate models and methods to support system growth.
Pilot Project	Process, Professional Practice in Software Engineering	Best practices in software engineering and professional experience are employed to validate the guide in a controlled and realistic environment.

the adoption or not of the micro-frontends architecture is available at GAM.

Technical Viability:

- Technical Compatibility: The aim is to ensure that the Micro-Frontends is compatible with the existing technologies in the company's infrastructure, minimizing conflicts and promoting a smooth transition. Recommendations for this phase include: a) Assessment of Current Infrastructure: Conduct a detailed analysis of the existing technological infrastructure, identifying the technologies, frameworks, and libraries in use; b) Identification of Potential Conflicts: Identify possible conflicts between the technologies currently used and those associated with the Micro-Frontends architecture; c) Standardization of Technologies: Consider standardizing technologies that are common to both the current infrastructure and the Micro-Frontends architecture to facilitate integration; and d) Compatibility Testing: Implement specific tests using the pilot project to verify compatibility between Micro-Frontends components and the existing infrastructure.
- Integration Challenges: Aims to proactively identify and address integration challenges that may arise during the adoption of the Micro-Frontends architecture. Recommendations for this phase include: a) Interface Mapping: Clearly map the interfaces between the Micro-Frontends and other components of the system; b) Definition of Contracts: Establish clear and documented contracts between the Micro-Frontends and backend services, ensuring consistency in interactions; and c) Continuous Monitoring: Implement mechanisms for continuous monitoring to identify and promptly resolve integration issues.
- Functional Complexities: Aims to understand and mitigate the functional complexities associated with the Micro-Frontends architecture, ensuring efficient delivery of functionalities. Recommendations for this phase are: a) Requirements Analysis: Conduct an in-depth analysis of the application's functional requirements, identifying potential complexities; b) Modular Design: Adopt a modular design for the Micro-Frontends, dividing the application into independent components to facilitate managing complexities; c) Unit and Integration Testing: Implement unit and integration tests to validate the functionality of Micro-Frontends individually and collectively; and d) Clear Documentation: Maintain clear documentation describing the functional interactions between Micro-Frontends and other components.

Managerial Viability:

- Strategic Objectives Alignment: This aims to ensure that the implementation of the micro-frontend architecture is aligned with the organization's strategic objectives. The recommendations are: a) Management Impact Analysis: Conduct an analysis of the impact of gradually introducing micro-frontends on management's strategic objectives. Evaluate the feasibility of gradual alignment without forcing an immediate transition; b) Benefits Mapping: Identify the potential benefits of gradually adopting micro-frontends for the company's strategic objectives, prioritizing alignment with existing goals; c) Strategic Goal Adjustments: Analyze the need to adjust or realign strategic goals to optimize the adoption of micro-frontends, ensuring they are aligned with the organization's long-term vision.
- Costs and Budget: It aims to evaluate the financial impacts related to the adoption of micro-frontends architecture in the GAM context, identifying associated costs and ensuring effective budget management. Recommendations for this phase include: a) Detailed Cost Assessment: Perform a survey of costs associated with the implementation of micro-frontends. Include expenses related to team training, potential infrastructure upgrades, and acquisition of specific tools; b) Flexible Budget: Establish a flexible budget that allows for adaptations as the micro-frontends adoption project evolves. Incorporate a margin for unforeseen circumstances and adjustments during the initial implementation phases; and c) Return on Investment (ROI) Evaluation: Develop metrics for assessing return on investment. Monitor key indicators to ensure that expected benefits align with the financial resources employed.
- Change Management and Training: In this phase, the organization's ability to deal with the changes that the adoption of the Micro-Frontends architecture may bring should be evaluated. The goal is to identify impacts on processes, organizational culture, and team structure, and to prepare the team for the transition. Recommendations include: a) Effective Communication: Establish a clear and open communication plan to inform team members about planned changes. Communicate the benefits of the new architecture, highlighting how it will contribute to the company's strategic objectives; b) Personalized Training: Develop specific training programs for teams directly involved with Micro-Frontends. Ensure team members have the necessary skills to work efficiently with the new architecture; and c) Define Milestones and Metrics: Set clear milestones and measurable metrics to assess progress during the transition. Defining success indicators will help monitor the effectiveness of change management strategies and identify areas that need adjustment.

4.1 Practical Implementation

Practical implementation is the point where theoretical choices begin to materialize, and each decision has a direct impact on the efficiency, scalability, and maintainability of the system. The flow of

operational and integration decisions during this phase is available at GAM.

Operational decisions include:

- Build and Deploy: The aim is to implement an efficient compilation and deployment process to enable quick and independent updates of each micro-frontend. It is recommended to: a) Continuous Integration and Delivery (CI/CD): Utilize continuous integration and delivery (CI/CD) tools to automate the compilation, testing, and deployment process. Ensure that every code change is submitted to an automated pipeline to validate and deploy automatically; b) Pipeline: Configure continuous delivery pipelines to ensure smooth integration and rapid deployments. Automate the generation of artifacts, such as containers or packages, ready for deployment in production environments; c) Orchestration: Explore tools like Kubernetes for container orchestration and scalability; d) Logs and Tracking: Integrate logging and tracking systems to monitor events during the compilation and deployment process. This facilitates quick identification of issues and performance analysis; and e) Performance Metrics: Implement performance metrics during compilation and deployment to monitor the efficiency of the process. Utilize tools that provide real-time visibility and alerts for critical events.
- Code Sharing: The aim of this phase is to facilitate and optimize code sharing among different micro-frontends, promoting an efficient and collaborative approach in distributed development. Recommendations include: a) Identification of Common Functionalities: Identify common functionalities that can be used by libraries; b) Web Components and Design System: Create Web Components to encapsulate specific functionalities, ensuring modularity and reusability, and integrate them into the organization's Design System, ensuring visual and interaction consistency across all micro-frontends; and c) Version Control: Implement a version control system such as git using package management tools such as npm or yarn.
- Performance: Aims to ensure efficient performance in the micro-frontends architecture, optimizing loading, rendering, and interaction of interfaces to provide an agile and responsive user experience. Recommendations include: a) Asynchronous Module Loading: Adopting asynchronous loading strategies, such as Lazy Loading, to load modules only when necessary, reducing initial loading time; b) Rendering Optimization: Using techniques like Virtual DOM or Shadow DOM to optimize rendering, minimizing unnecessary updates to the user interface; c) State Management: Implementing state management by choosing a solution that minimizes excessive reactivity and maintains consistent state among micro-frontends; and d) Resource Caching: Implementing caching strategies for static and dynamic resources, reducing the need to repeatedly fetch the same data from the server.
- Organization Structure: It should establish an efficient organizational structure for the implementation of Micro-Frontends, promoting seamless collaboration between teams and ensuring cohesion in feature delivery. It is recommended: a) Squad Model: Implement the Squad model, where cross-functional and autonomous teams are responsible for specific features. This promotes specialization and agility, favoring the independent delivery of micro-frontends; b) Agile Methodology: Adopt agile methodologies, such as Scrum or Kanban, to manage the development of micro-frontends. Regular sprints, retrospective meetings, and agile practices strengthen collaboration and continuous adaptation; c) Collaboration Tools: Use collaboration tools, such as Slack, Microsoft Teams, or other communication platforms. These tools facilitate communication between distributed teams, promoting effective information exchange; and d) DevOps Culture: Foster a DevOps culture, integrating development and operations.

Integration decisions include:

- Define: The aim is to identify and define the micro-frontend from the perspective of its division. We can decide to have multiple micro-frontends in the same view (Horizontal Division) or have only one micro-frontend per view (Vertical Division) [10]. Recommendations include: a) Effective Coordination - Horizontal Division: Given the presence of multiple micro-frontends in the same view, it's essential to establish effective channels of communication and coordination between teams to ensure cohesion in the user experience; b) Rigorous Governance - Horizontal Division: Due to the flexibility offered by this approach, it's crucial to implement rigorous governance to prevent uncontrolled proliferation of micro-frontends. Establish clear guidelines to ensure consistency and quality; c) Domain Responsibility - Vertical Division: When adopting vertical division, each team is responsible for a specific business domain. It's recommended to apply Domain-Driven Design (DDD) principles [3] to ensure an architecture aligned with business requirements; and d) Exploration of DDD - Vertical Division: While not common in frontend architectures, vertical division provides a good reason to explore DDD principles. Consider applying concepts such as aggregates and bounded contexts to improve modeling and cohesion.
- Composition: The strategy for composing micro-frontends should be chosen, covering Client-Side, Server-Side, or Edge-Side. For each of these strategies, the following is recommended: a) Client-Side: This strategy facilitates the dynamic integration of micro-frontends into the client application shell, allowing different parts of the interface to be loaded efficiently. The use of iframes, frameworks such as Single-SPA, Angular, React, and Vue, as well as Web Components, is recommended.
  b)Edge-Side: The visualization should be assembled at the CDN level, optimizing the global delivery of the application through strategies such as Edge Side Includes (ESI). Consider using Edge Side Includes (ESI) for edge-side composition, leveraging the scaling capabilities offered by CDNs and facilitating visualization assembly.
  c) Server-Side: Allow the origin server to compose the view, retrieving different micro-frontends and assembling the final page. To achieve this, an analysis of use cases should be conducted, and clear strategies for scalability and cacheability should be defined.

- Route: Aims is to ensure a cohesive and agile user experience aligned with the selected composition strategy (origin, edge-side, or client-side). This includes routing page requests to the correct micro-frontends based on project needs. Recommended approaches are: a) Server-Side Routing using servers like Nginx for routing and load distribution; b) Edge-Side Routing with CDNs like Cloudflare or Akamai supporting Edge Side Includes (ESI); and c) Client-Side Routing using libraries like React Router or Vue Router for efficient navigation in a SPA application shell managed by frameworks like Angular or React.
- Communication: Aims to ensure consistent interaction among different components, even when they are distributed across multiple teams or parts of the application. It is recommended to: a) Event Emitter: Implement an Event Emitter or utilize native JavaScript event functionality to enable asynchronous communication between micro-frontends; b) Custom Events: Adopt custom events like CustomEvent to notify other micro-frontends about specific events; c) EventBus Injection: Consider injecting an EventBus into the micro-frontends container to facilitate global communication; d) Local Storage (Web Storage): Utilize local storage (localStorage) to share persistent data between micro-frontends; and e) Query Strings: Explore data passing via query strings for specific cases, such as product identification or page information.

4.2 Expansion

This phase spans from selecting pilot projects to expanding to other components of the system. Practical aspects, continuous learning, team training, and constant monitoring are key elements to ensure the success of the expansion. The team is encouraged to collect feedback, make adjustments to the strategy as needed, and continuously improve the implementation based on lessons learned throughout the process. The suggested strategies for implementing this phase considering the scalability and maintainability of the system intended to adopt the micro-frontend architecture (GAM).

Scalability The scalability aims to empower the application to grow efficiently and adapt to the variable demands of the environment. The main areas of focus include adding new micro-frontends, managing load, and caching strategies.

- Addition of New Micro-Frontends: Aims to facilitate the incorporation of new micro-frontends into the existing system. Recommendations include: a) Dynamic Modularization: Develop a modular architecture that supports the dynamic addition of new micro-frontends without impacting existing ones; b) Load Balancing: Implement an automatic load balancing system such as orchestration services, using tools like Kubernetes, to facilitate automatic scaling based on traffic metrics; and c) Cache Management: Adopt caching strategies, such as Redis, to temporarily store microservices responses, improving the throughput of micro-frontends composition. Explore storing micro-frontends DOM in in-memory caches to reduce the need for composition on every request.
- Load Management: The aim is to ensure operational efficiency and optimized performance of the application, even in the face of increasing demands. Recommendations include: a) Cloud Auto-Scaling: Utilize auto-scaling features provided by cloud providers to dynamically adjust infrastructure based on traffic patterns. Choose efficient computing layers such as containers for rapid execution, and consider managed options like serverless services to simplify infrastructure operationalization; b) Load Prediction and Manual Adjustment: Establish a baseline infrastructure capable of handling predictable loads, such as Black Friday sales, by adopting practices of comparison between different services and plug-and-play options; and c) Latency Optimization with CDN: Utilize a CDN to enhance web page delivery speed, reducing latency between the client and requested content.
- Cache Strategies: The aim is to optimize application performance by reducing server load and speeding up response time. Recommendations include: a) Static Content Cache: Implement caching for static content such as images, stylesheets, and scripts, reducing loading latency; b) Dynamic Data Cache: Employ caching strategies for dynamic data, minimizing frequent server queries; c) DOM Storage Cache: Store the complete DOM of micro-frontends in in-memory caches to avoid composition with every request; d) Content Delivery Network (CDN): Utilize a CDN for static content caching, distributing copies of static resources to servers located in different geographical regions.

Maintainability In the context of GAM, maintainability emphasizes the code; we adapted the concepts to focus on Analysability, Modifiability, and Testability, highlighting the importance of efficient analysis, effective modification, and robust testing of the source code in micro-frontends environments.

- Analysability: aims to understand the structure and functioning of micro-frontends to diagnose problems quickly and accurately. Recommendations include: a) Naming Conventions: Adopt consistent naming conventions to facilitate the identification and understanding of components; b) Documentation: Maintain detailed documentation about the architecture, interactions, and responsibilities of each micro-frontend; c) Static Analysis Tools: Use tools such as ESLint and TSLint for static code analysis, identifying patterns, complexity, and potential issues. More comprehensive tools like Sonar can provide a holistic view of code quality, including source code metrics, code coverage, and detection of potential security issues.
- **Modifiability:** It should allow efficient modifications to the micro-frontends without introducing defects or degrading quality. It is recommended: a) Modular Design: Structure the micro-frontends in a modular way, minimizing coupling between them; b) Continuous Refactoring: Encourage refactoring practices to keep the code clean and adaptable; c) Version Control: Use efficient version control systems, such as Git, to track changes and facilitate reversions if necessary. Tools like Veracode can be incorporated to perform static and dynamic security analyses, helping ensure safe modifications.
- **Testability:** Aims to ensure the effectiveness of testing, allowing for early detection of failures and safe changes. It

suggests the following recommendations: a) Unit and Integration Testing: Implement unit tests (using Jest and Enzyme for React, for example) and integration tests to verify isolated functionality and interaction between micro-frontends; b) Test Automation: Utilize test automation tools, such as Cypress, to streamline the verification process; and c) Production-Like Test Environments: Maintain test environments that faithfully reproduce production conditions to validate the real system behavior.

Pilot Project

The pilot project was involved in all stages, from the feasibility study to the expansion phase. This was necessary as it allowed for the validation of technical and operational decisions in a controlled environment. Throughout the process, the pilot project served as a checkpoint to ensure that the strategies and guidelines defined in the GAM were practical and effective. During this phase, we will highlight not only the technical implementation but also the lessons learned and necessary adjustments as the team gains valuable insights into the practical application of the micro-frontends paradigm. The Pilot Project offers a unique opportunity to assess the effectiveness of architectural and integration choices in a controlled context before broader expansion to other projects or areas of the system. The selection of the Pilot Project is a pivotal point for the success of Micro-Frontends adoption. The selection of the pilot project is a pivotal point for the success of micro-frontends adoption. At this stage, stakeholders should consider the following aspects:

- *Project Complexity:* Evaluate the overall complexity of the project in terms of functionalities, interactions, and number of components. More complex and extensive projects may provide a better understanding of the challenges and benefits of using Micro-Frontends. However, it is crucial to note that it is prudent to limit the testing to some components or modules of the project, especially if it is of significant importance to the company, in order to avoid undesired compromises.
- *Diverse Component:* Look for projects that have a representative variety of components. This allows exploring how different types of functionalities can be encapsulated in micro-frontends and integrated efficiently.
- *Representativeness of the Scenario:* Ensure that the selected project is representative of the broader environment of the organization. A pilot that reflects the typical challenges and requirements of frontend development within the company will provide more applicable insights.
- *Potential Impact:* Consider the potential impact of the pilot project on the organization. Choosing a project with significant yet controlled impact allows for evaluating outcomes without compromising the overall stability of the system.

5 Case Study

The case study was based on modules from a generic system that represents a pilot project used to illustrate the approaches adopted by Serasa Experian Company [1]. This pilot project, referred to as “Products" in this research, resulted from a detailed analysis of the company’s needs, followed by meetings and discussions with key stakeholders. During this phase, the development team conducted a system requirements analysis and planned the micro-frontend architecture. The activity included identifying the different components of the application, defining which parts could be decoupled into micro-frontends, organizing the interactions between them and developing documentation. We opted for the Horizontal Decision, which is suitable for projects that require a consistent evolution of the user interface and a seamless experience across various views. All artifacts (source code) generated in the case study are available at https://github.com/guidemfe.

- Root (exp-root-config): It’s the root micro-frontend that acts as the orchestrator. Maintained by Team A, its role is to configure the application using the Single-Spa framework as a base and define the entry points for the other micro-frontends (Click here to access the repository).
- Shell (exp-shell-mf): It is the micro-frontend responsible for providing the basic layout of the application, including the page structure such as header, footer, and navigation. The Shell dynamically loads other micro-frontends based on the current URL. It was developed using the React framework and is maintained by Team A (Click here to access the repository).
- Navbar (exp-navbar-mf): It is the micro-frontend responsible for displaying the navigation bar of the application. Maintained by Team A, it utilizes the React framework (Click here to access the repository).
- Catalog (exp-catalog-mf): It is the micro-frontend responsible for displaying the product catalog. It allows users to view available products, add products to the cart, and publishes events when a product is added to the cart. The React framework was used, and it is maintained by Team B (Click here to access the repository).
- Cart (exp-cart-mf): It is the micro-frontend responsible for displaying the shopping cart. It receives events when a product is added to the cart from the catalog, maintains the state of items in the cart, updating it as products are added, calculates the total purchase based on the items in the cart, and provides the functionality to clear the cart. The React framework was used, and it is maintained by Team B (Click here to access the repository).
- Checkout (exp-checkout-mf): It is the micro-frontend responsible for displaying the checkout page. It displays contact information and payment options and allows the user to complete the purchase. The React framework was used, and it is maintained by Team B (Click here to access the repository).
- Home (exp-home-mf): It is the micro-frontend responsible for displaying the application’s home page. It can include general information about the store, promotional banners, or featured product categories. The Angular framework was used, and it is maintained by Team C (Click here to access the repository).
- Footer (exp-footer-mf): It is the micro-frontend responsible for displaying the application’s footer. The Angular framework was used, and it is maintained by Team C (Click here to access the repository).

[1] In accordance with the company’s code of ethics Serasa Experian, no portion of the source code, whether in whole or in part, was taken from actual applications

5.1 Survey Results

We conducted a survey with the case study participants who implemented techniques and methodologies for the micro-frontend architecture, following the patterns and insights identified in the SMS. The objective of this survey is to ratify the GAM processes, presented in Section 4. Survey consisted of 16 questions, as presented in Table at Zenodo. The questions addressed various aspects of the project, such as objectives, challenges faced, lessons learned, and team perceptions about transitioning to the new architecture.

In survey, we had the participation of 8 practitioners from the company Serasa Experian (P1 to P8), with different roles, levels of experience, and involvement with micro-frontends. Only 12.5% of the participants hold a master's degree. 37.5% of the participants hold an undergraduate degree in IT, and 50% of them have a specialization course in IT. Concerning the length of employment and experience that the participants have in the technology or software engineering field (Q2 from Table at Zenodo), 25% of the participants have more than 10 years of experience in the IT field. 25% of the participants have between 7 and 9 years, and 50% of the participants have between 4 and 6 years of experience in the field.

Regarding their role in the organization (Q3 in Table at at Zenodo), 25% of the participants stated they are tech leads, while 25% hold agile roles such as Scrum Master (SM) or Product Owner (PO), and the remaining 50% are software developers. The experience also has a significant impact on the implementation of new technologies (Q4 in Table at Zenodo). 12.5% of the participants have less than 1 year of experience at Serasa Experian, 12.5% of them have between 4 to 6 years, and the majority of practitioners working at Serasa Experian, 75% of them have between 1 to 3 years in the company. We also investigated which programming languages the participants had experience with (Q5 in Table at Zenodo). 75% of the participants reported having experience with JavaScript, 75% with TypeScript, 50% with Java, and only 12.5% of the participants do not have experience with these programming languages. It is noteworthy that this last percentage corresponds to agile roles, supporting the observation that this profile generally does not excel in programming skills but plays an important role in managing developers' activities.

Frameworks are indispensable tools for the adoption of the micro-frontend architecture. In this context, we asked the participants about their experiences with the main frameworks used in the adoption of micro-frontends (Q6 in Table at at Zenodo). 87.5% of the participants have experience with the Angular framework, 50% have experience with Spring Boot, 37.5% have experience with React, 25% have experience with Vue, and 12.5% have no experience with any of these frameworks.

In relation to the level of involvement of the participants with frontend architectural decisions at Serasa Experian, Q7 in Table at Zenodo aimed to understand the level of participation and influence of practitioners in decisions related to frontend architecture. This allows for assessing the alignment between the development team and the company's architectural strategies. 50% of the participants stated that the level of involvement is moderate, 25% reported a low level of involvement, and 25% stated they had a high level of involvement in frontend architectural decisions. Following the same approach, we also asked about the participants' familiarity with micro-frontend architecture (Q8 in Table at Zenodo). 37.5% reported having moderate familiarity, 25% reported low familiarity, 25% reported high familiarity, and finally, 12.5% of the participants reported having no familiarity with micro-frontend architecture, as presented in Figure 1.

Figure 1: Involvement in frontend development and knowledge of micro-frontend architecture

Questions Q9 and Q10 of Survey (Table at at Zenodo) were defined to understand the relationship between micro-frontend architecture, project size, and team size, respectively. Regarding Q9, which deals with project size, small projects are understood to have few features, requirements, and low complexity. Medium-sized projects have a moderate number of features and requirements with intermediate complexity. Finally, large projects are those with many features and requirements with high complexity. 25% of the participants stated that they work on large projects, and 75% work on medium-sized projects. On the other hand, the results of question Q10 present data that align with what is suggested by teams adopting agile methodology, where 100% of the participants chose the option "from 4 to 8 people".

Regarding the adoption and implementation of the micro-frontend architecture at Serasa Experian, in questions Q11 and Q12 (Table at Zenodo) respectively, we asked participants about their expectations and perceptions regarding the importance of POCs (Proof of Concepts) and pilot projects as tools for evaluating the adoption of the micro-frontend architecture. 25% of the participants had low expectations, while 37.5% attributed moderate expectations. Finally, the same percentage of participants, 37.5%, stated having high expectations regarding the micro-frontend architecture. Regarding the importance of POCs (Q12 in Table at Zenodo), all participants were unanimous regarding their high importance.

In question Q13 (Table at Zenodo), participants were asked to identify, from a list, the challenges faced by the team during the adoption and implementation of the micro-frontends architecture. 62.5% of the participants selected challenges related to the integration of different technologies and frameworks. Next, 37.7% of the participants indicated performance issues due to resource demand on the client side, while 50% of the participants indicated complexity in defining responsibilities among micro-frontends. Additionally, 25% of the participants mentioned obstacles related to routing and navigation in the application, while only 12.5% of them stated that their team did not face any of the mentioned challenges. Question Q14 from Table at Zenodo aimed to identify the main positive impacts perceived by the practitioners during the adoption and implementation of the micro-frontends architecture. code maintenance was selected by 62.5% of the participants. Team collaboration was considered positive by 37.5% of the participants, 25% of the participants mentioned modularity and improved application

performance, respectively, as positive impacts and only 12.5% of the participants mentioned benefits related to parallel development. Figure 2 shows us the negative impacts of adopting and implementing the micro-frontend architecture (question Q15 from Table at Zenodo) as indicated by the practitioners.

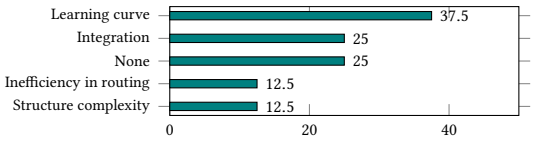

Figure 2: Negative impacts of adoption and implementation

37.5% of the participants stated that the learning curve was the biggest challenge faced by the team. 25% of the participants reported no negative impact on their team, and the same percentage was attributed to integration issues. Finally, 12.5% of the participants reported complexity in the architecture structure, and the same percentage was attributed to routing inefficiency. The survey had an open-ended question for participants to make any comments they deemed relevant. However, none of the 8 practitioners provided any comments.

5.2 Threats to Validity

In this section, we discuss the main threats to the validity of the GAM based on data collected from the case study and survey. These threats were identified throughout the process of developing, verifying, and validating the guide, and they may affect its applicability and effectiveness in different organizational contexts.

Internal Validity: Internal validity refers to the confidence that the results observed in the case study and survey truly reflect the impact of the strategies adopted in the development of the GAM, and are not influenced by external factors. For example, the case study was conducted in a single organization within the financial sector, which may limit the generalizability of the findings. The specific characteristics of the company's environment, culture, and team could have influenced the adoption of micro-frontends in a unique way, which may not necessarily apply to other organizations in different sectors. Additionally, the survey participants were selected based on their experience with technology and micro-frontends. However, there may be a response bias, as professionals more familiar with the architecture may have skewed opinions, whether due to past experiences or enthusiasm for the technology.

External Validity: External validity concerns the generalization of the results to contexts beyond those directly evaluated in the case study or survey. **Limited Generalization of the Case Study**: Although the case study provides a practical analysis of the GAM's application, it was restricted to a single organization. As a result, the guide's effectiveness in other sectors, with varying levels of technological maturity or organizational complexity, may differ. **Lack of Geographical and Sectoral Diversity in the Survey**: The survey targeted a specific audience of programming professionals but may not have captured a diverse range of opinions from different geographical regions or industry sectors. Organizations with distinct characteristics may face different challenges when adopting micro-frontends, which could require adjustments to the GAM. **Construct Validity**: Construct validity relates to the adequacy of the tools and methods used to assess the GAM. While the survey was useful for gathering professionals' perceptions, it is subject to biases, such as the interpretation of questions or responses that may not fully reflect real-world problems faced in daily practice. Additionally, the questions may not have sufficiently explored certain important aspects of micro-frontend adoption. In the case study, no quantitative metrics were used to evaluate the impact or success of applying the GAM (e.g., operational efficiency, software quality, performance, integration, and scalability metrics). The lack of objective metrics may limit the ability to accurately measure the guide's effectiveness in practice, making it harder to identify specific areas of improvement or success. This could affect the depth of the analysis and the strength of the conclusions drawn from the case study.

6 Conclusions

The case study and survey findings highlighted how micro-frontends can be practically adopted, leading to the development of the GAM framework. GAM assists practitioners and organizations by offering practical guidelines for selecting architectural practices, implementing consistent patterns, and ensuring a cohesive user experience across micro-frontends. It also provides a structured approach for integration and scalability, making it easier to adopt micro-frontends effectively.

The adoption of micro-frontends brings key benefits to software development, such as improved team efficiency and the flexibility for independent teams to make architectural decisions. This enhances productivity and responsiveness to market demands. The architecture also supports incremental updates, faster deployment, and the scalability required for dynamic applications, making it ideal for sectors with fast-paced development cycles.

Despite these advantages, challenges remain, particularly in managing technical complexity and ensuring a consistent user experience across micro-frontends. Managing multiple frontend technologies, maintaining state coherence, and addressing integration challenges require careful architectural planning and the use of best practices. Continuous refinement of strategies, investment in developer training, and the use of collaboration tools are essential to overcome these hurdles and fully realize the benefits of micro-frontends.

Data availability statement

The materials produced during this research, including the systematic mapping study (SMS), the survey form, and the file containing all survey responses, are available at https://zenodo.org/records/14063897. The GAM can be accessed at https://guidemfe.github.io/gam/.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, CNPq (Grant 316510/2023-8), FAPERJ (Grant E-26/204.404/2024), and UNIRIO.

References

[1] Amr S. Abdelfattah and Tomás Cerný. 2023. Filling The Gaps in Microservice Frontend Communication: Case for New Frontend Patterns. In *Proceedings of the 13th International Conference on Cloud Computing and Services Science, CLOSER 2023, Prague, Czech Republic, April 26-28, 2023*. SCITEPRESS, https://doi.org/10.5220/0011812500003488, 184–193. https://doi.org/10.5220/0011812500003488

[2] Fabian Bühler, Johanna Barzen, Lukas Harzenetter, Frank Leymann, and Philipp Wundrack. 2022. Combining the Best of Two Worlds: Microservices and Micro Frontends as Basis for a New Plugin Architecture. In *Service-Oriented Computing - 16th Symposium and Summer School, SummerSOC 2022, Hersonissos, Crete, Greece, July 3-9, 2022, Revised Selected Papers (Communications in Computer and Information Science, Vol. 1603)*. Springer, https://doi.org/10.1007/978-3-031-18304-1_1, 3–23. https://doi.org/10.1007/978-3-031-18304-1_1

[3] Eric Evans. 2004. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, https://www.bibsonomy.org/bibtex/2b613dcc2b969d6659bf549defb6f8dae/juve.

[4] Elvisa Gashi, Dhuratë Hyseni, Isak Shabani, and Betim Çiço. 2024. The advantages of Micro-Frontend architecture for developing web application. In *13th Mediterranean Conference on Embedded Computing, MECO 2024, Budva, Montenegro, June 11-14, 2024*. IEEE, https://doi.org/10.1109/MECO62516.2024.10577836, 1–5. https://doi.org/10.1109/MECO62516.2024.10577836

[5] M. Geers. 2020. *Micro Frontends in Action*. Manning Publications, https://books.google.com.br/books?id=FFD9DwAAQBAJ.

[6] James Lewis and Martin Fowler. 2014. Microservices. https://www.martinfowler.com/articles/microservices.html Acesso em 06 de janeiro de 2024.

[7] Julius Lorenz, Candy Lohse, and Leon Urbas. 2021. MicroFrontends as Opportunity for Process Orchestration Layer Architecture in Modular Process Plants. In *26th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2021, Vasteras, Sweden, September 7-10, 2021*. IEEE, https://doi.org/10.1109/ETFA45728.2021.9613474, 1–4. https://doi.org/10.1109/ETFA45728.2021.9613474

[8] Tomi Männistö, Antti-Pekka Tuovinen, and Mikko Raatikainen. 2023. Experiences on a Frameworkless Micro-Frontend Architecture in a Small Organization. In *20th International Conference on Software Architecture, ICSA 2023 - Companion, L'Aquila, Italy, March 13-17, 2023*. IEEE, https://doi.org/10.1109/ICSA-C57050.2023.00025, 61–67. https://doi.org/10.1109/ICSA-C57050.2023.00025

[9] 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. https://doi.org/10.1109/ACCESS.2019.2932196

[10] L. Mezzalira. 2021. *Building Micro-Frontends*. O'Reilly Media, https://books.google.com.br/books?id=MjpPEAAAQBAJ.

[11] Sabah Mohammed, Jinan Fiaidhi, Darien Sawyer, and Mehdi Lamouchie. 2022. Developing a GraphQL SOAP Conversational Micro Frontends for the Problem Oriented Medical Record (QL4POMR). In *Proceedings of the 6th International Conference on Medical and Health Informatics, ICMHI 2022, Virtual Event, Japan, May 13-15, 2022*. ACM, https://doi.org/10.1145/3545729.3545738, 52–60. https://doi.org/10.1145/3545729.3545738

[12] Yuma Nishizu and Tetsuo Kamina. 2022. Implementing Micro Frontends Using Signal-based Web Components. *J. Inf. Process.* 30 (2022), 505–512. https://doi.org/10.2197/IPSJJIP.30.505

[13] Nattaporn Noppadol and Yachai Limpiyakorn. 2021. Application of Microfrontends to Legal Search Engine Web Development. In *IT Convergence and Security*. Springer Singapore, Singapore, 165–173.

[14] Andrey Pavlenko, Nursultan Askarbekuly, Swati Megha, and Manuel Mazzara. 2020. Micro-frontends: application of microservices to web front-ends. *J. Internet Serv. Inf. Secur.* 10, 2 (2020), 49–66. https://doi.org/10.22667/JISIS.2020.05.31.049

[15] Severi Peltonen, Luca Mezzalira, and Davide Taibi. 2021. Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review. *Inf. Softw. Technol.* 136 (2021), 106571. https://doi.org/10.1016/J.INFSOF.2021.106571

[16] 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)*, Vol. 17. IEEE, 10.1109/AICT59525.2023.10313208, 1–6. https://doi.org/10.1109/AICT59525.2023.10313208

[17] A. Petcu, M. Frunzete, and D.A. Stoichescu. 2023. BENEFITS, CHALLENGES, AND PERFORMANCE ANALYSIS OF A SCALABLE WEB ARCHITECTURE BASED ON MICRO-FRONTENDS. *UPB Scientific Bulletin, Series C: Electrical Engineering and Computer Science* 85, 3 (2023), 319–334. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85169813523&partnerID=40&md5=f812d625e1144766be2156970c61590e cited By 0.

[18] K. Petersen, S. Vakkalanka, and L. Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

[19] István Pölöskei and Udo Bub. 2021. Enterprise-level migration to micro frontends in a multi-vendor environment. *Acta Polytechnica Hungarica* 18, 8 (2021), 7 – 25. https://doi.org/10.12700/APH.18.8.2021.8.1 Cited by: 4; All Open Access, Bronze Open Access.

[20] F. Rappl and L. Schöttner. 2021. *The Art of Micro Frontends: Build websites using compositional UIs that grow naturally as your application scales*. Packt Publishing, https://books.google.com.br/books?id=lGI0EAAAQBAJ.

[21] Muddasir Shakil and Alois Zoitl. 2020. Towards a Modular Architecture for Industrial HMIs. In *25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2020, Vienna, Austria, September 8-11, 2020*. IEEE, https://doi.org/10.1109/ETFA46521.2020.9212011, 1267–1270. https://doi.org/10.1109/ETFA46521.2020.9212011

[22] Bruno Simões, María del Puy Carretero, Jorge Martínez Santiago, Sebastián Muñnz Segovia, and Nieves Alcaín. 2023. TwinArk: A Unified Framework for Digital Twins based on Micro-frontends, Micro-Services, and Web 3D. In *The 28th International ACM Conference on 3D Web Technology, Web3D 2023, San Sebastian, Spain, October 9-11, 2023*. ACM, San Sebastian, Spain, 9:1–9:10. https://doi.org/10.1145/3611314.3615915

[23] Bruno Simões, María del Puy Carretero, Jorge Martínez Santiago, Sebastián Muñnz Segovia, and Nieves Alcaín. 2023. TwinArk: A Unified Framework for Digital Twins based on Micro-frontends, Micro-Services, and Web 3D. In *The 28th International ACM Conference on 3D Web Technology, Web3D 2023, San Sebastian, Spain, October 9-11, 2023*. ACM, https://doi.org/10.1145/3611314.3615915, 9:1–9:10. https://doi.org/10.1145/3611314.3615915

[24] Jonas Sorgalla, Philip Wizenty, Florian Rademacher, Sabine Sachweh, and Albert Zündorf. 2021. Applying Model-Driven Engineering to Stimulate the Adoption of DevOps Processes in Small and Medium-Sized Development Organizations. *SN Comput. Sci.* 2, 6 (2021), 459. https://doi.org/10.1007/S42979-021-00825-Z

[25] Single SPA. 2022. Single SPA - Getting Started Overview. https://single-spa.js.org/docs/getting-started-overview Acesso em: 10 Janeiro de 2024.

[26] Emilija Stefanovska and Vladimir Trajkovik. 2022. Evaluating Micro Frontend Approaches for Code Reusability. In *ICT Innovations 2022. Reshaping the Future Towards a New Normal - 14th International Conference, ICT Innovations 2022, Skopje, Macedonia, September 29 - October 1, 2022, Proceedings (Communications in Computer and Information Science, Vol. 1740)*. Springer, https://doi.org/10.1007/978-3-031-22792-9_8, 93–106. https://doi.org/10.1007/978-3-031-22792-9_8

[27] Davide Taibi and Luca Mezzalira. 2022. Micro-Frontends: Principles, Implementations, and Pitfalls. *ACM SIGSOFT Softw. Eng. Notes* 47, 4 (2022), 25–29. https://doi.org/10.1145/3561846.3561853

[28] Davide Taibi and Luca Mezzalira. 2022. Micro-Frontends: Principles, Implementations, and Pitfalls. *ACM SIGSOFT Softw. Eng. Notes* 47, 4 (2022), 25–29. https://doi.org/10.1145/3561846.3561853

[29] 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, Hyderabad, India, 1–4. https://doi.org/10.1109/HYDCON48903.2020.9242913

[30] 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, 10.1109/HYDCON48903.2020.9242913, 1–4.

[31] 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. In *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES-2020, Virtual Event, 16-18 September 2020 (Procedia Computer Science, Vol. 176)*. Elsevier, https://doi.org/10.1016/j.procs.2020.09.168, 1567–1576. https://doi.org/10.1016/J.PROCS.2020.09.168

[32] Sylvester Timona Wanjala. 2022. A framework for implementing micro frontend architecture. *Int. J. Web Eng. Technol.* 17, 4 (2022), 337–352. https://doi.org/10.1504/IJWET.2022.10054340

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

[34] C. Zhang, D. Zhang, H. Zhou, X. Wang, L. Lv, R. Zhang, Y. Xie, H. Liu, Y. Li, D. Jia, Y. Huang, and T. 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 Computer Society, Los Alamitos, CA, USA, 306–310. https://doi.org/10.1109/ISCSIC60498.2023.00070