

**A FRAMEWORK FOR IMPLEMENTING MICRO
FRONTEND ARCHITECTURE**

SYLVESTER TIMONA WANJALA

**MASTER OF SCIENCE IN
SOFTWARE ENGINEERING**

**JOMO KENYATTA UNIVERSITY
OF
AGRICULTURE AND TECHNOLOGY**

2026

A Framework for Implementing Micro Frontend Architecture

Sylvester Timona Wanjala

**A Thesis Submitted in Partial Fulfillment of the Requirements for
the Degree of Master of Science in Software Engineering of the
Jomo Kenyatta University of Agriculture and Technology**

2026

DECLARATION

This thesis is my original work and has not been presented for a degree in any other University.

Signature..... Date.....

Sylvester Timona Wanjala

This thesis has been submitted for examination with our approval as University Supervisors:

Signature..... Date.....

Dr. Lawrence Nderu, PhD

JKUAT, Kenya

Signature..... Date.....

Dr. Kennedy Malanga Ndenga, PhD

Karatina University, Kenya

DEDICATION

I dedicate this work to my late father, Alphonse, whose memory continues to inspire me, and to my mother, Jerida, for her unwavering support and encouragement. To my wife, Aulgah, and my children, Tetta and Alphonse, for their love and for making my studies easier. To my friends, Steve and Dennis, for their steadfast encouragement. To my brothers, Bena, Crispus, Cyprian, and Gregory, and my sisters, Sr. Bridgit and Eunice, for their prayers and support.

ACKNOWLEDGMENT

I express my heartfelt gratitude to the following individuals for their invaluable support throughout this project. I am profoundly thankful to God, our Creator, for providing me with strength, wisdom, and all necessities during this journey. To my late father, Alphonse Wanjala, and my mother, Jeridah Nekesa, I extend my deepest appreciation for their unwavering emotional and financial support. I am sincerely grateful to my supervisors, Dr. Lawrence Nderu and Dr. Malanga Kennedy Ndenga, as well as the entire staff of the JKUAT School of Computing and Information Technology, for their consistent, indispensable guidance and dedication throughout our collaboration. Special thanks are also extended to my friends for their constant encouragement and assistance during times of need. Your collective support has been instrumental in the successful completion of this project.

TABLE OF CONTENTS

DECLARATION.....	ii
DEDICATION.....	iii
ACKNOWLEDGMENT	iv
TABLE OF CONTENTS.....	v
LIST OF TABLES	x
LIST OF FIGURES	xi
ABBREVIATIONS AND ACRONYMS	xii
ABSTRACT	xiii
CHAPTER ONE	1
INTRODUCTION.....	1
1.1 Background	1
1.2 Problem Statement	2
1.3 Objectives.....	3
1.3.1 Main Objectives	3
1.3.2 Specific objectives.....	3
1.3.3 Research Questions	4
1.4 Scope.....	4
1.5 Justification	5

1.6 Structure	5
CHAPTER TWO	7
LITERATURE REVIEW	7
2.1 Introduction	7
2.2 Layered Architecture and the Monolithic Model.....	7
2.3 Micro-Services Architecture (MSA).....	10
2.3.1 Core Principles and Characteristics	11
2.3.2 Benefits of Microservices Architecture	12
2.3.3 Challenges and Limitations of Micro-Services Architecture.....	13
2.4 Micro Frontend Architecture.....	14
2.4.1 Conceptual Origins and Definition	14
2.4.2 Architectural Patterns and Composition Strategies.....	15
2.4.3 Review of Existing Micro Frontend Implementations.....	16
2.4.4 Security in Micro Frontend Architecture	19
2.4.5 User Experience Consistency in Micro Frontend Architecture	20
2.4.6 Implementation of Complexity and the Need for Architectural Frameworks	21
2.5 Research Gaps	22
CHAPTER THREE	24
METHODOLOGY.....	24

3.1 Introduction.....	24
3.2 Research Design	24
3.3 Research Methods.....	25
3.3.1 Architectural Framework Design	25
3.4 Architecture Evaluation Methodology.....	31
3.4.1 Evaluation by Technical Review	31
3.4.2 Evaluation by Prototyping.....	34
3.5 Validity and Reliability.....	47
CHAPTER FOUR.....	48
RESULTS AND DISCUSSION	48
4.1. Introduction	48
4.2. Technical Review Result.....	48
4.2.1. Reviewer Profile.....	48
4.2.2. Perceived Benefits of the Proposed Architecture.....	50
4.2.3. Challenges Introduced by the Proposed Architecture	51
4.2.4. Architecture Adoption Preference.....	53
4.3. Prototype Evaluation Results	54
4.3.1. Overview	54
4.3.2. First Contentful Paint (FCP)	55

4.3.3.	Speed Index (SI).....	56
4.3.4.	Total Blocking Time (TBT).....	57
4.3.5.	Largest Contentful Paint (LCP)	58
4.3.6.	Cumulative Layout Shift (CLS) - UX Consistency	58
4.3.7.	Overall Performance Score	59
4.4.	Discussion	60
4.4.1.	Addressing UX Inconsistency	60
4.4.2.	Addressing Authentication and Authorization Vulnerabilities	60
4.4.3.	Addressing the Absence of Implementation Guidance	61
4.4.4.	Performance Trade-offs and Scalability.....	61
4.5.	Summary	61
CHAPTER FIVE.....		64
CONCLUSION, CONTRIBUTIONS, AND FUTURE WORK		64
5.1	Introduction.....	64
5.3	Review of Research Objectives and Findings.....	64
5.3.1	Review of First Objective	64
5.3.2	Review of Second Objective.....	65
5.3.3	Review of Third Objective.....	66
5.3.4	Review of Fourth Objective	67

4.5	Conclusions	68
4.6	Recommendations	69
4.6.1	Policy Recommendations	70
4.6.2	Industry and Practitioner Recommendations	71
4.7	Research Contribution.....	72
4.8	Future Research.....	73
REFERENCES		75

LIST OF TABLES

Table 4.1: Google Lighthouse Performance Metrics.....	54
--	----

LIST OF FIGURES

Figure 2.1: Monolithic Design (Geers M.2020)	9
Figure 2.2: Monolithic frontend and Microservices Backend (Geers, 2019).	11
Figure 3.1 Hut Framework Analogy (Ismail et al., 2023)	26
Figure 3.2: Proposed Architectural Framework	31
Figure 3.3: Prototype Components Diagram	37
Figure 3.4: Module Definition using Single-spa Library	38
Figure 3.5: Loading Micro-Apps into the Launcher Micro-App	40
Figure 3.6: Micro-App Mapping in Launcher App	42
Figure 4.1: Summary of Technical Reviewer Area of Specialization	49
Figure 4.2: Level of Experience of Reviewers.	49
Figure 4.3: Feedback on the Benefits of Architecture	50
Figure 4.4: Feedback on Challenges Introduced by Architecture	51
Figure 4.5: Feedback on Micro Frontend Adoption	53
Figure 4.6: FCP Metric Comparison	55
Figure 4.7: SI Metric Comparison	56
Figure 4.8: TBT Metric Comparison	57
Figure 4.9: LCP Metric Comparison	58
Figure 4. 10: Overall Score Comparison	59

ABBREVIATIONS AND ACRONYMS

MSA	Micro Services Architecture
UX	User Experience
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
HTTP	Hypertext Transfer Protocol
CSS	Cascading Style Sheets
MVC	Model View Controller
CSRF	Cross Site Request Forgery
CORS	Cross Origin Resource Sharing
API	Application Programming Interface
REST	Representational State Transfer

ABSTRACT

Web applications are a critical part of any enterprise business system. Recently, we have observed maturity in technologies that enable the separation of frontend and backend. While backend systems have largely evolved toward microservices architectures, frontend development has predominantly remained monolithic. Micro Frontends have emerged as a solution to the typical monolithic frontend. However, there is no predominant approach to implementation that satisfies the different practical requirements of a state-of-the-art web application. The goal of this work is to develop an architectural framework for implementing micro frontends to address challenges experienced in current implementations, such as inconsistent user experience, managing security and authorization among different micro frontends, and handling complexity in development. To achieve this, the study reviews both horizontal and vertical decomposition approaches for Micro Frontend design, examines the challenges associated with each method, and draws insights from existing implementations and practitioner literature. Based on this analysis, a coherent architectural framework is developed, encompassing team organization, domain decomposition, composition strategies, testing, and deployment design. In conclusion, the proposed architectural framework provides a formalized approach that reduces adoption risks for organizations considering Micro Frontend Architecture by making key architectural decisions explicit and traceable. The architectural framework is intended as a practical reference guide rather than a rigid prescription.

CHAPTER ONE

INTRODUCTION

The opening chapter of this study establishes the intellectual and practical foundation upon which the entire research is built. It situates the study within the broader context of software engineering and web application development, articulates the specific problem that motivates the research, and defines the boundaries, purpose, and structure within which the investigation is conducted. The chapter outlines the historical and technological forces that have shaped modern web application complexity, identifies the architectural gap that this study addresses, and concludes with a clear statement of the objectives, scope, justification, and organizational structure of the research. Readers of this chapter will gain a complete understanding of why the study was conducted, what it sets out to achieve, the boundaries within which it operates, and how the subsequent chapters are structured to deliver on the objectives.

1.1 Background

New trends in software design emerge continually and evolve as a result of users' demand for interactive, rich, and dynamic experiences on various software system platforms. These demands can be satisfied by building applications that adhere to evolutionary architecture (Parson et al., 2022). The evolutionary architectural style supports guided, incremental changes without affecting the quality of the software system. Microservices Architecture (MSA) is an example of such architecture. It is an approach to developing an application as a suite of fine-grained services, each running independently and communicating with lightweight mechanisms, often HTTP resource APIs. This style allows for designing flexible services with agile code bases that can be deployed independently (Lewis & Fowler, 2019). This architecture has provided significant benefits such as solid module boundaries, smaller codebases, independent deployment, and diversification of technologies for backend development of web applications (Lewis & Fowler, 2019). Consequently, we now have an architecture where the front end is implemented as a monolith that sits on top of the microservices backend, hence diminishing the highlighted benefits (Cam, 2019). The monolithic frontend often results in a bulky codebase, making it challenging to scale

the development and deployment especially when multiple teams work on the same product (Lewis & Fowler, 2019). Even minor changes to one component will require rebuilding the entire frontend application, redeployment, and, depending on the technology in use, sometimes restarting. Large codebases result in significant time spent by developers merging code due to code conflicts (Cam, 2019).

Microservices have decoupled the backend, while the frontends have remained an entangled single-page application (Michael, 2021). The complex frontend affects the web application's ability to scale and support the system's evolution (Lewis & Fowler, 2019). This complexity has made it difficult to extend the benefits of microservices to the frontend (Lewis & Fowler, 2019). This development has resulted in a flexible and loosely coupled backend of large complex systems. Microservices have decoupled the backend, while the frontends have remained an entangled single-page application (Michael, 2021). The complex frontend affects the web application's ability to scale and support the system's evolution (Lewis & Fowler, 2019). This complexity has left few benefits of Microservices to extend to the frontend. Micro-Frontends try to address this challenge by breaking the frontend into smaller independent units, increasing team independence, and reducing the overall complexity. The application of Micro-Frontends has confirmed the expected benefits, like the benefits that Microservices offer to the backend implementations (Severi et al., 2021). This research presents a framework for implementing Micro-Frontends that addresses some of the challenges experienced in early implementations.

1.2 Problem Statement

First introduced at the Thoughtworks Technology Radar in 2016 (Geers, 2019), Micro Frontends extend the Microservices concept to the frontend. While the approach has delivered notable benefits such as cross-functional team scalability and reduced frontend complexity (Severi et al., 2021), It has also introduced a set of challenges.

User Experience (UX) Inconsistency is a primary concern, as users expect a seamless experience regardless of the underlying technologies. The isolation inherent to Micro Frontends amplifies this risk (Oner, 2019). Composition strategies such as client-side

reverse proxies disrupt navigation between modules. The use of CSS per module leads to duplication and to inconsistent design standards (Severi et al., 2021).

Security and access control present another significant challenge. Unlike Microservices, Micro Frontends must appear as a unified application to end users, yet each runs independently on its own port with a publicly accessible URL (Lahiru , 2022). Traditional session-based authentication models are ill-suited to this environment, where multiple independent modules share the same browser session. Further complexity arises from both horizontal and vertical split architectures, where each team independently manages authentication and authorization logic, leading to an inconsistent security model (Severi et al., 2021).

Implementation complexity is also a recurring obstacle, with developers often spending more time designing architecture than building features (Pavlenko et al., 2020). Effective decomposition of an application into well-defined modules requires careful prior analysis (Severi et al., 2021). Without a clear strategy, teams risk creating isolated "islands of knowledge," resulting in duplicated efforts.

To address these three core issues- UX inconsistency, security model, and implementation clarity— this research proposes a comprehensive architectural framework for Micro Frontend development.

1.3 Objectives

1.3.1 Main Objectives

The primary objective of this study is to develop and evaluate an architectural framework that guides the implementation of Micro Frontend architecture.

1.3.2 Specific objectives

By the end of this research, the study should be able to:

- i. Analyze challenges experienced by current implementations of Micro Frontend systems.

- ii. Develop an architectural framework for implementing Micro Frontend systems that will address the identified challenges.
- iii. Evaluate the technical feasibility of the developed architectural framework in handling the identified architectural challenges.
- iv. Develop a prototype based on the proposed architectural framework.

1.3.3 Research Questions

In line with the above objective, the following research questions arise:

- i. What challenges are experienced by current implementations of Micro Frontend systems?
- ii. What architectural framework can be designed for implementing Micro Frontend systems while addressing the identified challenges?
- iii. What design artifacts are encompassed and addressed by the architectural framework?
- iv. How can a prototype be constructed to demonstrate and validate the architectural framework?

1.4 Scope

The scope of this study is to develop and evaluate a Micro Frontend architectural framework where frontend complexity presents a recognized engineering challenge. While the study assumes a Microservices backend accessed through an API gateway as the underlying infrastructure, the scope of the proposed framework is confined entirely to the frontend architecture; the design and implementation of backend services are treated as given and fall outside the boundary of investigation. The review is structured around quality concerns derived from the Micro Frontend adoption literature and grounded in the artifact evaluation framework of ISO/IEC/IEEE 42010:2022. Empirical performance benchmarking, load testing, and longitudinal observation of the framework in production environments are outside the scope of this study and are identified as potential directions for future research.

1.5 Justification

The justification for this study is rooted in the growing complexity of modern web applications and the absence of a structured architectural framework that addresses this complexity in a principled, reusable, and comprehensive manner on the frontend. As web applications expand in scale and functional breadth across sophisticated domains, monolithic frontend architectures become progressively difficult to develop, maintain, and evolve, yet the industry's response to equivalent complexity on the backend, Microservices architecture, has not been systematically extended to the frontend layer. The absence of a guiding framework means that teams adopting Micro Frontends independently arrive at inconsistent and often incomplete solutions to shared problems, particularly around authentication, consistent user experience, and cross-application design governance. This study is therefore justified as a direct and targeted response to a documented gap: the need for a structured Micro Frontend framework that addresses both functional and non-functional requirements, integrates security from the outset of the development lifecycle, and provides a reusable architectural foundation that teams can adopt with confidence.

1.6 Structure

This thesis is structured into five chapters, each representing a distinct and progressive stage of the research. Chapter 1 introduces the study by establishing the research context, articulating the problem statement, defining the research objectives and guiding questions, and delineating the scope and justification of the investigation. Chapter 2 presents the literature review, examining the theoretical foundations underpinning the research, including the principles of software architecture, separation of concerns, and the evolution of Microservices and Micro Frontend patterns. This chapter critically surveys the state of the art, presenting and synthesizing existing scholarly and practitioner literature to identify the challenges associated with current Micro Frontend implementations and establish the theoretical gap that motivates the proposed framework. Chapter 3 describes the research methodology, detailing the design science research approach adopted, the rationale for the framework's component structure, and the systematic process by which the proposed Micro

Frontend architectural framework was designed and developed, including the implementation of a prototype that gives practical expression to the framework's design artifacts. Chapter 4 presents the results and evaluation of the study, reporting the findings of the technical architecture review conducted with industry practitioners, analyzing the prototype's performance against the identified architectural challenges, and discussing the implications of the findings in relation to the research objectives. Chapter 5 concludes the thesis by outlining the outcomes of the research, reflecting on the theoretical and practical contributions of the study, acknowledging its limitations, and proposing directions for future research in the design, adoption, and evolution of Micro Frontend architectural frameworks.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The rapid evolution of web application development has fundamentally transformed the way organizations design, build, and maintain software systems. The increasing complexity of user-facing applications, combined with growing demands for scalability, resilience, and development agility, has prompted a continuous re-evaluation of the architectural principles governing both backend and frontend systems. The need to support scalable software systems, reduce development time, and maintain consistent user experience has made architectural decision-making one of the most critical aspects of modern software engineering.

This chapter presents a comprehensive review of existing literature relevant to the research, establishing the theoretical and conceptual foundation upon which the proposed architectural framework is built. The review is structured around three interconnected themes. First, we look at web application development, with a particular focus on monolithic architecture and its limitations. The second explores the emergence of Microservices architecture as a response to the limitations of centralized architecture, reviewing its principles, benefits, and contemporary challenges within backend systems. Lastly, we review Micro Frontend architecture, focusing on its origins, implementations, identified challenges, and the existing gaps in research that this study seeks to address.

This review serves to contextualize the research problem, validate the significance of the identified challenges, and justify the need for the proposed architectural framework.

2.2 Layered Architecture and the Monolithic Model

Modern web applications are designed to serve multiple users and support complex organizational workflows. The development of these systems brought with it a set of unique engineering challenges, particularly around scalability, maintainability, and the

management of increasingly large and complex codebases. These challenges necessitated the adoption of structured architectural patterns to bring order to the growing complexity of enterprise software development. The most widely adopted architectural framework was the layered architecture, which organizes software into distinct horizontal layers, each responsible for a specific concern within the system (Parson et al., 2022). In a typical three-tier architecture, these layers consist of the presentation layer, responsible for rendering the user interface; the business logic layer, responsible for processing rules and workflows; and the data persistence layer, responsible for managing database interactions. This separation of concerns was intended to improve modularity, testability, and maintainability, allowing developers to work on individual layers without directly affecting others. In practice, however, the boundaries between layers often became blurred over time, as the pressures of rapid feature development and evolving business requirements led to the gradual erosion of architectural boundaries. The result is a tightly coupled, monolithic codebase in which changes to one layer inevitably ripple across the entire system.(Natesan, 2019).

A monolithic architecture, by definition, bundles the entire application, including all its layers and modules, into a single deployable unit that is hosted and executed on a server (Parson et al., 2022). While this approach is straightforward to develop and deploy in the early stages of a project, it becomes increasingly problematic as the application grows. As the code base expands, the deployable unit grows proportionally, and the tight coupling between components means that even minor changes require a full redeployment of the entire system. This not only increases the risk associated with each deployment but also significantly extends the time required to deliver new features or bug fixes to end users.

Parson, Nealford, and Rabbecca (2022) define the concept of an architectural quantum as an independently deployable component with high functional cohesion, incorporating all the structural elements required for the system to function appropriately. In monolithic architecture, the entire application constitutes a single architectural quantum; everything is deeply coupled, and the development team is therefore constrained to deploy the application as a whole, regardless of which component or feature has changed. This fundamentally limits the organization's ability

to adopt modern software delivery practices such as continuous integration, continuous delivery, and independent team autonomy.

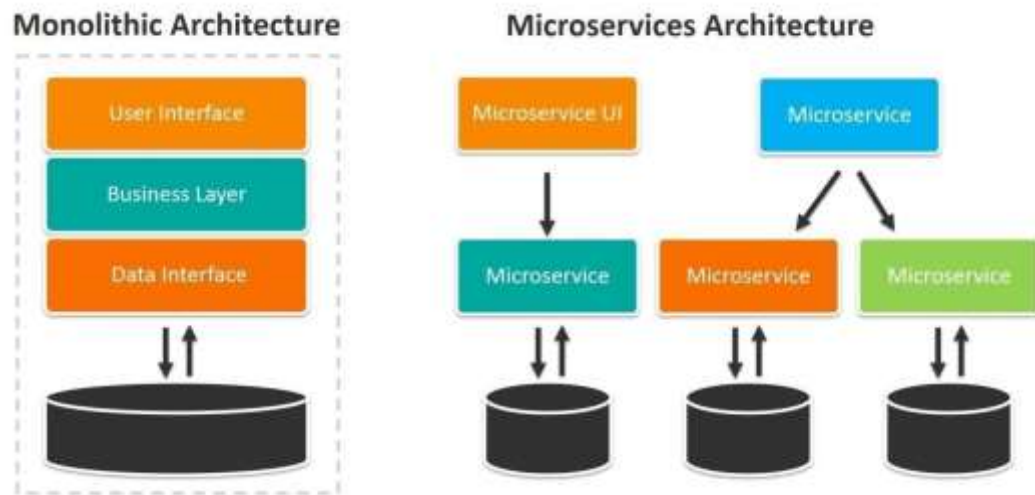


Figure 2.1: Monolithic Design (Geers M.2020)

The limitations of monolithic architecture have been extensively documented in both academic and industry literature. Several core challenges have been consistently identified.

First, scalability in monolithic systems is inherently constrained. Since all application components are bundled together, scaling one part of the system requires scaling the entire application, even if only a single module is experiencing high load. This leads to inefficient resource utilization and increased operational costs, particularly in cloud environments where resources are provisioned dynamically (Newman, 2021).

Secondly, monolithic architecture introduces substantial barriers to development agility. Large codebases composed of tightly coupled components become increasingly difficult to understand, modify, and test over time. As more developers contribute to the same codebase, merge conflicts, dependencies, and coordination overhead grow, slowing the pace of development and increasing the likelihood of introducing defects. This phenomenon, often referred to as the "big ball of mud," is a well-recognized anti-pattern in software architecture (Parson et al., 2022). Literature describing systems that have grown organically without clear structural boundaries,

resulting in code that is difficult to reason about and nearly impossible to refactor safely.

Third, monolithic architecture imposes significant constraints on technology evolution. Since all components of the application share the same runtime environment and technology stack, organizations are locked into a single set of languages, frameworks, and tools. As newer, more capable technologies emerge, migrating a monolithic system to adopt them is a costly, high-risk undertaking that often requires a complete system rewrite rather than an incremental evolution (Fowler & Lewis, 2014).

Finally, the concept of evolutionary software architecture, as articulated by Parson, Nealford, and Rabbecca (2022), highlights the need for architectures that are designed to accommodate change rather than resist it. Software systems are inherently dynamic, and the requirements they serve evolve continuously in response to changing business needs, user expectations, and technological advances. A sound architectural framework must therefore support incremental change, promote modularity, and enable the decoupling of features along business capability boundaries. Monolithic architecture, as discussed, is fundamentally misaligned with these principles, making it unsuitable as a long-term foundation for modern enterprise web application development

2.3 Micro-Services Architecture (MSA)

The recognition of monolithic architecture's limitations prompted the software engineering community to explore alternative approaches that could deliver greater modularity, scalability, and development independence. One of the most significant architectural innovations to emerge from this exploration is Microservices architecture (MSA), which has rapidly become one of the dominant paradigms for backend software development in large-scale enterprise systems (Fowler & Lewis, 2014). Microservices architecture has emerged as a variant of service-oriented architecture (SOA), an earlier paradigm that advocated for the decomposition of software systems into interoperable services (Pooyan & James, 2018). While SOA provided a conceptual foundation, it was often implemented using heavyweight enterprise service

bus (ESB) middleware and complex, standardized communication protocols that reintroduced many of the coupling and complexity problems it sought to solve. Microservices architecture addressed these limitations by advocating for lightweight communication mechanisms, decentralized governance, and a strong emphasis on independent deployability and team autonomy (Fowler & Lewis, 2014)

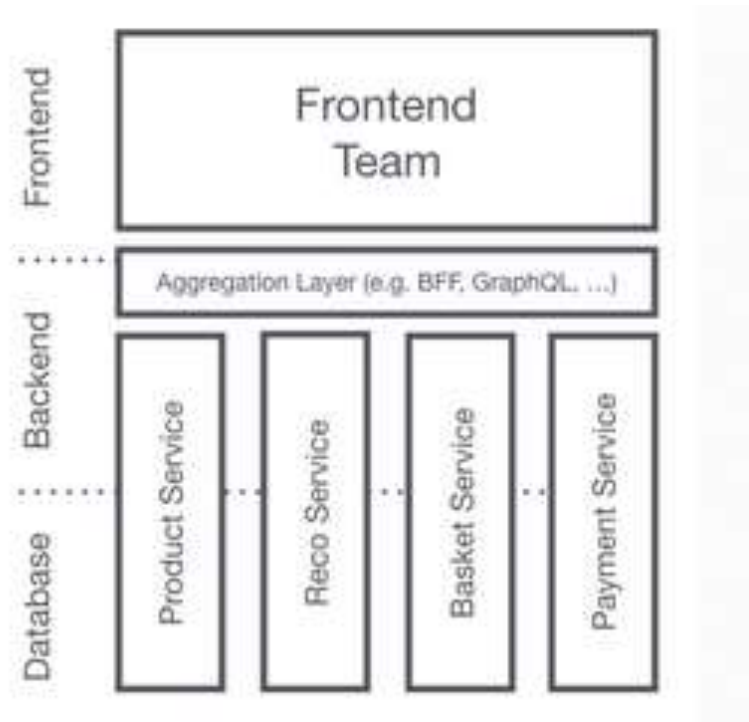


Figure 2.2: Monolithic frontend and Microservices Backend (Geers, 2019).

2.3.1 Core Principles and Characteristics

The Microservices architectural style is characterized by a set of core principles that collectively define its approach to system design and organization. Understanding these principles is essential for appreciating both the benefits the architecture delivers and the challenges it introduces, particularly as they relate to the extension of these principles to the frontend layer through Micro Frontends.

The principle of single responsibility dictates that each Microservice should be responsible for one and only one well-defined business capability, making it easier to understand, maintain, and evolve independently (Newman, 2021). This aligns with the

broader software engineering principle of separation of concerns and ensures that changes within one service area do not unnecessarily affect others.

Independent deployability is one of the most operationally significant characteristics of microservices. Unlike monolithic systems, where a change to any component necessitates a full redeployment of the entire application, individual Microservices can be updated, redeployed, and scaled without affecting the rest of the system (Newman, 2021). This capability is foundational to modern continuous delivery practices, enabling organizations to release new features and apply fixes to production environments with greater frequency, lower risk, and reduced operational overhead.

Decentralized governance is another defining characteristic, allowing individual teams to make their own technology choices such as programming languages, frameworks, and databases based on what is best suited to their specific service's requirements (Fowler & Lewis, 2014). This technology heterogeneity is a significant departure from the enforced uniformity of monolithic architecture and enables organizations to adopt the most appropriate tool for each specific problem domain.

Finally, the principle of design for failure acknowledges that in a distributed system, individual components will inevitably fail, and the system must be designed to handle these failures gracefully without cascading into a system-wide outage (Newman, 2021). This fault isolation is a significant advantage over monolithic systems, where a defect in one module can potentially bring down the entire application.

2.3.2 Benefits of Microservices Architecture

The adoption of Microservices architecture has delivered substantial and well-documented benefits to the backend development processes of organizations across a wide range of industries. Hassan and Bahsoon (2016) highlight significant improvements in development agility and operational resilience that have resulted from decomposing monolithic backends into independently managed services. The ability to develop, test, and deploy services autonomously has enabled organizations to organize development work around cross-functional product teams, each owning a

specific business domain end-to-end, rather than along horizontal technology layer boundaries.

The scalability advantages of Microservices are particularly significant in cloud-native environments. Because individual services can be scaled independently, organizations can allocate computational resources precisely where they are needed, rather than scaling the entire application uniformly (Newman, 2021). This targeted scalability not only improves system performance under variable load conditions but also reduces infrastructure costs by avoiding the over-provisioning associated with monolithic deployments.

The fault tolerance benefits of Microservices are equally compelling. In a well-designed Microservices system, the failure of one service does not necessarily propagate to others, provided appropriate resilience patterns such as circuit breakers, timeouts, and fallbacks are implemented (Fowler & Lewis, 2014). This stands in sharp contrast to monolithic architecture, where a defect in one area of the codebase can cause the entire application to become unavailable, with potentially severe consequences for the organization and its users.

2.3.3 Challenges and Limitations of Micro-Services Architecture

Despite its many advantages, Microservices architecture is not without its challenges, and these have been the subject of considerable academic and industry scrutiny. The transition from monolithic to a Microservices architecture introduces a new category of distributed systems complexity, which manifests across several dimensions.

Network communication between services introduces latency and potential points of failure that do not exist within a monolithic application's in-process function calls. Managing this inter-service communication reliably, particularly in systems with many services and complex dependency graphs, requires sophisticated infrastructure and careful architectural design (Newman, 2021). Additionally, ensuring data consistency across multiple independently managed service databases requires careful consideration of eventual consistency patterns and distributed transaction management strategies.

Operational complexity is another significant concern. Managing the deployment, monitoring, logging, and debugging of many independent services requires substantially more sophisticated tooling and operational practices than are needed for a monolithic application (Pooyan & James, 2018). Organizations adopting Microservices must invest in container orchestration platforms, centralized logging and monitoring infrastructure, and distributed tracing capabilities to maintain visibility and control over their distributed systems.

Despite these challenges, the benefits of Microservices have driven their widespread adoption as the dominant paradigm for backend enterprise software development. However, as (Cam, 2019) crucially observes, while backends have been successfully decomposed through Microservices, the frontend layer has, in many cases, remained a monolith, negating much of the development agility that the Microservices backend was intended to enable. This observation forms the central motivation for the emergence of Micro Frontend architecture.

2.4 Micro Frontend Architecture

2.4.1 Conceptual Origins and Definition

The concept of Micro Frontends emerged as a direct response to the persistent challenge of monolithic frontends in otherwise distributed systems. The term was first introduced at the Thoughtworks Technology Radar in late 2016 (Geers, 2019), marking a formal recognition within the software development community that the principles driving the success of Microservices in the backend could and should be applied to the frontend layer as well. At its core, Micro Frontend architecture advocates for the decomposition of a complex, monolithic frontend application into smaller, independently developable, testable, and deployable units, each owned by a distinct team and aligned to a specific business domain (Geers, 2019).

The fundamental motivation behind Micro Frontends is the desire to extend the team autonomy and operational benefits of Microservices across the full stack of a web application. (Betts & Humble, 2020), articulate this intent clearly, noting that by splitting a complex frontend system into smaller, more manageable pieces, teams can develop and release new features and functionality with greater speed and

independence, without being constrained by the coordination overhead inherent in a shared monolithic frontend codebase.

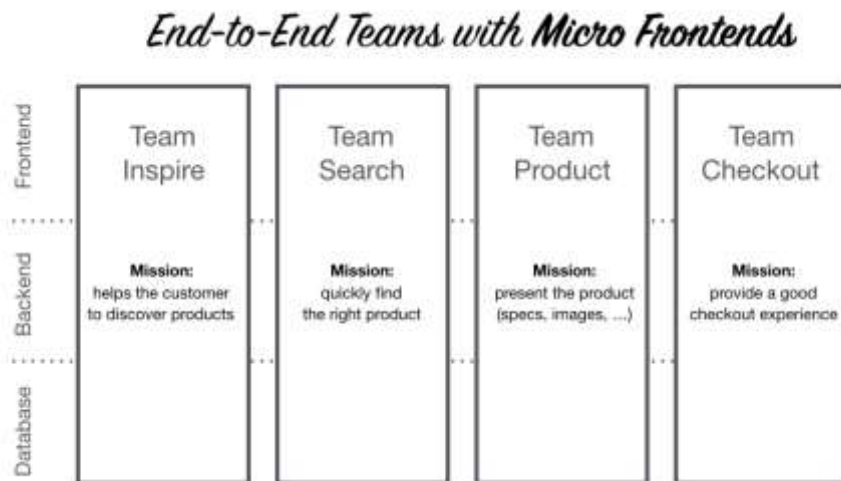


Figure 2.3: Micro Frontends Architecture (Geers, 2019)

2.4.2 Architectural Patterns and Composition Strategies

One of the most significant aspects of Micro Frontend architecture is the variety of approaches available for composing individually developed frontend modules into a cohesive, unified application experience. The selection of an appropriate composition strategy has far reaching implications for system performance, team independence, user experience consistency, and operational complexity.

At the broadest level, Micro Frontend composition strategies can be categorized along two: The timing of composition (build-time versus run-time) and the location of composition (client-side versus server-side). Build-time integration, while simpler to implement, reintroduces many of the coupling and coordination challenges of monolithic frontends, as all teams must align on release cycles and the integrated package must be rebuilt and redeployed whenever any individual module changes (Geers, 2019). In run-time integration strategies, individual Micro Frontends are composed dynamically at the time of the user's request and are generally preferred in production environments seeking genuine team independence (Geers , 2019).

Client-side composition, one of the most common run-time integration approaches, involves assembling individual Micro Frontend modules within the user's browser, typically using JavaScript-based orchestration mechanisms such as Web Components, Module Federation, or iframes (Geers, 2019). While this approach offers strong runtime independence and enables teams to deploy their modules without coordinating with others, it introduces notable challenges. Severi, Luca, and Davide (2021) observe that the use of reverse proxies in client-side composition can result in broken navigation experiences when transitioning between Micro Frontends, undermining the perception of a unified application.

Server-side composition, by contrast, assembles the individual Micro Frontend modules on the server before delivering the composed page to the client. This approach can offer improved initial page load performance and better search engine optimization (SEO), but it introduces greater coupling between teams at the infrastructure level and can present challenges for delivering highly interactive, application-like user experiences (Geers, 2019).

The vertical split and horizontal split represent two dominant organizational approaches to dividing a frontend application into Micro Frontends. In a vertical split, each Micro Frontend encompasses the full user interface for a specific business domain or user journey. In a horizontal split, different layers or regions of the same page are owned by different teams. Each approach carries distinct implications for team organization, communication overhead, and the complexity of composition, and neither has emerged as universally superior for all contexts (Severi, Luca, & Davide, 2021).

2.4.3 Review of Existing Micro Frontend Implementations

A review of the existing literature reveals a growing body of practical work exploring the application of Micro Frontend architecture across a range of contexts. While these contributions have collectively advanced the understanding of architecture's potential and limitations, they also reveal significant gaps that the present research seeks to address.

(Caifang et al., 2019), described the development of a content management system (CMS) using Micro Frontend architecture, employing the Mooa Framework (Huang, 2018). The system achieved a meaningful separation of frontend and backend concerns and reportedly offered a satisfactory user experience. The authors concluded that the Micro Frontend-based CMS facilitated independent team development and testing, thereby supporting continuous integration and deployment practices. However, the implementation was constrained to a single-page application deployment model and was tightly coupled to the Angular 2+ framework, Mooa Framework, substantially reducing the technology agnosticism that defines one of the advantages of Micro Frontend architecture. Furthermore, the work did not address the challenges of UX consistency, security, or cross-team governance, which researchers increasingly recognize as central concerns in Micro Frontend adoption.

(Mena et al., 2019), explored the feasibility of implementing a progressive web application using Micro Frontends in conjunction with a Microservices backend. Their study demonstrated that Micro Frontend architecture could effectively support the dynamic composition of a feature-rich web application. However, the scope of their investigation was primarily oriented toward the developer experience and team-level benefits, with limited attention given to end-user concerns such as UX consistency, performance optimization, or security considerations, limiting the comprehensiveness of their contribution, limiting its applicability as a basis for enterprise-grade adoption.

(Pavlenko et al., 2020), contributed to a practical investigation of Micro Frontend architecture in the context of single-page applications, documenting the architectural decisions made and the technologies employed throughout their implementation. Their findings raised important concerns about the practical feasibility of the Micro Frontend adoption, particularly for smaller or less experienced development teams. Specifically, they reported that a disproportionate amount of development time was consumed by architectural deliberation and tooling configuration, rather than actual feature development, significantly increasing the overall time-to-delivery. Importantly, their work was characterized as a practical discovery exercise rather than the proposal of a reusable and generalizable architectural framework.

Severi, Luca, and Davide (2021) made one of the most comprehensive contributions to the Micro Frontend literature through a multivocal literature review examining the motivations, benefits, and challenges associated with Micro Frontend adoption across organizations. Their findings confirmed that the primary drivers of adoption were the desire for increased team autonomy and a reduction in the overall complexity of large frontend codebases. The benefits observed were broadly analogous to those previously documented for microservices in the backend, including the establishment of cross-functional development teams capable of independently scaling their processes.

However, the review by Severi, Luca, and Davide (2021) also revealed a substantial set of challenges. They categorized these into two distinct groups: Technological issues such as increased application payload size due to duplicated dependencies, significant code duplication, persistent UX inconsistencies, challenges in managing shared dependencies, and difficulty in monitoring and debugging distributed frontend systems. People-related issues: encompassing elevated implementation complexity, governance challenges, the formation of isolated "islands of knowledge," environmental inconsistencies, and heightened deployment risk. Most significantly, they issued an explicit call for the development of implementation frameworks capable of reducing user interface (UI) inconsistency, minimizing disruption to shared design systems, and providing thoroughly designed architectural guidance that practitioners could confidently follow.

Bob (2020) reinforced these concerns, observing that despite the growing volume of industry attention directed at Micro Frontends, no clear or dominant implementation approach has yet emerged. He argued that for architecture to achieve widespread and sustainable adoption, it must demonstrate a clearly superior architectural approach capable of satisfying the diverse and demanding practical requirements of modern enterprise web applications. Bob (2020) also specifically identified unresolved concerns around authentication and authorization across independently deployed Micro Frontends, as well as the persistent difficulty of ensuring UX consistency across modules developed and maintained by different teams.

2.4.4 Security in Micro Frontend Architecture

Security is one of the most critical and least comprehensively addressed dimensions of Micro Frontend architecture in existing literature. The distributed and independently deployable nature of Micro Frontends introduces a unique set of security challenges that cannot be adequately addressed by traditional session-based authentication and authorization models that have historically governed web application security.

In traditional monolithic web applications, authentication and session management are typically handled by a centralized security layer, which issues a session token upon successful user login and manages access control for all subsequent requests within the bounds of that session (Lahiru , 2022). While this approach is straightforward to implement in a tightly coupled system, it is fundamentally ill-suited to the Micro Frontend architecture, in which multiple independently deployable modules coexist within the same browser session, each potentially operating under different security contexts and requiring access to different sets of protected resources.

Lahiru (2022) highlights that individual Micro Frontends, running independently on dedicated ports and exposing unique URLs, are inherently vulnerable to direct access without appropriate authorization controls. Without a carefully designed security architecture, it is possible for users or malicious actors to access individual Micro Frontend modules directly, bypassing the authentication mechanisms of the broader application shell. This vulnerability is particularly acute given that Micro Frontends are often exposed as independently accessible endpoints, unlike Microservices, which are typically shielded behind API gateways and network-level access controls.

The Zero Trust Security model has emerged as a compelling framework for addressing these challenges (Buck et al., 2021). Unlike traditional perimeter-based security models, which grant broad access to authenticated entities within a trusted network boundary, Zero Trust operates on the principle of "never trust, always verify," requiring explicit authentication and authorization for every request and interaction between components, regardless of their origin or network location. This model is particularly well-suited to distributed architectures such as Micro Frontends, where the assumptions of a bounded trust perimeter are fundamentally undermined by the

independently deployable and potentially heterogeneously hosted nature of individual frontend modules.

Further compounding the security challenge is the observation by Severi, Luca, and Davide (2021) that in both horizontal and vertical split architectures, the responsibility for implementing authentication and authorization logic is often distributed across individual team boundaries. This decentralization frequently results in duplicated, inconsistent, and potentially incompatible authentication and authorization logic across different Micro Frontend modules, increasing the attack surface and introducing the risk of security vulnerabilities arising from inconsistent enforcement of access control policies.

2.4.5 User Experience Consistency in Micro Frontend Architecture

User experience (UX) consistency is one of the most extensively discussed yet persistently unresolved challenges in Micro Frontend architecture. Users of web applications have come to expect a coherent, seamless interaction experience regardless of the underlying technologies or team structures that produced the application. In a Micro Frontend environment, where different modules are developed and maintained by autonomous teams, potentially using different technology stacks, design systems, and development tooling, achieving this consistency is a significant and non-trivial challenge (Oner, 2019)

The root causes of UX inconsistency in Micro Frontend systems are multifaceted. At the most fundamental level, independently developed Micro Frontends are likely to exhibit differences in visual design, including typography, color palettes, spacing, and interactive element styling, unless all teams adhere rigorously to a shared design system or component library. Maintaining such adherence across autonomous teams with independent deployment cycles is an ongoing governance challenge that requires both technical mechanisms and organizational discipline (Severi et al., 2021)

Beyond visual consistency, behavioral and navigational consistency presents an additional layer of complexity. As users navigate between different Micro Frontend modules, they may encounter differences in interaction patterns, loading behaviors,

error handling approaches, and transition animations that break the perception of a unified application. Severi, Luca, and Davide (2021) specifically note that the use of reverse proxies in client-side composition strategies can result in broken navigation experiences when switching between Micro Frontends, further undermining the seamlessness of the user experience.

The management of shared CSS presents a particularly illustrative example of the trade-offs inherent in achieving UX consistency within Micro Frontend systems. The use of a globally shared CSS stylesheet across all Micro Frontends can help enforce visual consistency, but it introduces a form of coupling between independently deployed modules. Conversely, maintaining separate CSS per Micro Frontend avoids this coupling but leads to significant code duplication and makes it difficult to enforce consistent design standards (Severi, Luca, & Davide, 2021). This tension between consistency and independence is a recurring theme in the Micro Frontend literature and highlights the need for an architectural approach that can reconcile these competing concerns.

2.4.6 Implementation of Complexity and the Need for Architectural Frameworks

One of the most consistently reported challenges in the Micro Frontend literature is the significant implementation complexity that the architecture introduces, particularly in the absence of clear, prescriptive guidance on how to structure and approach the development process. Pavlenko, Nursultan, and Mazzara (2020) reported that development teams frequently spend a disproportionate amount of time on architectural deliberation and tooling decisions rather than on productive feature development, significantly extending the overall time to delivery and increasing project costs.

This challenge is compounded by the relative novelty of Micro Frontend architecture as a formal discipline. Unlike Microservices, which benefit from a mature ecosystem of design patterns, reference architectures, frameworks, and tooling, Micro Frontends lack an equivalent body of standardized and widely validated implementation guidance (Severi et al., 2021). Development teams adopting Micro Frontends must

therefore navigate a landscape of disparate, context-specific experiments and case studies, extracting and adapting guidance where possible but frequently having to make fundamental architectural decisions without the benefit of established best practices.

Severi, Luca, and Davide (2021) further identified the phenomenon of "islands of knowledge" as a significant organizational risk in Micro Frontend environments. When autonomous teams develop their Micro Frontends in isolated repositories with limited visibility into the work of other teams, the result is often duplicated effort, inconsistent technical approaches, and the accumulation of siloed expertise. This not only reduces overall development efficiency but also increases the risk of architectural drift and gradual divergence of individual Micro Frontend implementations from the broader architectural principles of the application. This compounds the challenges of integration, maintenance, and UX consistency over time.

The absence of a clear, comprehensive architectural framework that can guide implementation decisions, promote cross-team alignment, and provide tested solutions to the known challenges of Micro Frontend development represents one of the most significant gaps in the existing body of research. It is this gap that the present study directly seeks to address by proposing a structured and reusable architectural framework.

2.5 Research Gaps

The literature reviewed in this chapter charts a clear trajectory in the evolution of web application architecture, from the layered monolithic systems that dominated the early era of enterprise software development, through the transformative adoption of Microservices architecture in the backend, to the emerging and rapidly evolving paradigm of Micro Frontend architecture. Each stage of this evolution has been driven by the limitations of the preceding paradigm and motivated by the desire to achieve greater scalability, development agility, and organizational autonomy.

The body of work surrounding Micro Frontend architecture, while growing, reveals a set of persistent and interrelated challenges that have not yet been comprehensively

addressed by existing research and frameworks. First, the challenge of UX inconsistency, arising from the independent development and deployment of frontend modules by autonomous teams, remains an open problem. Second, the challenge of authentication and authorization in distributed frontend environments has received insufficient attention, with no widely adopted security model specifically designed for the unique characteristics of Micro Frontend systems. Thirdly, the absence of a clear, prescriptive, and technology-agnostic implementation framework continues to impose significant complexity on development teams adopting Micro Frontends, consuming time and resources that could be more productively directed towards feature development.

These three gaps are closely interconnected and collectively define the problem space that this research addresses. The existing literature largely comprises isolated practical experiments, multivocal reviews of current implementations, and technology-specific case studies, rather than the proposition of comprehensive, reusable architectural frameworks that directly tackle these challenges in an integrated manner. This absence of a holistic architectural framework, validated through both design evaluation and prototypical implementation, represents the central gap that the present research seeks to fill.

The following chapter presents the research methodology employed to address these identified gaps, outlining the approach taken to design, develop, and evaluate the proposed architectural framework for Micro Frontend systems.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

This chapter presents and justifies the research methodology employed in this study. It outlines the philosophical stance underpinning the research, the research design selected to guide the overall inquiry, and the specific methods used to collect and analyze data. The chapter also discusses the strategies adopted to ensure the validity and reliability of the research. The overarching aim of this research is to develop and evaluate an architectural framework for implementing Micro Frontend systems that address the challenges of UX inconsistency, security vulnerabilities, and the absence of clear implementation guidance. Achieving this aim requires a methodology that supports not only the systematic identification and analysis of existing challenges but also the design, construction, and evaluation of the artefacts that define the architectural framework. The architectural framework proposed in this research, its practical instantiation in a working prototype, and its evaluation against the specific challenges it was designed to address, are discussed. This emphasis on empirical validation aligns well with the research design described in the sections that follow.

3.2 Research Design

The research methodology applied in this study is rooted in the Design Science Research (DSR) paradigm, a well-established and widely adopted approach in applied computing and information systems research, particularly suited to studies that involve the development and evaluation of technological artefacts (Brocke et al., 2020). DSR advances both technological and scientific knowledge through the creation of innovative artefacts designed to address real-world problems and improve the context of their deployment. Central to this paradigm is the production of tangible outputs such as models, frameworks, or prototypes that not only resolve specific practical challenges but also generate meaningful contributions to domain knowledge (Brocke et al., 2020). By integrating theoretical understanding with practical application, DSR provides a rigorous and structured foundation for problem-solving and innovation.

In this study, the proposed solution is shaped by the application of established architectural principles and design patterns. The artefacts of the architectural framework are developed through a disciplined design process and subsequently evaluated via expert review alongside prototype implementation and testing. The inclusion of a prototype development and evaluation phase introduces an empirical dimension to the research, enabling practical validation of the proposed framework through the construction and testing of a fully functional software system. This mixed-methods approach, combining qualitative analysis with hands-on artefact construction and evaluation, provides a comprehensive basis for assessing the extent to which the artefact effectively addresses the identified problem.

3.3 Research Methods

3.3.1 Architectural Framework Design

The architectural framework is designed as a set of structured, technology-agnostic guidelines, principles, and design decisions that address the three core challenges identified in the literature: UX inconsistency, authentication and authorization vulnerabilities, and the absence of a clear implementation strategy. A software architectural framework is an integrated collection of components that collaborate to produce a reusable artifact to develop related applications (Fowler, 2019). It represents a solution to problems that arise when developing software within a context. It captures the static and dynamic structure and collaboration in software designs, thus facilitating reuse and reducing complexity (Fowler, 2019). Software engineering processes have always sought ways to reduce time to market, reduce development costs, standardize software development, improve quality, improve reliability, and reduce complexity. Most software systems in the domain tend to follow known or nearly known architecture. Thus, among these methods that are the most important is the reuse of the known architectural frameworks. It's notable, though, that frameworks are usually domain-specific and applicable only to families of applications.(Fowler, 2019). Pavlenko,et reported having experienced much complexity trying to develop an application using Micro Frontends architecture due to spending much time focusing on the architecture and development tools rather than development work, which increased the overall development time.

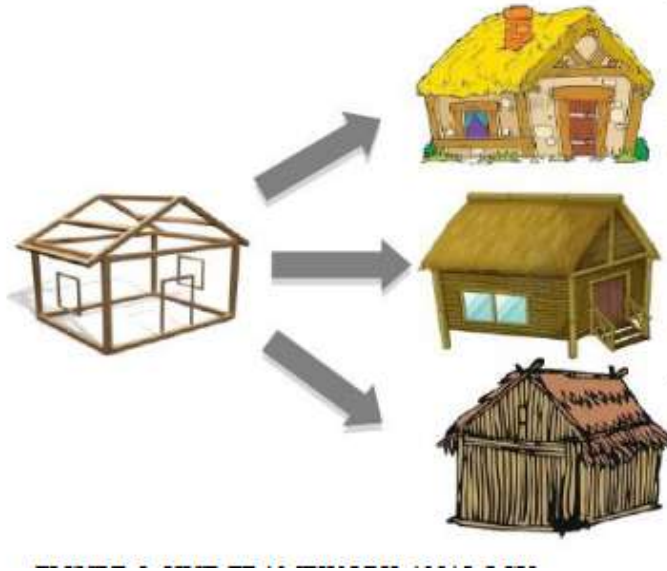


Figure 3.1 Hut Framework Analogy (Ismail et al., 2023)

By defining an architectural framework for Micro Frontend, we will enable the developer to focus on actual work instead of the application's architecture, reducing complexity and providing a standard approach to development. Like the hut framework above, which takes the constructor off the responsibility of designing the shape of the hut, the architectural framework takes off the developer's responsibility of defining the framework and hence can focus on development work. (Anwar, 2023).

3.3.1.1 Methodology for Deriving the Architectural Components

Separation of Concerns as a Foundational Principle

The methodology used to derive the components of the proposed architectural framework is grounded in established software architecture and system design principles, particularly separation of concerns, principle-driven architectural reasoning, and requirements-driven decomposition.

Web applications solve various problems and currently deal with increasing complexity in different sophisticated domains such as financial, medical, and geographical. The need to provide accessible frameworks that reduce this complexity in building applications is paramount. This complexity in software can only be dealt

with by separating modeling concerns in a clear and modular way (Natesan, 2019). The overall goal of the separation of concerns is to establish a well-organized system where each part fulfills a meaningful and intuitive role while maximizing its ability to adapt to change (Natesan, 2019). Mature web application frameworks such as MVC recognize the importance of clear separation of behavior aspects of application and Navigation and interface (Anwar, 2023).

Requirements-Driven Decomposition

The methodology applied a requirements-driven decomposition approach, in which the system structure is derived explicitly from functional and non-functional requirements.(Bass et al., 2021). Functional requirements, which define the business functions that a system must perform (Bigelow, 2020), were mapped to independently developed Business Micro-Apps. This aligns with the principles of Domain-Driven Design, where system modules correspond to bounded business contexts (1962-, 2004). In our architecture, just like Microservices, we propose a separate Micro Frontend application to handle each business functionality; this becomes the first Micro-App in our proposed architecture. We will call this a "Business Micro-App". This will modularize the frontend around business functionality. An application can have multiple Business Micro-Apps depending on the number of functional requirements.

Principle-Driven Architectural Reasoning

Principle-driven architectural reasoning was used to ensure architectural decisions were guided by quality attributes such as scalability, security, usability, and maintainability, rather than ad hoc design choices.(Bass et al., 2021). Non-functional requirements, particularly authentication, authorization, navigation consistency, and user experience, were identified as cross-cutting concerns that require explicit architectural support. Prior studies highlight these aspects as recurring challenges in Micro Frontend implementations. (Bob, 2020; Pavlenko et al., 2020)

To handle Authentication requirements, we propose the second Micro Frontend application called “Authentication Micro-App.” The Micro-App handles login/security-related requirements. To handle application layout, user experience, and ensure a consistent design and navigation, we propose to have the third Micro Frontend application called "Layout/Navigational Micro-App."

Composition Strategy and Integration

To integrate independently developed and deployed Micro Frontends into a cohesive system, the methodology adopted a client-side composition strategy. Composition strategies define how independently owned frontend modules are assembled and coordinated at runtime (Jackson, 2019; Mezzalana, 2021). This led to the introduction of a Launcher Micro-App that serves as the application entry point and acts as an integration shell for loading other Micro Frontends.

Having a separate Micro Frontend application for each function helps reduce complexity, relieving developers of the need to worry about them, thus allowing them to focus on business requirements.

Components of the Proposed Architectural Framework.

The proposed framework will have four main Micro-Apps, as highlighted in the design rationale above. The Micro-Apps can be developed and deployed independently. The final application merges the different components into a single seamless single-page application. To illustrate this, we will assume the backend of the application is also implemented as Microservices.

Authentication Micro-app

The seamless integration of access control is an essential aspect of any modern web development. Role-Based Access Control (RBAC) is a technique for multi-user systems and networks introduced in 1992 by Ferriolo and Kuhn (Chatterjee & Pitroda, 2020). Organizations use RBAC to protect their information resources from unauthorized access. It enables the modeling of the security requirements of today's web applications through complex role hierarchies and restricted access to predefined

users (Chatterjee & Pitroda, 2020). A common problem of securing web applications lies in the late integration of security at the test phase or the end of the development process. It is recommended to integrate security into the life cycle of a web application (Chatterjee & Pitroda, 2020). By incorporating security in our proposed framework, we ensure its integration into the Micro Frontend early in the life cycle. Having a separate Micro Frontend for handling login relieves other Micro Frontends of the responsibility of handling authentication; hence, they focus on other core logic. The user is redirected to this Micro-App when they access a protected resource without authorization. This Micro-App then presents the user with a login form where the user authenticates. After successful authentication, the backend Authentication service generates a JSON Web Token (JWT)(Bucko & Vishi, 2023) that is saved in browser session storage. This gives the other Micro Frontends a standard approach to implementing a zero-trust security model (Lahiru , 2022). The other Micro Frontends that require privileged access implement an interceptor that checks the validity of the current user, and if valid, allows access to the resource or redirects the user to the Authentication Micro Frontend for login.. The token is encrypted with a secret to guard against unauthorized modification and session fixation attacks. To guard against Cross-Site Request Forgery (CSRF) and CORS, the backend API Gateway Microservices also validate the token and check if the user initiating the request is authorized.

The Navigational/Layout Micro-app

UI patterns such as Card-style and grid-based layouts have been around for years and continue looking modern and contemporary due to their adaptability to changing trends. UI designers can implement this pattern in a variety of ways (Bob, 2020). Having a separate Micro Frontend for layout provides a reusable component that is applied to other pages, ensuring consistency in the web application. This separation of concerns also ensures developers working on other Micro Frontends focus on the domain logic instead of developing a new design. This design also makes it possible to have a shared CSS for specific application components, furthering consistency in design. This design helps standardize the principles of design within an application consisting of multiple Micro Frontends and reduces conflicts since it is defined and

shared by the entire team working on different Micro Frontends within the application. Unlike in horizontal split and vertical split architectures, in which different teams are responsible for the entire design for a particular business domain, having this module handle layout enforces standard design across the teams. Authorization can also be implemented on this Micro Frontend to ensure users can view menus from the Micro Frontends they are authorized to view.

Launcher Micro-app

The central role of this module is to load the other Micro-Apps, and present to the user a single application. This is a client-side composition strategy that serves as the entry point that loads other Micro Frontends and initializes the JavaScript application. It integrates all other Micro-Apps into one application that is presented to the user. It serves as the engine of the application. The users access the URL for this Micro-App. The index page of this Micro Frontend loads the layout page.

Business Micro-App

Micro Frontends require a clear analysis to understand how everything will work in integration and how an application can be broken down into modules. This architecture proposes splitting the application based on business functionality within the application. While business functionality could span multiple departments within an organization, the logic would be domiciled within one Micro Frontend for easy maintenance and management. The Business Micro-App defines custom applications that define the business requirement of the web application. Each module that exists in the application can be developed separately and added to the layout of the Micro Frontend. For example, on an Internet banking site, we can have an account balance Micro-App, an account statement Micro-App, a loan application Micro-App, an account opening Micro-App, etc. These Micro-Apps represent the functional business requirements of the application.

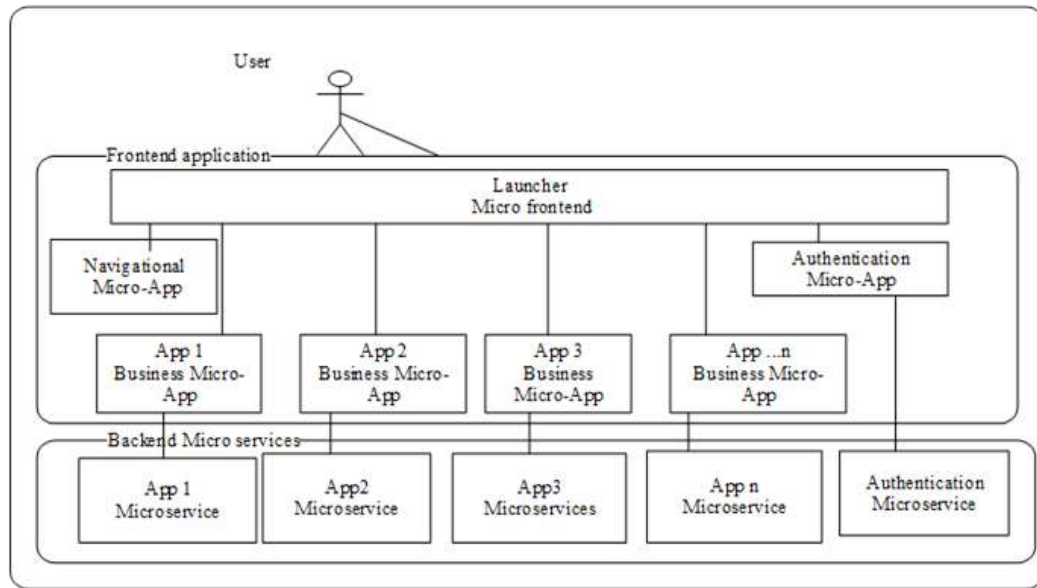


Figure 3.2: Proposed Architectural Framework

3.4 Architecture Evaluation Methodology

3.4.1 Evaluation by Technical Review

The evaluation of the proposed Micro Frontend architectural framework is grounded in the principles established by ISO/IEC/IEEE 42010:2022, the international standard for architecture description. The standard defines architecture evaluation as a structured process of assessing whether an architecture description adequately addresses the concerns of its stakeholders through documented viewpoints and well-defined artifacts (2022). Consistent with this standard, the review methodology employed in this study treats architecture not as an abstract artefact but as a set of documented decisions that must be validated against the concerns of those who have a stake in the system's outcomes.

The evaluation adopts a Technical Architecture Review approach, which applies static analysis techniques grounded in architecture documentation, reviewer experience, and domain expertise (Ågren & Knauss, 2022). Rather than executing the system, this method examines architectural artifacts and documentation through structured walkthroughs with selected stakeholders, a practice directly aligned with the requirement in ISO/IEC/IEEE 42010:2022 that architecture descriptions be assessed

for their ability to satisfy stakeholder concerns (2022). The objective of the review is to build stakeholder confidence that the proposed architecture fulfils its intended concerns, with particular focus on its suitability for supporting agile development within a Micro Frontend context.

3.4.1.1 Stakeholder Identification

ISO/IEC/IEEE 42010:2022 stipulates that an architecture description must identify the stakeholders of a system and their concerns (2022). Consistent with this requirement, three categories of stakeholders were engaged as technical reviewers, each representing a distinct perspective of concern.

The target audience for the technical review comprises the key stakeholders of an architecture, as the same architecture can be reviewed by different stakeholders for different outcomes (Ågren & Knauss, 2022). The following three groups were selected for this study:

- i. **Software Engineers:** Software engineers are the primary users of a software architectural framework. They realize an architectural framework by developing and integrating its components into a functioning software system. Constituting 73% of the reviewers in this study, they represent the most significant stakeholder group in evaluating the practical implementability of the proposed architecture.
- ii. **Software Architects and Analysts:** This group is responsible for analyzing the architecture to ensure it satisfies defined quality attributes and aligns with business requirements and available infrastructure constraints. Their concern is primarily with architectural fitness and conformance to specified quality goals.
- iii. **System Administrators and Support Engineers:** This group accepts completed systems from software engineers and deploys them in production, making them operational and fulfilling business functions. They support and maintain the system post-deployment, representing concerns related to operability, governance, and maintainability concerns that ISO/IEC/IEEE 42010:2022 recognizes as legitimate architectural concerns that must be addressed in any comprehensive evaluation (2022).

3.4.1.2 Artifact Selection

In accordance with ISO/IEC/IEEE 42010:2022's requirement that architecture evaluations be conducted against defined concerns and documented artifacts. The review was structured around artifacts drawn from a multivocal literature review conducted by Severi et al. (2021) on the adoption of Micro Frontend architectures. These artifacts capture the principal motivations for adopting Micro Frontend architecture and its documented shortcomings, and served as the evaluative lens through which the proposed architecture was examined.

The identified benefits of adopting a Micro Frontend architecture, as presented by Severi et al. (2021), include the following:

- Smaller codebase per application module
- Independent deployment of individual Micro Frontend applications
- Improved development scalability across distributed teams
- Reduced risk when deploying incremental changes
- Independent scaling of individual services
- A more effective approach to agile development and product release cycles
- Reduced complexity in frontend development

Conversely, Severi et al. (2021) also identified the following shortcomings associated with the Micro Frontend approach:

- Technological challenges in integration and tooling
- Code duplication across independently developed Micro Frontends
- Increased governance and support overhead in production environments
- Difficulty in monitoring and debugging distributed frontend components

3.4.1.3 Architectural Review Questions

A structured review questionnaire was developed to elicit stakeholder judgements systematically across the identified concerns. The instrument comprised five questions designed to collect both contextual and evaluative data from reviewers.

The first two questions established the professional background and experience level of each reviewer, providing the contextual information necessary to interpret responses appropriately, consistent with ISO/IEC/IEEE 42010:2022's guidance on the role of reviewer expertise in architecture evaluation (2022). The remaining three questions addressed the substantive concerns of the review, as follows:

Q1. What is your area of specialization? This question was motivated by the need to establish the domain expertise of each reviewer, as stipulated by ISO/IEC/IEEE 42010:2022 in the identification of qualified stakeholders (2022).

Q2. What is your level of experience in your field of specialization? This question assessed the depth of expertise brought by each reviewer, enabling appropriate weighting of responses in the analysis.

Q3. What will be the benefits of the proposed architecture? Reviewers were presented with the proposed architecture and asked to provide feedback on the benefits it would present in an organizational or development context.

Q4. What challenges will such architecture introduce in your development and deployment? This question examined the potential difficulties and risks the proposed architecture would introduce, directly addressing the shortcomings identified in the review artifacts.

Q5. Between the proposed architecture, monolithic architecture, and hybrid architecture (Microservices in the backend and monolithic in the frontend), which would you prefer for a development project? This comparative question was designed to gauge the likelihood of adoption of the proposed architecture relative to existing alternatives in web application development.

3.4.2 Evaluation by Prototyping

A prototype is an early sample or model that is used for testing functions or a system and is created to find a design solution. It leads the designer in problem-solving processes (Koskinen & Frens, 2018). The process involving testing, refinement, and completion of designs using prototypes is called prototyping. Prototypes are used to

test if designs achieve their intended performance. Prototyping is also used in architectural design processes. Research prototypes have connections to the theoretical literature of their respective areas of study. The purpose of research prototypes is to articulate and test concepts that respond to questions from theoretical literature at the foundation of a research program rather than product development processes (Koskinen & Frens, 2018). In this research, we develop a prototype to evaluate the applicability of our proposed Micro Frontend architectural framework. We measure its performance and how it addresses challenges encountered in the current implementation of Micro Frontends.

To demonstrate the practical applicability of the proposed architectural framework and provide a basis for empirical evaluation, a working prototype web application is developed in accordance with the framework's principles and guidelines. The prototype serves as a proof-of-concept implementation that instantiates the key architectural decisions of the framework in a realistic software system, enabling the evaluation of the framework's effectiveness in addressing the identified challenges. The development process is documented to maintain traceability between the framework's design decisions and their practical implementation in the prototype, supporting the evaluation of the framework's guidance clarity and completeness. Technology selections for the prototype were made in accordance with the framework's principle of technology agnosticism, selecting widely adopted, industry-standard tools and frameworks that are representative of contemporary Micro Frontend development practice. The prototype includes multiple independently developed and deployed Micro Frontend modules, a shared design system component library, an application shell responsible for orchestrating module composition, and an integrated authentication and authorization based on Zero Trust.

3.4.2.1 Overview of Requirements

The prototype functional and non-functional requirements are defined as follows:

The Frontend

- i. The system shall support privileged access control
- ii. The system should have a superuser role that enables the creation of roles, permissions, and assigns users to roles
- iii. The system shall allow users to authenticate using a username/password
- iv. The system shall enforce the roles and permissions given to users.
- v. The system shall save data created to a database and list the records created.
- vi. The system should have a consistent look and feel across all modules.

The Backend

- i. The system should provide an API for posting and accessing data in the backend database.
- ii. Access to the backend API shall be through a gateway service that authenticates requests.
- iii. The system shall reliably store data.

3.4.2.2 Development View

The prototype is built based on a multi-layered architecture. Notably, this is a three-tier architecture with classical tiers: presentation, business logic, and data. The frontend or presentation layer is implemented as a Micro Frontend based on the described architecture. In this case, five Micro Frontends were developed: the Authentication Micro-App, the Layout/Navigation Micro-App, the Launcher Micro-App, and two business Micro-Apps (Admin Micro-App and Farmer Micro-App). All Micro Frontend applications access the backend business logic through an API Gateway that also implements security for the backend. The API Gateway provides access security to individual Microservices. The data layer has been implemented using MongoDB, and each service maintains its database as is standard in microservice

implementations

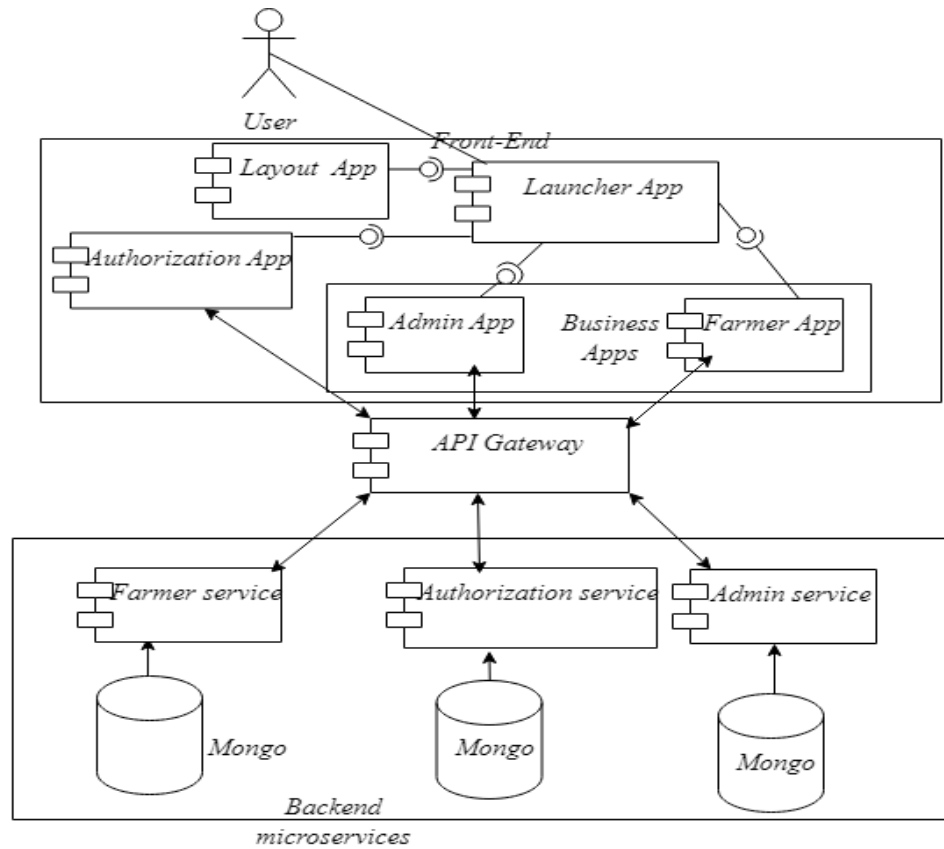


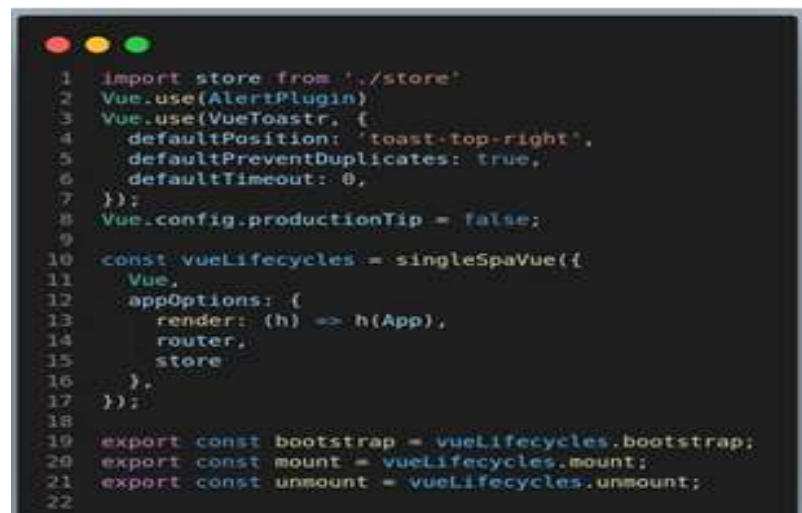
Figure 3.3: Prototype Components Diagram

3.4.2.3 Implementation of the Frontend

In this research, JavaScript was used to develop the prototype since it is one of the popular programming languages for frontend development. The Vue JavaScript library was used to speed up the development process. Vue is a JavaScript library for building user interfaces. It consists of a core library that focuses on the view layer and an ecosystem of supporting libraries that help tackle complexity in large web applications. (You & Hong, 2016).

The Vue.js single-spa library brings together multiple applications into one frontend application. It helps develop an application as a single-spa registered module by controlling its lifecycle; that is, loading the application and mounting it in a browser when the application is activated and unloading it when deactivated (Joel, 2020). The code snippet in Figure 7 shows how each Micro-App is registered as a Single-spa-vue

module. The single spa manages actions associated with the registration of Micro Frontends as modules and controls the mounting and unmounting of Micro Frontend applications into the browser's DOM tree. (Joel, 2020). The module definition code used to register each Micro Frontend application within the Single-spa framework. Single-spa is a JavaScript meta-framework that enables multiple independently developed frontend applications to coexist and operate within a single browser environment. The module definition is the foundational configuration artifact that informs the Launcher Micro-App how to identify, locate, and load each Micro Frontend at runtime. Each entry in this configuration corresponds to one independently deployed Micro-App, and the configuration as a whole gives the Launcher its knowledge of the full application topology.



```
1 import store from './store'
2 Vue.use(AlertPlugin)
3 Vue.use(VueToastr, {
4   defaultPosition: 'toast-top-right',
5   defaultPreventDuplicates: true,
6   defaultTimeout: 0,
7 });
8 Vue.config.productionTip = false;
9
10 const vueLifecycles = singleSpaVue({
11   Vue,
12   appOptions: {
13     render: (h) => h(App),
14     router,
15     store
16   },
17 });
18
19 export const bootstrap = vueLifecycles.bootstrap;
20 export const mount = vueLifecycles.mount;
21 export const unmount = vueLifecycles.unmount;
22
```

Figure 3.4: Module Definition using Single-spa Library

Line 1: imports from the Vuex store from the local store module. The store manages the reactive state of this specific Micro-App, keeping its data layer entirely self-contained and independent of the state of other Micro-Apps .

Line 2: registers the AlertPlugin with the Vue application instance. This plugin provides a global notification mechanism, enabling any component within this Micro-App to trigger user-facing alerts without importing the plugin individually at the component level.

Lines 3–7: register the VueToastr plugin with a configuration object. The `defaultPosition` property places toast notifications in the top-right corner of the screen. The `defaultPreventDuplicates` property ensures that identical notifications are not displayed more than once simultaneously. The `defaultTimeout` of 0 means notifications remain visible until explicitly dismissed by the user, rather than auto-closing after a time interval. These settings are configured here at the application entry point rather than in the Layout Micro-App, because toast notification behavior is a concern of the individual Business Micro-App, not a shared design system concern.

Line 9 : disables Vue's production tip, suppressing the console message that Vue displays in development mode reminding developers to use the production build. This is a standard housekeeping configuration present in all Vue applications.

Lines 10–17: define the `vueLifecycles` constant by calling the `singleSpaVue` function. This function is provided by the `single-spa-vue` adapter library and is the critical integration point between the Vue application and the Single-spa orchestration framework. It takes a configuration object with properties. The `singleSpaVue` call wraps these into a set of lifecycle functions that Single-spa can call in sequence to control this Micro-App.

Line 19: exports the bootstrap lifecycle function. Single-spa calls bootstrap once, the very first time the Micro-App is activated. It is used for one-time initialization tasks. After bootstrapping, Single-spa does not call this function again unless the Micro-App is fully unregistered and re-registered.

Line 20: exports the mount lifecycle function. Single-spa calls mount each time the user navigates to a route that activates this Micro-App. The mount function instantiates the Vue application and attaches it to the designated DOM element defined in the Launcher's HTML shell. Multiple mount and unmount cycles can occur within a single user session as the user navigates between Micro-Apps.

Line 21: exports the unmount lifecycle function. Single-spa calls unmount when the user navigates away from the routes handled by this Micro-App. The unmount function tears down the Vue instance and removes it from the DOM, freeing memory and

preventing event listener leaks. The Micro-App's bundle remains loaded in memory, so a subsequent navigation back to this Micro-App triggers mount directly without repeating the bootstrap phase.

Figure 3.5 below illustrates the SystemJS import map configuration used by the Launcher Micro-App to resolve and load the JavaScript bundles of each registered Micro Frontend at runtime. SystemJS is a dynamic module loader that enables the browser to fetch and execute ES modules from remote URLs (guybedford, 2020). The import map acts as a registry, mapping each Micro Frontend module name to its deployed bundle URL, allowing Single-spa to load each Micro-App from its independently hosted location

```
1 <script type='systemjs-importmap'>
2 {
3   'imports': {
4     'navbar': 'http://127.0.0.1:5001/js/app.js',
5     'research-vue-auth': 'http://localhost:5002/js/app.js',
6     'research-vue-admin': 'http://localhost:5003/js/app.js',
7     'research-vue-farmer': 'http://localhost:5004/js/app.js',
8     'research-react-product': 'http://localhost:3000/static/js/main.js',
9     'single-spa': 'https://.../single-spa/4.3.7/system/single-spa.min.js',
10    'vue': 'https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js',
11    'vue-router': 'https://.../npm/vue-router@3.0.7/dist/vue-router.min.js'
12  }
13 }
14 }
15 </script>
```

Figure 3.5: Loading Micro-Apps into the Launcher Micro-App

The loaded modules are registered in the Launcher Micro-App and mapped to a specific URL pattern, as shown in the code snippet in Figure 8 above.

Line 1: opens a script tag with the type attribute set to systemjs-importmap. This non-standard type value signals to the browser that the content of this script block should not be executed as regular JavaScript. Instead, it is consumed exclusively by the SystemJS module loader that has been previously loaded into the page. The browser

ignores this block entirely; SystemJS intercepts and parses it before any Micro-App loading begins.

Lines 2–3 open the JSON object and the imports key. The imports object is the standard structure required by the SystemJS import map specification. Every key-value pair inside it maps a logical module name.

Line 4 : This entry registers the Layout/Navigational Micro-App, which is served from port 5001 on the local development server. The navbar module is activated for all routes in the application and provides the persistent navigation shell and shared design system within which all other Micro-Apps are rendered.

Line 5-8: This entry registers the specific Micro-App. It is activated when the browser navigates to the route that is responsible for presenting the micro app. The micro apps runs on a different ports and are registered independently confirms that it can be developed, tested, and deployed by a separate team without any coordination with the admin or product Micro-Apps.

Line 9: single-spa maps to a CDN URL for the Single-spa library version 4.3.7. **Line 10: vue** maps to a CDN URL for Vue.js version 2.6.10.

Line 11: vue-router maps to a CDN URL for Vue Router version 3.0.7. This is the client-side routing library used by each Vue-based Micro-App to manage its internal navigation. Declaring it as a shared dependency ensures that all Micro-Apps use a consistent router version, which is a prerequisite for the JWT-based navigation guard described in the Zero Trust pseudocode

Lines 12–15: close the imports object, the outer JSON object, and the script tag respectively, completing the import map declaration. After SystemJS has parsed this block, it is ready to resolve any module name used in the application to its corresponding URL and fetch the bundle on demand when Single-spa activates the corresponding Micro-App.

Figure 3.9 below shows code in the root configuration script embedded in the Launcher Micro-App's HTML shell. It is the bootstrap engine of the entire Micro Frontend

application; the first JavaScript that executes when a user loads the application URL. It loads the shared framework dependencies (Single-spa, Vue, and Vue Router) in parallel, registers every Micro-App with Single-spa along with its activation rules, and starts the Single-spa orchestration engine. Once this script is completed, Single-spa takes full control of the browser's navigation events and manages the mounting and unmounting of each Micro-App for the remainder of the user's session.

```
1 <script>
2   (function () {
3     Promise.all([System.import('single-spa'),
4       System.import('vue'),
5       System.import('vue-router')]).then(function (modules) {
6       var singleSpa = modules[0];
7       var Vue = modules[1];
8       var VueRouter = modules[2];
9
10      Vue.use(VueRouter)
11
12      singleSpa.registerApplication(
13        'navbar',
14        () => System.import('navbar'),
15        location => location.pathname !== '/login'
16      );
17
18      singleSpa.registerApplication(
19        'research-vue-auth',
20        () => System.import('research-vue-auth'),
21        location => location.pathname.startsWith('/login')
22      )
23
24      singleSpa.registerApplication(
25        'research-vue-admin',
26        () => System.import('research-vue-admin'),
27        location => location.pathname.startsWith('/admin')
28      )
29
30      singleSpa.registerApplication(
31        'research-vue-farmer',
32        () => System.import('research-vue-farmer'),
33        location => location.pathname.startsWith('/farmer')
34      )
35
36      singleSpa.registerApplication(
37        'research-react-product',
38        () => System.import('research-react-product'),
39        location => location.pathname.startsWith('/product')
40      )
41
42      singleSpa.start();
43    })()
44  })()
45 </script>
```

Figure 3.6: Micro-App Mapping in Launcher App

Line 1 opens a standard HTML script tag. This block is executed synchronously by the browser as part of the page's initial load sequence, before any Micro-App bundle has been fetched.

Lines 2–5 : Promise.all and System.import initiate the parallel loading of three shared dependencies: single-spa, vue, and vue-router. Promise.all receives an array of three SystemJS import calls and resolves only when all three modules have been successfully fetched and parsed. Using Promise.all rather than sequential imports is a deliberate performance optimization

Lines 6–8 : extracts the three resolved modules from the results array. `singleSpa` is assigned `modules[0]` (the Single-spa library), `Vue` is assigned `modules[1]` (the `Vue.js` constructor), and `VueRouter` is assigned `modules[2]` (the `Vue Router` library). The order corresponds exactly to the order of the import calls in the `Promise.all` array on Lines 3–5.

Line 10 : globally registers the `Vue Router` plugin with the `Vue` constructor. This must be called before any `Vue` application instance is created, ensuring that every `Vue`-based `Micro-App` that is subsequently mounted has access to routing capabilities without needing to register the plugin individually in its own entry point.

Lines 12–17: registers the `Layout/Navigational Micro-App` with `Single-spa`. The first argument (`'navbar'`) is the unique name matching the import map entry. The second argument is the application loader; an arrow function that calls `System.import('navbar')` to dynamically fetch the `Micro-App`'s bundle when needed.

Lines 18–22 : `registerApplication: research-vue-auth` registers the `Authentication Micro-App`.

Lines 23–28: `registerApplication: research-vue-admin` registers the `Admin Business Micro-App`.

Lines 29–33 : `registerApplication: research-vue-farmer` registers the `Farmer Business Micro-App`, activated for all paths beginning with `/farmer`. This `Micro-App` encapsulates all functionality relevant to the farmer user role. Its separate registration and distinct URL path mean it can be released, rolled back, or scaled independently of the admin and product `Micro-Apps` — directly demonstrating the reduced deployment risk benefit endorsed by 56.25% of the technical review panel.

Lines 34–38: `registerApplication: research-react-product` registers the `Product Business Micro-App`, activated for paths beginning with `/product`. This is architecturally the most significant registration in the script: the `Micro-App` loaded here is built with `React` (`research-react-product`), not `Vue.js`. The fact that a `React` application can be registered alongside three `Vue` applications using exactly the same API call with no special handling or adapter code in the `Launcher` confirms the

framework's technology agnosticism. The Launcher does not know or care what framework a Micro-App uses internally; it only requires that the Micro-App exposes the Single-spa bootstrap, mount, and unmount lifecycle hooks.

Line 39 — `singleSpa.start()` starts the Single-spa orchestration engine. Until this line is called, all the preceding `registerApplication` calls have only declared the Micro-Apps to Single-spa. The `start()` call hands control of the browser's URL to Single-spa, which immediately evaluates all registered activation functions against the current URL, mounts the matching Micro-Apps, and begins listening for subsequent navigation events. From this point forward, every URL change is handled by Single-spa rather than the browser's native navigation mechanism.

Lines 40–42 close the then callback function, the immediately invoked function expression (IIFE) wrapper, and the script tag respectively.

The Navigation Micro-App controls the design and layout of the web application. The Menus and navigation are defined on this Micro-App to ensure a consistent look. Vue Router library is used to implement the client-side transitions between pages in the browser for the Micro-Apps (You & Hong, 2016). The Router library directly manipulates the history state of the browser window to achieve the effect of changing the URL when the page is reloaded.

For deployment, Webpack is used to create a bundle for each Micro Frontend. Webpack processes the application and builds a dependency graph to generate the packaged application. It is configured to pass all application files through the TypeScript compiler and then package the results into a single file loaded by the user's browser.

3.4.2.4 Pseudocode for Zero Trust Security Model

The following pseudocode describes the implementation of the Zero Trust Security Model within the Authentication Micro-App of the proposed Micro Frontend architectural framework. The model is enforced through a global navigation guard that intercepts every route transition before it occurs, verifying whether the destination

route requires authentication and whether a valid session token is present. This mechanism ensures that no transition between application states is implicitly trusted and that every access to a protected resource is explicitly validated, regardless of how navigation is initiated. The pseudocode is derived from the prototype implementation and is presented alongside a description of each logical step to facilitate understanding of the security design decisions embedded in the artifact.

```
BEGIN Security Interceptor  
IMPORT router  
ON EVERY route transition (destination, origin, proceed):  
IF destination route requires authentication  
AND session token DOES NOT exist OR is null THEN  
// User is not authenticated — redirect to login page  
REDIRECT to '/login'  
PRESERVE destination path as redirect query parameter  
// Enables return to intended page after successful login  
ELSE  
IF session token EXISTS in session storage THEN  
// Retrieve and clean token : remove Bearer prefix (first 7 characters)  
token ← GET session token FROM session storage  
token ← REMOVE first 7 characters from token  
// Validate token integrity and expiry  
CALL checkToken(token)  
// If invalid or expired, checkToken handles redirect to login  
END IF  
// Token is valid : allow navigation to proceed  
CALL proceed()  
END IF  
END ON EVERY route transition  
END Security Interceptor
```

The Security Interceptor operates globally and is registered on the application router, meaning it is invoked on every navigation event regardless of the originating component or Micro-App. This design choice reflects the Zero Trust security principle, which assumes that no transition between application states should be implicitly trusted and that every access to a protected resource must be explicitly validated (Lahiru, 2022).

The Security Interceptor's first logical operation is to inspect the metadata of the destination route to determine whether it has been marked as requiring authentication. This is achieved through the route's meta configuration, which carries a flag set at the time of route definition. If the route is protected and no session token is found in browser session storage, or if the stored token value is null, the interceptor redirects the user to the login page. Critically, the full path of the intended destination is preserved as a query parameter in the redirect, enabling the Authentication Micro-App to return the user to their originally requested resource after successful login, thereby preserving a seamless user experience.

If a session token is present in storage, the interceptor proceeds to extract and validate it. The token is retrieved from session storage, and the first seven characters are removed, stripping the Bearer prefix that is prepended to JSON Web Tokens (JWT) by the backend Authentication service upon login (Bucko et al., 2023). The cleaned token is then passed to a dedicated "checkToken" function, which is responsible for verifying the token's integrity, confirming that it has not been tampered with, and checking that it has not expired. If the token fails validation, the request is redirected to the login Micro-App. The Interceptor maintains a single point of responsibility for authentication and authorization. If the token is valid, the Interceptor calls the proceed function, allowing the navigation to continue to the requested route.

This implementation directly addresses one of the principal non-functional requirements identified in the framework design: the provision of a standardized, architecture-wide authentication and authorization mechanism that relieves individual Business Micro-Apps of the responsibility for managing login state and security enforcement. By centralizing this logic within the Authentication Micro-App and enforcing it globally through the router guard, the framework ensures that security is integrated into the application lifecycle from the outset, consistent with the recommendations of Chatterjee and Pitroda (2020) regarding early security integration in web application development. The JWT is stored in browser session storage rather than local storage, limiting token persistence to the duration of the browser session and reducing the risk of token theft through persistent storage attacks. Token validation logic is delegated to a dedicated "checkToken" utility function, maintaining the single

responsibility principle and making the validation logic independently testable and reusable.

3.5 Validity and Reliability

Ensuring the validity and reliability of research findings is a fundamental obligation of rigorous academic research. The use of structured ADRs to document design rationale provides a transparent and auditable record of the design process, supporting the assessment of internal coherence. The expert review evaluation, which includes practitioners with diverse backgrounds and experience contexts, provides an assessment of the framework's perceived generalizability from multiple professional perspectives.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1. Introduction

This chapter presents and discusses the findings derived from the two evaluation methods employed in this study: the technical architecture review conducted with industry experts and the empirical prototype evaluation using Google Lighthouse performance testing. Together, these methods provide a comprehensive basis for assessing the extent to which the proposed Micro-Frontend architectural framework addresses the three core challenges identified in the literature: UX inconsistency, authentication and authorization vulnerabilities, and lack of clear implementation guidance.

4.2. Technical Review Result

4.2.1. Reviewer Profile

The technical review engaged three categories of stakeholders: Software Engineers, who constituted 73% of the reviewers and represent the primary users of an architectural framework; Software Architects and Analysts, responsible for assessing conformance to quality attributes and business requirements; and System Administrators and Support Engineers, whose concerns Centre on operability, governance, and post-deployment maintainability. The distribution of reviewer specializations and experience levels provided a balanced and representative expert panel capable of evaluating the architecture from multiple stakeholder perspectives (Ågren & Knauss, 2022).

What is your area of specialization

15 responses

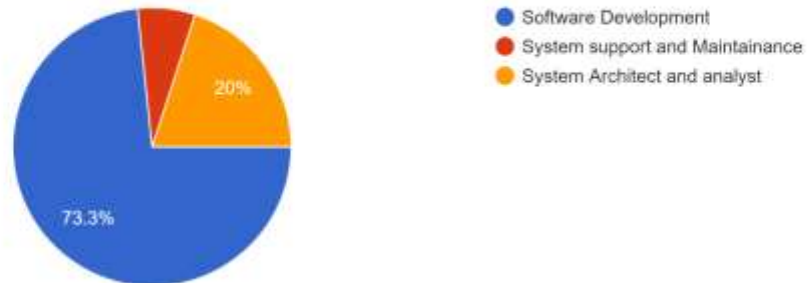


Figure 4.1: Summary of Technical Reviewer Area of Specialization

What is your level experience in your field of specialization

15 responses

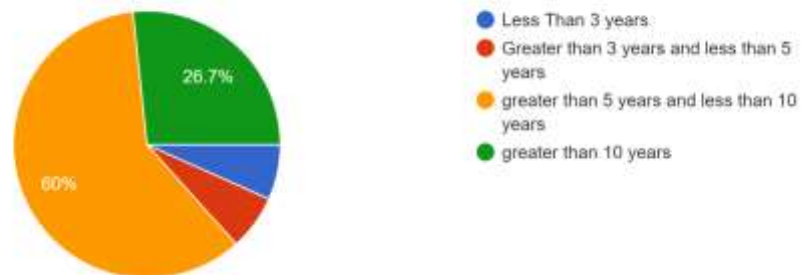


Figure 4.2: Level of Experience of Reviewers

4.2.2. Perceived Benefits of the Proposed Architecture

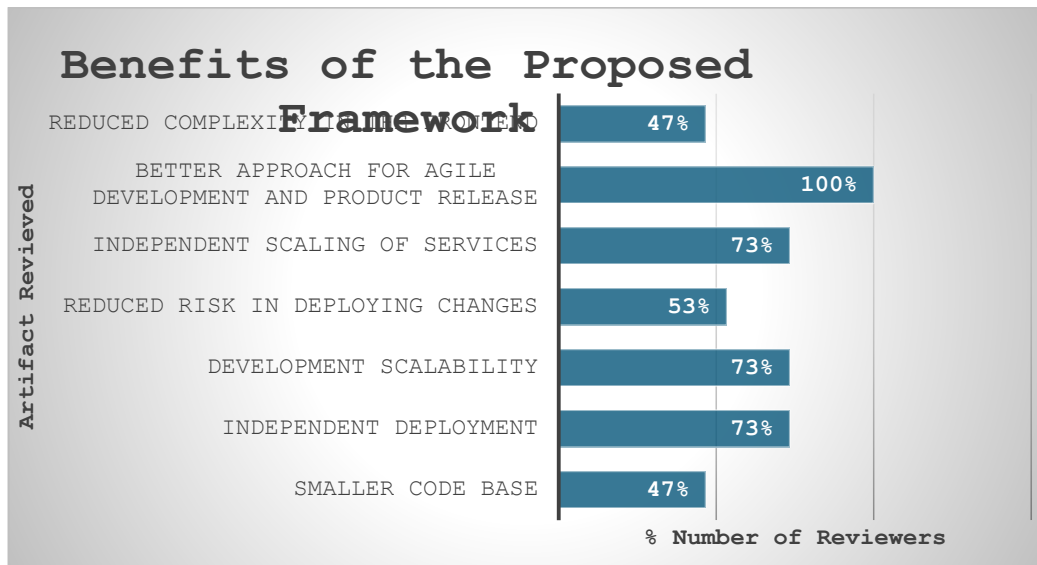


Figure 4.3: Feedback on the Benefits of Architecture

Reviewers were presented with the proposed architectural framework and asked to evaluate its anticipated benefits relative to established adoption motivations for Micro Frontend architectures, as documented by Severi et al. (2021).

The results revealed strong consensus across the reviewer panel regarding the framework's merits. Notably, 100% of reviewers agreed that the proposed architecture offers a superior approach to agile development and iterative product release cycles. This unanimous agreement is particularly significant, as agile alignment is one of the most frequently cited motivations for Micro Frontend adoption in the literature (Peltonen et al., 2021).

Furthermore, 73% of reviewers affirmed that architecture supports the following:

- Independent deployment of individual Micro Frontend modules
- Development scalability across distributed engineering teams
- Independent scaling of individual services

These findings collectively indicate that the proposed framework successfully extends the core benefits of Microservices architecture to the frontend layer, a key design

objective of this study. The Launcher Micro-App's role as a composition shell and the separation of the Authentication, Layout, and Business Micro-Apps into independently deployable units directly enable the independent deployment and scaling capabilities acknowledged by reviewers.

The support for development scalability is particularly noteworthy in the context of the framework's design rationale. By abstracting architectural concerns such as composition, authentication, and layout into dedicated Micro-Apps, the framework allows development teams to work autonomously on distinct business domains without coupling or conflict. This aligns with the principle of separation of concerns that underpins the framework's derivation methodology (Bass, Clements, & Kazman, 2021) and validates its practical utility in distributed development environments.

4.2.3. Challenges Introduced by the Proposed Architecture

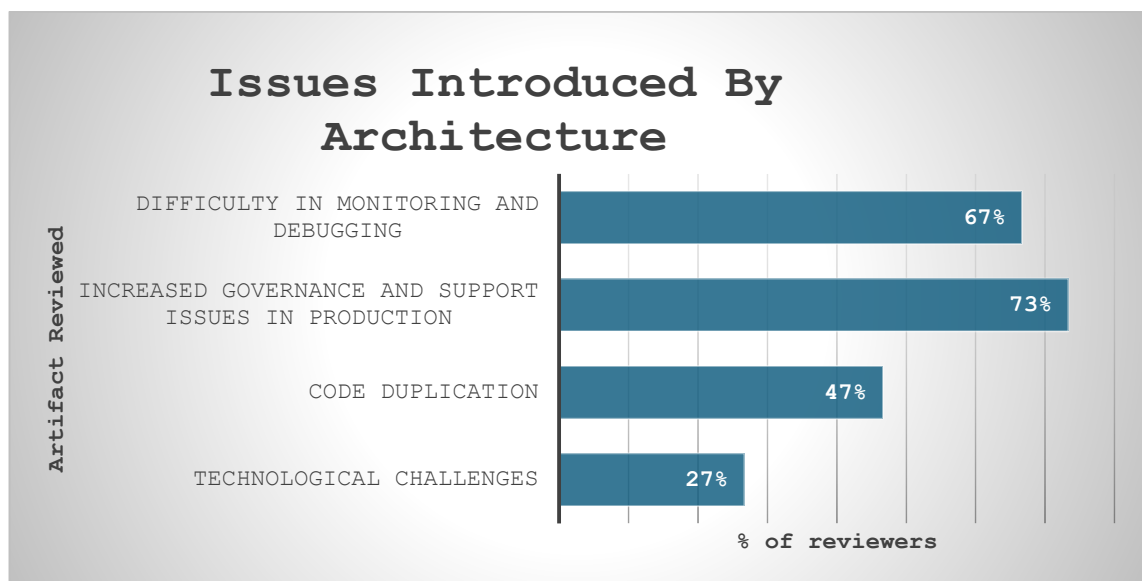


Figure 4.4: Feedback on Challenges Introduced by Architecture

While reviewers broadly affirmed the framework's benefits, the evaluation also surfaced important limitations consistent with the known shortcomings of Micro Frontend architectures identified by Severi et al. (2021).

73% of reviewers identified increased governance and support overhead in production as a significant concern. This finding is unsurprising given that the framework introduces multiple independently deployed applications, each requiring its own operational management. Although the framework reduces development complexity, it does not eliminate the operational complexity inherent in distributed frontend systems.

67% of reviewers cited difficulty in monitoring and debugging as a challenge introduced by the architecture. The distributed nature of the Micro-App modules, each running as an independent process in the browser, makes end-to-end tracing and fault isolation more complex than in a monolithic application. This represents a recognized trade-off of the Micro Frontend paradigm rather than a deficiency specific to the proposed framework.

46% of reviewers flagged code duplication as a potential concern. While the proposed framework mitigates some duplication through the shared Layout/Navigation Micro-App and a common design system component library, the independent development of Business Micro-Apps may still lead to redundant utility functions or shared dependencies bundled separately across modules.

These findings underscore that the proposed framework, while addressing core challenges of UX inconsistency and authentication vulnerability, does not resolve all inherent trade-offs of Micro Frontend architecture. Rather, it provides a structured baseline that reduces complexity for developers while explicitly acknowledging the operational overhead that adopting teams must plan for (Pavlenko, Nursultan, & Mazzara, 2020).

4.2.4. Architecture Adoption Preference

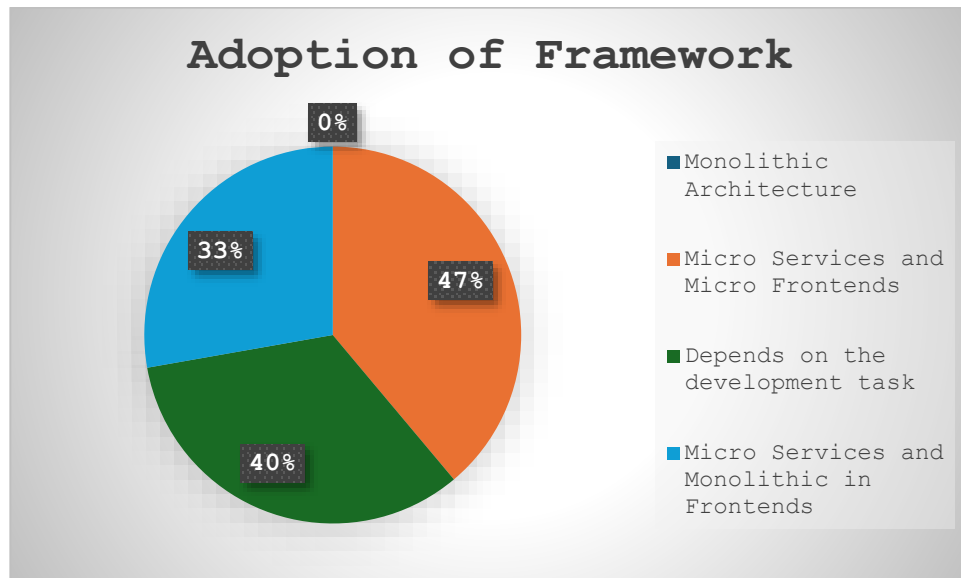


Figure 4.5: Feedback on Micro Frontend Adoption

Reviewers were asked to indicate their preferred architecture between the proposed Micro Frontend framework, a monolithic architecture, or a hybrid architecture for a real-world development project. Significantly, no reviewer recommended monolithic architecture, indicating that the perceived benefits of the proposed framework meaningfully outweigh its challenges for practitioners. 46% of reviewers indicated a preference for the proposed Micro Frontend architecture, reflecting a notable level of adoption confidence among experts who had reviewed its design in detail.

33% of reviewers remained neutral, indicating that their choice of architecture would depend on the specific nature and scale of the problem. This response is analytically meaningful: it suggests that the proposed architecture is not universally applicable to all development scenarios but is well-suited to large-scale, distributed, or agile development contexts, precisely the contexts for which it was designed (Brocke, Hevner, & Maedche, 2020). These findings strongly support the practical relevance of the proposed framework and its potential for adoption in industry settings.

4.3. Prototype Evaluation Results

4.3.1. Overview

To empirically validate the proposed framework and assess its performance characteristics relative to a conventional approach, we developed a working prototype in accordance with the framework's architectural principles and evaluated it against a comparable monolithic implementation. We used Google Lighthouse version 8 as the evaluation instrument, generating performance scores based on six weighted metrics: First Contentful Paint (FCP), Speed Index (SI), Largest Contentful Paint (LCP), Time to Interactive (TTI), Total Blocking Time (TBT), and Cumulative Layout Shift (CLS) (Google Developers, 2016).

Table 4.1: Google Lighthouse Performance Metrics

Metric	Good	Moderate	Slow	Weight
FCP	0-1.8 seconds	1.8-3.0 seconds	>3Seconds	10%
SI	0-3.4 seconds	3.4-5.8 seconds	>5.8Seconds	10%
LCP	0-2.5 seconds	2.5-4.0 seconds	>4.0Seconds	25%
TTI	0-3.8 seconds	3.8-7.3 seconds	>7.3 Seconds	10%
TBT	0-200 Millisecond	201-600 Millisecond	>600 Millisecond	30%
CLS	0-0.1	0.1-0.25	>0.25	15%
Overall Score	90-100%	50-89%	<50%	100%

4.3.2. First Contentful Paint (FCP)

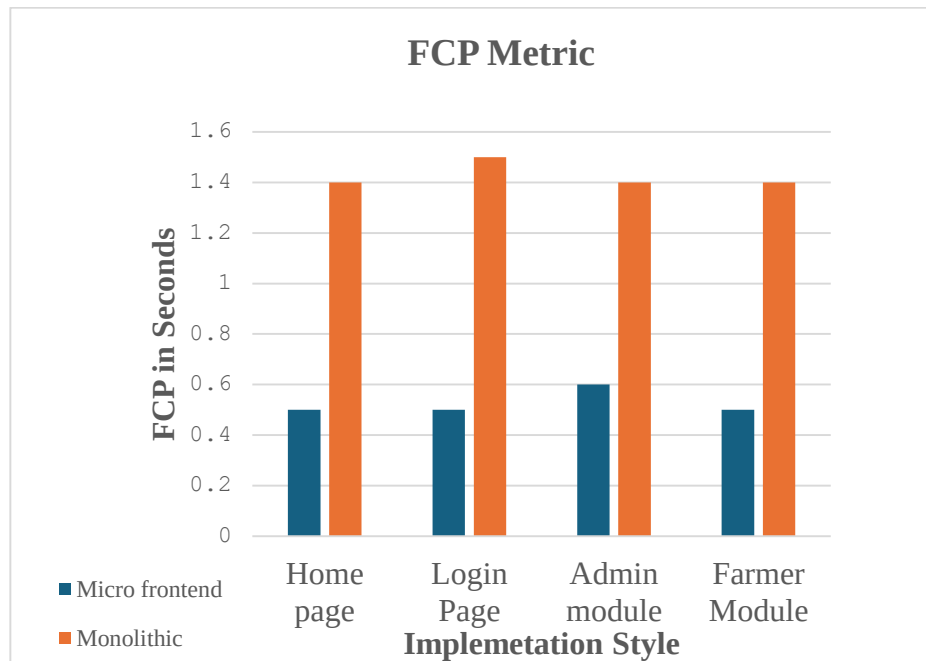


Figure 4.6: FCP Metric Comparison

The FCP results demonstrated a clear performance advantage for the Micro Frontend architecture. This outcome is attributable to the lazy-loading behavior of the framework, whereby only the Micro-App relevant to the current view is loaded and mounted in the browser's DOM at any given time. In contrast, the monolithic architecture loads the entire application bundle during the initial page access before rendering the specific view. As a result, the Micro Frontend prototype consistently shipped less code to the browser per view, yielding faster initial paint times and an improved user perception of responsiveness.

4.3.3. Speed Index (SI)

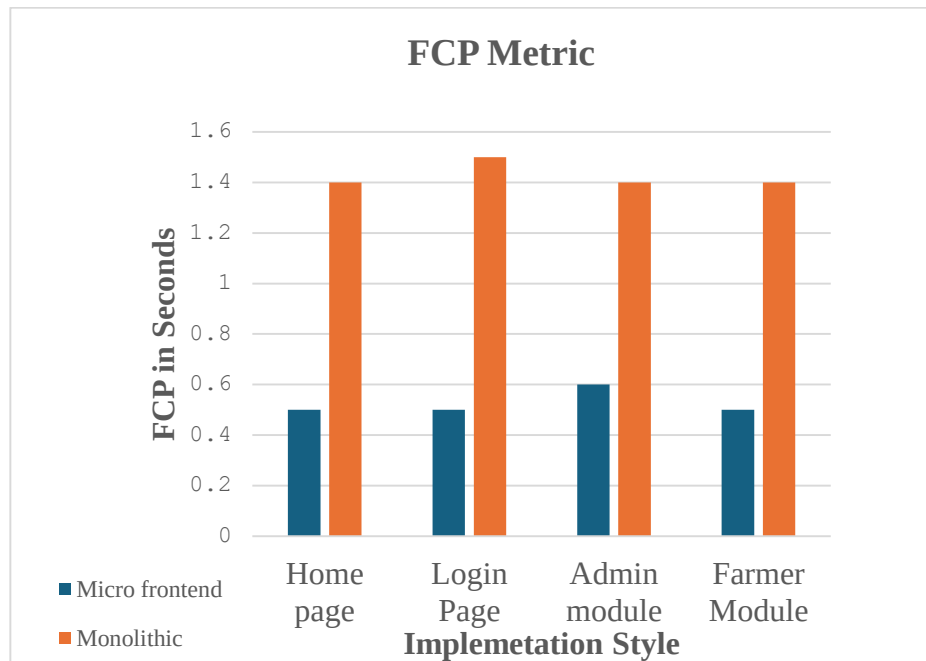


Figure 4.7: SI Metric Comparison

The Speed Index comparison revealed no significant difference between the two architectures. Both prototypes achieved comparable scores within the acceptable performance range, indicating that the introduction of multiple independently loaded Micro-Apps does not materially degrade the visual progression of page loading when compared to a monolithic application. This finding addresses a common concern regarding the overhead of client-side composition strategies (Jackson, 2019; Mezzalira, 2021).

4.3.4. Total Blocking Time (TBT)

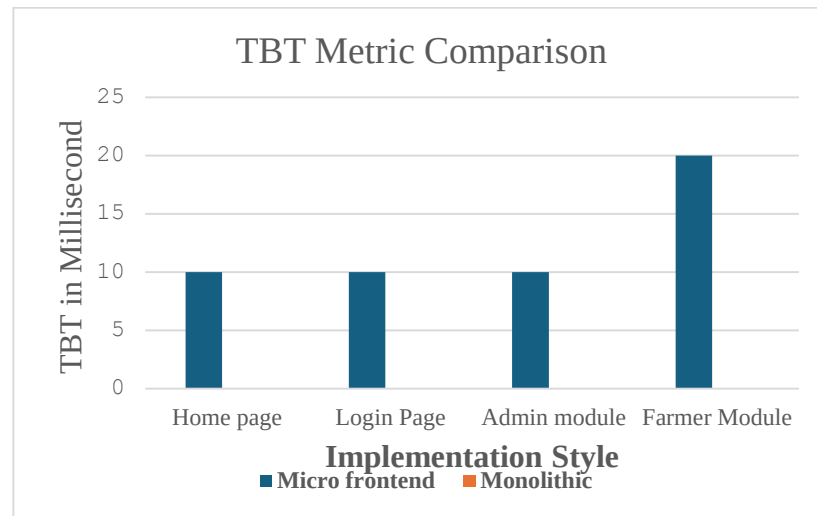


Figure 4.8: TBT Metric Comparison

The TBT metric showed a marginal advantage for monolithic architecture. This is attributed to the overhead introduced by the lazy loading and client-side composition of Micro-App modules when a URL transition occurs, momentarily delaying the browser's readiness to respond to user input. In the monolithic prototype, the initial full-application load means that subsequent page transitions involve only the rendering of smaller view components, resulting in slightly lower blocking times. However, both architectures recorded TBT values well within the 'Good' threshold of under 200 milliseconds, rendering this difference practically negligible from a user experience perspective.

4.3.5. Largest Contentful Paint (LCP)

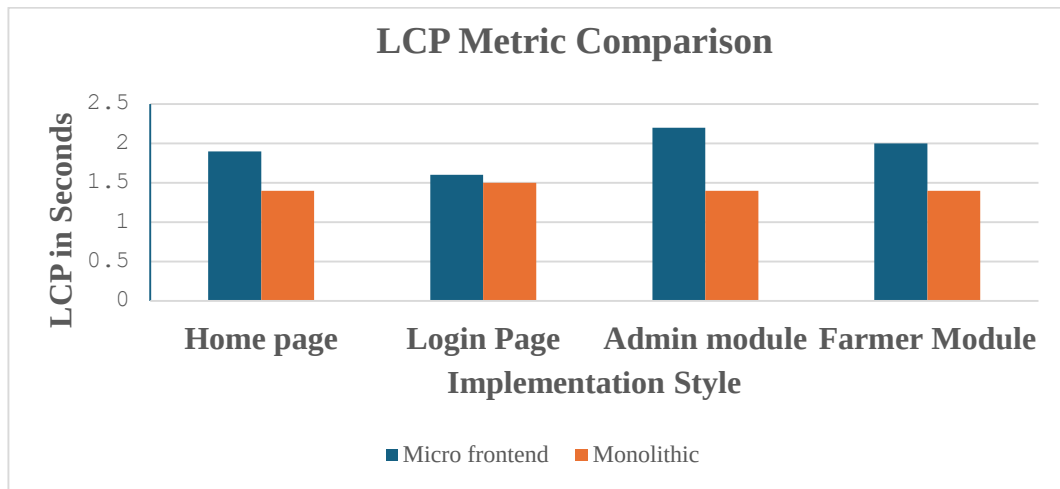


Figure 4.9: LCP Metric Comparison

Similarly, the LCP results showed a marginal advantage for monolithic architecture, again attributable to the lazy-loading composition overhead of the Micro Frontend approach. Nonetheless, both architectures achieved LCP scores within the 'Good' rating of less than 2.5 seconds. The difference is therefore statistically present but practically insignificant and does not constitute a meaningful performance regression for the proposed framework.

4.3.6. Cumulative Layout Shift (CLS) - UX Consistency

The CLS results represent one of the most significant findings of the prototype evaluation. Both the Micro Frontend prototype and the monolithic prototype achieved a CLS score of zero, indicating perfect visual stability with no unexpected layout shifts experienced by the user during page transitions.

This result is of particular importance given that UX inconsistency is documented as one of the most persistent challenges in existing Micro Frontend implementations (Pavlenko, Nursultan, & Mazzara, 2020). Earlier implementations of Micro Frontend architecture exhibited elevated CLS scores due to difficulties in coordinating layout across independently developed modules. The zero CLS achieved by the proposed framework's prototype directly validates the effectiveness of the Layout/Navigation

Micro-App as an architectural mechanism for enforcing design consistency across all Business Micro-Apps. By centralizing layout control and providing a shared design system component library, the framework eliminates the conflicting styles and inconsistent rendering behaviors that have historically undermined user experience in Micro Frontend systems.

4.3.7. Overall Performance Score

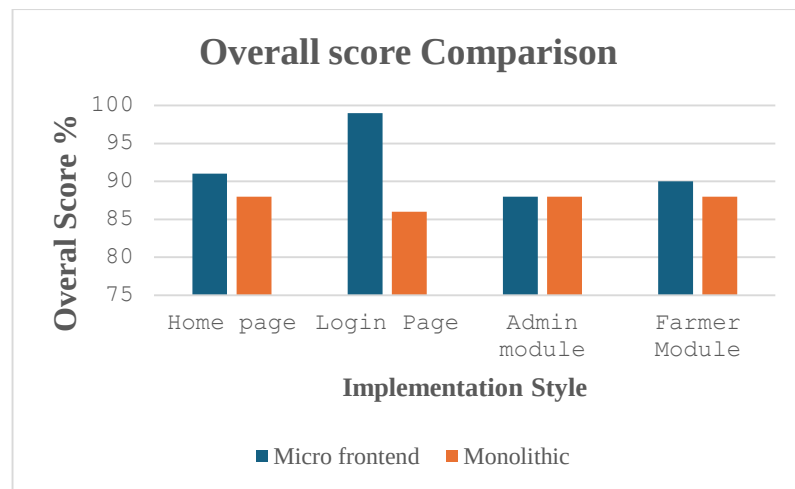


Figure 4. 10: Overall Score Comparison

The overall performance score comparison across all prototype pages demonstrated that the Micro Frontend architecture performed marginally better than or equivalent to the monolithic architecture across the evaluated pages. Both architectures recorded scores in the moderate-to-good range, with the Micro Frontend prototype achieving a score of 99 on the login page compared to 86 for the monolithic equivalent, 90 versus 88 on the home page, and equivalent scores of 88 on the admin module page.

Importantly, when the number of Business Micro-Apps was scaled to ten modules by replicating existing applications, no degradation in performance metrics was observed. This finding confirms a key architectural property of the framework: because Micro-Apps are loaded independently and only the active module is mounted in the DOM at any given time, increasing the number of modules does not increase load time or resource consumption for the end user. This scalability characteristic represents a significant advantage of the proposed framework over monolithic architecture, where

increasing application size directly increases the initial bundle delivered to the browser.

4.4. Discussion

4.4.1. Addressing UX Inconsistency

The prototype evaluation provides strong empirical evidence that the proposed framework effectively resolves the UX inconsistency challenge. The zero CLS score achieved by the Micro Frontend prototype stands in direct contrast to the layout instability reported in prior Micro Frontend implementations and validates the architectural decision to centralize layout governance within a dedicated Layout/Navigation Micro-App. The shared design system component library further reinforces this consistency by providing a single source of truth for visual components, eliminating the risk of divergent styling decisions across independently developed Business Micro-Apps (Bob, 2020).

4.4.2. Addressing Authentication and Authorization Vulnerabilities

The prototype successfully demonstrated role-based privileged access control in accordance with Zero Trust security principles (Lahiru, 2022). The Authentication Micro-App provides a centralized and standardized mechanism for user login and JSON Web Token (JWT) issuance (Bucko et al., 2023), relieving individual Business Micro-Apps of authentication responsibilities and reducing the risk of inconsistent or insecure implementations. The interceptor pattern implemented across each Micro-App requiring privileged access enforces token validation on every URL transition, ensuring that expired or missing tokens result in immediate redirection to the Authentication Micro-App regardless of the current session state.

This two-layered security model; centralized authentication at the framework level combined with distributed authorization enforcement at the Micro-App level, directly addresses the security vulnerabilities identified in the literature (Bob, 2020; Lahiru, 2022) and provides a replicable, framework-level solution that developers can adopt without designing custom security mechanisms for each module.

4.4.3. Addressing the Absence of Implementation Guidance

The structured design methodology, combined with the prototype's documented development process, demonstrates that the proposed framework provides the clear, technology-agnostic implementation guidance identified as absent in the existing literature. By defining four canonical Micro-App types; Authentication, Layout/Navigation, Launcher, and Business Micro Apps with explicit responsibilities and interaction patterns, the framework gives development teams a reproducible structural template that reduces architectural decision-making overhead (Fowler, 2019). The prototype confirmed that this guidance is sufficiently concrete to be implemented using widely adopted, industry-standard tools, while remaining flexible enough to accommodate the technology preferences of different teams.

4.4.4. Performance Trade-offs and Scalability

The overall performance results confirm that the proposed framework does not introduce meaningful performance regressions relative to monolithic architecture. The marginal disadvantages observed in TBT and LCP are attributable to the inherent overhead of client-side composition and lazy loading, and both metrics remained within acceptable performance thresholds. The framework's performance advantage in FCP, combined with its demonstrated linear scalability as the number of Business Micro-Apps increases, positions it as a technically sound choice for large-scale enterprise web application development (Luca, 2021).

4.5. Summary

The combined findings of the technical expert review and prototype evaluation provide converging evidence that the proposed Micro Frontend architectural framework successfully addresses the three core challenges of UX inconsistency, authentication and authorization vulnerabilities, and the absence of clear implementation guidance. The framework achieves this without introducing significant performance trade-offs relative to conventional monolithic approaches, and its benefits are recognized and affirmed by domain experts across multiple stakeholder categories. The challenges of governance overhead and monitoring complexity acknowledged by reviewers

represent known systemic trade-offs of distributed frontend architectures rather than deficiencies of the proposed framework specifically and should be addressed through complementary operational strategies during deployment planning (Severi, Luca, & Davide, 2021).

The proposed framework design offered a straightforward approach to developing web applications using Micro Frontends architecture, thereby reducing overall complexity and giving time to developers to concentrate on the business requirements without concern for the implementation details of Micro Frontend architecture. By having the Launcher Micro Frontend, developers do not have to worry about merging several Micro Frontends into a single page application; this, in turn, reduces complexity for developers. The design abstracts the developer from the details of orchestrating the Micro Frontends into single page application for the users. With the prototype developed using the proposed architecture showing a Cumulative Layout Shift of zero on Google Lighthouse, the proposed framework achieves and sufficiently handles consistent user experience.

(Bob, 2020), noted that authentication and authorization in Micro Frontends are a challenge because Micro Frontends exist in the same browser but run independently on different processes and access data and resources on the backend Micro Services using the same communication plane. (Lahiru , 2022) Recommended the use of a zero-trust security model in Micro Frontends to ensure proper authentication and authorization take place for each request and communication. The use of JWT for authorization of requests between the frontend and backend services offers this zero-trust security model by ensuring each request that is sent to the backend service is validated and authorized without regard to the existing session, as JWT is a stateless protocol. The Authentication Micro-App offers a standard approach to managing user authentication, implementing an interceptor code on each Micro Frontend that requires privileged access offers a second level of zero-trust security by ensuring a switch from one Micro Frontend requires validation of the token to confirm whether the current user is authorized to view the page and whether the token is still valid. This ensures unauthorized requests are rejected based on the configured authorization parameters.

The proposed architecture achieved privileged access, enabling users to access the prototype with different roles.

By having the Launcher Micro Frontend, developers do not need to concern themselves with merging several Micro Frontends into a single page application; this, in turn, reduces complexity for developers. The design abstracts the developer from details of orchestrating the Micro Frontends into single page application for the users.

The use of Micro Frontend framework architecture performed better than similar applications developed using monolithic architecture. This can be attributed to reduced load time for Micro Frontend pages, unlike in monolithic architecture where an entire application is loaded during initial page access. At any given time, only one Micro Frontend is active at any given time.

CHAPTER FIVE

CONCLUSION, CONTRIBUTIONS, AND FUTURE WORK

5.1 Introduction

This chapter brings the study to a close by synthesizing the key findings of the research, articulating the contributions the study makes to the existing body of knowledge, and identifying productive directions for future inquiry. The chapter begins by revisiting each of the four research objectives and presenting the principal findings associated with each, before proceeding to broader conclusions, a statement of research contributions, and recommendations for further work.

5.2 Review of Research Objectives and Findings

5.2.1 Review of First Objective

The first objective of this study was to identify the challenges experienced by current implementations of Micro Frontend systems. This was pursued through the guiding research question: What challenges are experienced by current implementations of Micro Frontend systems? The motivation for this inquiry was to develop a thorough understanding of the shortcomings present in existing Micro Frontend deployments, thereby establishing an empirical foundation upon which targeted solutions could be proposed with the intent of improving the overall adoption and quality of Micro Frontend-based applications.

A review of early Micro Frontend implementations revealed a range of recurring difficulties. This study focused on three principal challenges that most significantly impede effective adoption of the paradigm. The first is user experience (UX) inconsistency, arising from the independent development of frontend modules by separate teams without a shared design governance mechanism. The second is the complexity of implementing authentication and authorization securely across independently running Micro Frontend applications that coexist within the same browser environment. The third is the absence of a clear, structured implementation strategy which particularly disadvantages less experienced developers seeking to scale

web applications using this relatively novel architectural paradigm (Pavlenko, Nursultan, & Mazzara, 2020).

5.2.2 Review of Second Objective

The second objective was to propose an architectural framework for implementing Micro Frontend architecture that directly addresses the challenges identified in the first objective. This was guided by the question: Can the identified challenges be addressed by proposing a new architectural framework? The motivation was to document, in a structured and replicable manner, how each identified challenge could be resolved through deliberate architectural decisions.

The proposed framework applies the principle of separation of concerns by assigning each class of challenge to a dedicated Micro Frontend application. The Layout/Navigation Micro-App provides a reusable, centrally managed layout component applied consistently across all modules of the application. By centralizing design authority within a single Micro-App and enabling a shared CSS design system, the framework eliminates the risk of conflicting styles and inconsistent visual experiences that arise when independent teams manage their own layouts in isolation (Bob, 2020). This approach enforces design standardization across the development organization and reduces inter-team conflicts over interface conventions.

The Authentication Micro-App addresses the security challenge by providing a standardized, centralized mechanism for user authentication within the framework. It incorporates interceptor logic to operationalize a Zero Trust security model, ensuring that every transition between Micro-App modules is independently validated and authorized. The use of JSON Web Tokens (JWT) as the bearer of session credentials adds a stateless security layer to all frontend-to-backend communications, enabling each request to be independently authenticated without reliance on shared session state thereby mitigating risks such as session fixation and Cross-Site Request Forgery (CSRF) (Lahiru, 2022; Bucko & Vishi, 2023). The Launcher Micro-App abstracts the complexity of orchestrating multiple independently developed Micro Frontend modules into a cohesive single-page application (SPA). By handling composition at the framework level, it relieves individual development teams of the burden of

integration, reducing overall development complexity and providing a clearly defined entry point for the assembled application (Jackson, 2019).

Finally, the Business Micro-App concept extends the architectural decomposition approach of Microservices to the frontend layer. By proposing that each discrete business capability be encapsulated within its own independently deployable Micro-App, the framework enables frontend teams to benefit from the same autonomy, scalability, and release independence that backend teams achieve through Microservices, effectively bridging the architectural gap between frontend and backend development paradigms (Luca, 2021).

5.2.3 Review of Third Objective

The third research objective was to evaluate the developed architectural framework, pursued through the guiding research question: "What design artifacts are encompassed and addressed by the architectural framework?" The developed Micro Frontend architectural framework encompasses four primary design artifacts: the Authentication Micro-App, the Layout/Navigational Micro-App, the Business Micro-App, and the Launcher Micro-App with each corresponding to a discrete, independently deployable Micro-Application with a clearly defined responsibility boundary. Collectively, these artifacts give concrete expression to the framework's foundational design principle of separation of concerns, ensuring that each architectural component addresses a single, well-defined aspect of the application without encroaching on the responsibilities of the others.

The validity of these artifacts was confirmed through two complementary means of evaluation: a technical review by 16 experienced industry practitioners and the development of a working prototype. The technical review panel unanimously confirmed that the framework supports agile development and product release; 75% affirmed independent deployment capability and 68.75% endorsed development scalability and independent service scaling as tangible benefits delivered by architecture. The shared design system embedded within the Layout/Navigational Micro-App was recognized as a mechanism that partially mitigates the code duplication risk acknowledged by 50% of reviewers, and the clear component

boundaries established across all four artifacts were found to reduce frontend complexity and limit the risk surface of individual deployments.

The prototype implementation provided further empirical confirmation of the framework's quality artifacts. The four Micro-Apps were developed and deployed independently, each running on a dedicated port and integrated at runtime by the Launcher Micro-App into a cohesive single-page application demonstrating in practice the independent deployment and composition capabilities that reviewers endorsed in theory. The Authentication Micro-App successfully managed JWT-based session handling and enforced Zero Trust security across Micro-App transitions, validating the framework's security artifact under realistic conditions. The Layout/Navigational Micro-App rendered a consistent design and navigation structure across all Business Micro-Apps without requiring each team to implement its own layout logic, confirming the reusability and design standardization qualities of the framework. Individual Business Micro-Apps were developed in isolation and integrated into the running application without disrupting other components, directly demonstrating the reduced deployment risk and development scalability that the review panel endorsed. The prototype therefore corroborates the technical review findings and confirms that the framework addresses both the functional and non-functional requirements of modern web applications through a structured, well-documented artifact set consistent with the architectural description requirements of ISO/IEC/IEEE 42010:2022 (2022).

5.2.4 Review of Fourth Objective

The fourth objective was to develop a working prototype using the proposed framework to demonstrate its practical applicability and illustrate how specific implementation details can be realized. This was guided by the question: How can a prototype be developed for the proposed framework? The purpose was to produce a concrete implementation guide that developers intend to adopt the framework that can reference in real-world application development.

The prototype was developed using the Vue.js JavaScript framework in conjunction with the Single-SPA library for Micro Frontend orchestration and the SystemJS module loader for dynamic module loading. The implementation demonstrated in

practical terms how authentication and authorization are achieved through JWT issuance and validation, and how an interceptor mechanism enforces the Zero Trust security policy at every navigational transition between Micro-Apps. The integration of the backend API gateway as the authorization enforcement point for service requests was also demonstrated. The composition of independently developed Micro-Apps into a unified single-page application via the Launcher Micro-App was documented with accompanying code examples, providing a traceable link between the framework's architectural decisions and their concrete implementation (You & Hong, 2016; Joel, 2020).

4.5 Conclusions

The proposed architectural framework offers a structured and modular approach to implementing the Micro Frontend paradigm, extending the proven benefits of Microservices architecture to the frontend layer of web applications. By abstracting the concerns of design consistency, security, module composition, and business decomposition into four dedicated Micro-App types, the framework enables development teams to work independently and deliver features in parallel, merging their contributions into a cohesive single-page application with minimal integration effort and without introducing performance degradation.

The Layout Micro-App directly resolves the long-standing challenge of UX inconsistency in Micro Frontend systems, providing a centralized and reusable design governance mechanism. The Authentication Micro-App establishes a framework-level security standard that reduces the risk of insecure or inconsistent access control implementations across independently developed modules. The Launcher Micro-App significantly lowers the complexity barrier for developers new to the Micro Frontend paradigm by abstracting orchestration concerns away from individual development teams. Together, these components constitute a cohesive and replicable implementation strategy that addresses the core challenges identified in literature.

However, the framework is not without limitations. A notable coordination challenge arises from the fact that developers working on individual Business Micro-Apps may concurrently need to modify the Launcher and Layout Micro-Apps, particularly when

introducing new modules or updating navigation structures. Without disciplined version control practices and clear team communication protocols, concurrent modifications risk introducing integration conflicts that could destabilize the entire application. This challenge underscores the importance of establishing clear ownership and change management procedures for the shared framework components.

A further limitation relates to code duplication across independently developed Micro-Apps. While this can be mitigated by introducing a shared JavaScript library for common utilities, doing so creates a high degree of dependency on a single shared artifact, a pattern that conflicts with autonomy and loose coupling principles of both Microservices and Micro Frontend architecture. Practitioners adopting this framework are therefore advised to carefully evaluate the trade-off between reducing duplication and preserving module independence, and to establish explicit team agreements on what constitutes acceptable shared infrastructure versus domain-specific implementation (Severi, Luca, & Davide, 2021).

Overall, Micro Frontend architecture continues to gain traction as a compelling paradigm for large-scale frontend development, and the proposed framework represents a meaningful step toward making its adoption more accessible, secure, and consistent for development teams of varying experience levels.

4.6 Recommendations

The findings of this study yield a set of recommendations directed at two distinct audiences: organizations and institutions responsible for governing software development practice at a policy level, and software development practitioners and engineering teams responsible for implementing architectural decisions in real-world projects. These recommendations address the operational challenges surfaced by the technical review, the adoption gap between Microservices and Micro Frontend architectures identified in the survey findings, and the broader implications of the proposed framework for sustainable frontend engineering practice.

4.6.1 Policy Recommendations

The following recommendations are directed at organizations, enterprise architecture governance bodies that set architectural policy for software development programs.

Adoption of Micro Frontend architecture standards. Organizations developing complex web applications in regulated domains; particularly financial services, healthcare, and government, should consider formalizing the adoption of a structured Micro Frontend framework within their enterprise architecture governance policies. The absence of an institutional standard for Micro Frontend decomposition was identified as a key driver of the inconsistent and incomplete implementations reported in the literature (Severi et al., 2021). The proposed four-component framework provides a reusable, standards-aligned blueprint that reduces the governance risk associated with ad hoc frontend decomposition.

Early security integration mandate. Organizations should mandate the early integration of authentication and authorization mechanisms within the frontend development lifecycle, consistent with the Zero Trust security model implemented in this framework. Software development lifecycle governance policies should require that security architecture reviews of frontend systems be conducted at the design phase, not deferred to the testing or pre-release phase. The framework's Authentication Micro-App demonstrates that early security integration is architecturally feasible without imposing a burden on Business Micro-App development teams.

Investment in frontend architectural capability. The survey findings of this study reveal a significant maturity gap: 81.25% of respondents reported Microservices adoption in their organizations, while only 18.75% had adopted Micro Frontend architecture and 37.5% were uncertain of their organization's position. This asymmetry suggests that training, skills development, and knowledge transfer investment has followed architectural maturity without corresponding investment in frontend architectural capability. Organizations should address this gap through structured professional development programs, internal communities of practice, and incorporation of Micro Frontend architectural patterns into software engineering standards and onboarding curricula.

4.6.2 Industry and Practitioner Recommendations

The following recommendations are directed at software architects, engineering managers, and development teams responsible for making and implementing architectural decisions in practice.

Adopt the four-component decomposition model as a baseline. Development teams adopting Micro Frontend architecture should implement the proposed four-component structure; Authentication, Layout/Navigation, Business, and Launcher Micro-Apps as a starting baseline rather than designing component decomposition from scratch. Teams should resist the temptation to consolidate concerns across components, particularly the embedding of authentication logic within Business Micro-Apps, as this reintroduces the tight coupling and shared state management complexity that the framework is specifically designed to eliminate.

Establish observability infrastructure before production deployment. Given that 75% of technical reviewers cited increased governance and production support complexity as a challenge of the proposed architecture, development teams must invest in centralized logging, distributed tracing, and cross-Micro-App monitoring tool before deploying to production environments. Observability infrastructure should be treated as a first-class architectural concern, designed and provisioned in parallel with the Micro-App components themselves and not as an operational afterthought.

Define the shared design system before Business Micro-App development begins. The Layout/Navigational Micro-App's shared CSS design system should be defined, agreed upon, and published as a shared dependency before individual Business Micro-App development commences. Teams that defer design standardization risk the code duplication and visual inconsistency identified by 50% of reviewers. The design system should be versioned independently, with a clear governance process for proposing and approving changes, to prevent individual teams from diverging unilaterally.

Apply the framework selectively and proportionately. The proposed framework delivers the greatest value in applications with multiple development teams, complex

business domains, and frequent independent release requirements. For applications maintained by a single small team or with limited functional scope, the governance overhead associated with distributed frontend deployment may outweigh the benefits. Teams should conduct a structured cost-benefit analysis aligned with the quality attribute evaluation approach before committing to full Micro Frontend decomposition.

Plan explicitly for inter-team communication overhead. The autonomy benefits of Micro Frontend architecture depend on clear, stable interfaces between components. Teams should define and document the contract for each Micro-App's public interface including event bus schemas, shared context objects, and URL conventions before beginning implementation. Breaking changes to these interfaces require coordinated releases across teams, which reintroduces the coupling the architecture is designed to avoid.

4.7 Research Contribution

This research makes several original contributions to the body of knowledge in the domain of frontend software architecture, specifically in the context of Micro Frontend systems. The primary contributions are as follows:

- A structured architectural framework for Micro Frontend implementation that addresses three core challenges: UX inconsistency, authentication and authorization vulnerabilities, and the lack of clear implementation guidance.
- A demonstration of how consistent design and user experience can be achieved within a Micro Frontend architecture through the centralization of layout governance in a dedicated Layout/Navigation Micro-App, supported by a shared design system component library.
- A practical implementation of the Zero Trust Security Model within a Micro Frontend context, utilizing JWT-based stateless authentication, interceptor-enforced authorization at every navigational transition, and API gateway-level request validation for backend services, providing a replicable security blueprint for Micro Frontend practitioners.

- A working prototype that serves as a concrete implementation reference for developers intending to adopt the proposed framework, with documented code examples covering module registration, composition, authentication, authorization, and routing.
- An empirical comparison of the performance characteristics of Micro Frontend architecture and monolithic architecture using Google Lighthouse, providing evidence that the Micro Frontend paradigm, when implemented using the proposed framework, does not introduce meaningful performance regressions. This addresses a frequently cited concern among practitioners considering adoption of this architectural style.

Collectively, these contributions reduce barriers to Micro Frontend adoption by providing both theoretical guidance and practical implementation artifacts that development teams building modern web applications can apply directly.

4.8 Future Research

While this research has addressed several significant challenges in Micro Frontend architecture, it also opens several avenues for further investigation. No architectural framework constitutes a universal solution to all problems in frontend web development, and continued effort from both practitioners and researchers will be essential to mature the Micro Frontend paradigm for adoption in mission-critical, production-grade applications.

A productive direction for future research concerns the characterization of problem domains for which Micro Frontend architecture is most suitable. Not all applications benefit equally from the complexity and overhead introduced by a distributed frontend architecture. Further empirical studies are needed to identify the application characteristics, such as team size, business domain complexity, release frequency, and scalability requirements that most reliably predict whether a Micro Frontend approach will outperform monolithic or hybrid alternatives.

A second important area relates to tooling for automated integration testing of Micro Frontend systems. The independent development of Micro-App modules introduces

challenges in validating the correctness of inter-module interactions during development, prior to merging contributions into the shared Launcher application. The availability of mature, purpose-built automated testing frameworks for Micro Frontend integration would significantly reduce the risk of regression and lower the barrier to adoption in environments that require high reliability guarantees. This gap represents a compelling opportunity for tool developers and researchers to collaborate.

A third avenue for future work concerns the challenge of code duplication across independently developed Micro-Apps. This study noted that utilities such as interceptor logic for JWT validation are duplicated across multiple Micro-App modules, each independently validating the token to enforce access control. Future research should investigate the conditions under which code abstraction into a shared library is beneficial relative to the cost of coupling such abstraction introduces, and explore architectural patterns that minimize duplication without compromising module autonomy particularly in the context of security-critical utilities where inconsistent implementations pose meaningful risk (Brocke, Hevner, & Maedche, 2020).

Finally, future work should explore the governance and operational frameworks required to manage Micro Frontend systems effectively at scale in production environments. The technical review conducted in this study identified increased governance and support overhead as the concern most frequently cited among expert reviewers. Research into monitoring architecture, distributed tracing strategies, and incident management practices specifically tailored to Micro Frontend deployments would make a valuable contribution to the practical maturity of this paradigm.

REFERENCES

- Agren, S. M., & Knauss, E. (2022). Architecture evaluation in continuous development. *Journal of Systems and Software*, 184. <https://doi.org/10.1016/j.jss.2021.111098>
- Anwar, P. (2023). Development of web application using Model-View-Controller (MVC) architecture. *International Journal of Progressive Sciences and Technologies*, 37(1), 156–161.
- Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley Professional.
- Betts, T., & Humble, C. (2020). *Software architecture and design InfoQ trends report*. InfoQ. <https://www.infoq.com/articles/architecture-trends-2020>
- Bigelow, S. J. (2020). *Functional vs. nonfunctional requirements in software engineering*. TechTarget. <https://searchsoftwarequality.techtarget.com/answer/Functional-and-nonfunctional-requirements>
- Bob, M. (2020). *Micro-frontends strengths and benefits*. Toptal. <https://www.toptal.com/front-end/micro-frontends-strengths-benefits>
- Brocke, J. V., Hevner, A., & Maedche, A. (2020). Introduction to design science research. In J. V. Brocke, A. Hevner, & A. Maedche (Eds.), *Design science research. Cases* (pp. 1–13). Springer.
- Buck, C., Olenberger, C., Schweizer, A., Völter, F., & Eymann, T. (2021). Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. *Computers & Security*, 110. <https://doi.org/10.1016/j.cose.2021.102436>
- Bucko, A., & Vishi, K. (2023). Enhancing JWT authentication and authorization in web applications based on user behavior history. *Computers*, 12(4), 78. <https://doi.org/10.3390/computers12040078>

- Caifang, Y., Chuanchang, L., & Zhiyuan, S. (2019). Research and application of micro frontends. *IOP Conference Series: Materials Science and Engineering*, 569. <https://doi.org/10.1088/1757-899X/569/5/052024>
- Cam, J. (2019). Micro-frontends. Martin Fowler. <https://martinfowler.com/articles/micro-frontends.html>
- Chatterjee, S., & Pitroda, Y. (2020). Dynamic role-based access control for decentralized applications. In *Proceedings of the International Conference on Blockchain* (pp. 185–197). Springer..
- Di Francesco, P., Malavolta, I., & Lago, P. (2017). Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *Proceedings of the IEEE. International Conference on Software Architecture (ICSA)* (pp. 21–30). IEEE.
- Fowler, M. (2019). Software architecture guide. Martin Fowler. <https://martinfowler.com/architecture/>
- Fowler, M., & Lewis, J. (2014). Microservices: A definition of this new architectural term. Martin Fowler. <https://martinfowler.com/articles/microservices.html>
- Geers, M. (2019). Micro frontends. *Micro-frontends.org*. <https://micro-frontends.org>
- Guybedford. (2020). SystemJS: Dynamic ES module loader. *GitHub*. <https://github.com/systemjs/systemjs>
- Hassan, S., & Bahsoon, R. (2016). Microservices and their design trade-offs: A self-adaptive roadmap. In *Proceedings of the IEEE International Conference on Services Computing (SCC)* (pp. 813–818). IEEE.
- Huang, P., Simpkins, C., & Coma, R. D. (2018). Mooo framework. *GitHub*. <https://github.com/phodal/mooo>

- IEEE. (2022). ISO/IEC/IEEE 42010:2022 Systems and software engineering Architecture description. *IEEE Standards Association*. <https://standards.ieee.org/ieee/42010/6846/>
- Jackson, C. (2019). Micro frontends. *Martin Fowler*. <https://martinfowler.com/articles/micro-frontends.html>
- Joel, D. (2020). Single-SPA: The complete guide. *Single-SPA*. <https://single-spa.js.org/docs/ecosystem-vue/>
- Koskinen, I., & Frens, J. (2018). Research prototypes. *Archives of Design Research*, 31(3), 5–15.
- Lahiru, H. (2022). Why micro frontends need zero-trust. *Bits and Pieces*. <https://blog.bitsrc.io/why-microfrontends-need-zero-trust-89ff8dd3121e>
- Lewis, J., & Fowler, M. (2019). Microservices: A definition of this new architectural term. *Martin Fowler*. <https://martinfowler.com/microservices/>
- Mena, M., Corral, A., Iribarne, L., & Criado, J. (2019). A progressive web application based on microservices combining geospatial data and the Internet of Things. *IEEE Access*, 7. <https://doi.org/10.1109/ACCESS.2019.2929222>
- Mezzalana, L. (2021). *Building micro-frontends*. O'Reilly Media.
- Michael, G. (2021). *Micro frontends in action* (1st ed.). Manning Publications.
- Natesan, N. (2019). How to implement design pattern: Separation of concerns. *CAST Software*. <https://www.castsoftware.com/blog/how-to-implement-design-pattern-separation-of-concerns>
- Newman, S. (2021). Building microservices (2nd ed.). O'Reilly Media. Oner, Z. (2019). Understanding micro frontends. HackerNoon. <https://hackernoon.com/understanding-micro-frontends-b1c11585a297>

- Parsons, R., Ford, N., & Kua, P. (2022). *Building evolutionary architectures* (2nd ed.). O'Reilly Media.
- Pavlenko, A., Nursultan, A., & Mazzara, M. (2020). Micro-frontends: Application of microservices to web front-ends. *Journal of Internet Services and Information Security*, 10(2), 49–66.
- Pooyan, J., & James, L. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3), 24–35. <https://doi.org/10.1109/MS.2018.2141039>
- Riehle, D. (2000). *Framework design: A role modeling approach*. ETH Zurich. <https://riehle.org/computer-science/research/dissertation/index.html>
- Severi, P., Cerny, T., & Taibi, D. (2021). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. *Information and Software Technology*, 136. <https://doi.org/10.1016/j.infsof.2021.106571>
- Google. (2016). Lighthouse. Retrieved from <https://developers.google.com/web/tools/lighthouse>
- You, E., & Hong, B. (2016). *Vue.js: The Progressive JavaScript Framework*. Retrieved from <https://vuejs.org>