Report: An Exploratory Study of Micro Frontends

Summary:

This study aimed to explore both the strengths and challenges associated with micro frontends, focusing specifically on how the modifiability of a web application is impacted by adopting a micro frontends approach. The investigation involved the implementation and comparison of a Single Page Application (SPA) and a Micro Frontends Architecture (MFA) prototype. Additionally, interviews were conducted with practitioners having a background in microservices, gathering their perspectives on micro frontends and related aspects. The results were used to assess the most suitable prototype for the specific web application. Furthermore, measurements on the implemented prototypes were conducted to estimate their modifiability using the Software Quality Measurement for Modifiability Assessment (SQMMA) mathematical model. However the prototype was really small and simple, so it may not provide accurate results. Moreover, the practitioners lacked hands-on experience with micro-frontends; their insights were solely speculative, drawn from their familiarity with microservices in the backend.

Keywords:

MF implementation | MF strengths | MF weaknesses | practitioners interviews | micro-services

Research Questions:

1. **What are the effects of adopting the micro frontends concept in a web application project?**
    a. What do practitioners believe are the positive effects on the development of the software as well as the software itself?
    b. What risks should be considered when starting a project using the micro frontends concept according to practitioners?
2. **How is the modifiability of an application affected by adopting the micro frontends concept?**
    a. What do practitioners believe are the implications of implementing a web application using the micro frontends concept with regards to modifiability?

b. Can the claim that the modifiability of a micro frontends application is better than that of an SPA be verified with a quantitative modifiability model using code metrics?

3. **When should the micro frontends concept be used?**
    a. In which type of projects do practitioners believe it can be valuable to adopt the micro frontends concept, and when should it not be used?
    b. Is it suitable to use for the application prototype presented in the study?

Key Points:

➔ **Micro frontends are not just an architectural approach but also an organisational strategy.**
- the presentation layer is divided into small, manageable pieces, dedicated to a specific feature
- system of autonomous micro applications with individual continuous delivery pipelines
- each micro-application function independently, even if others are down
- different implementation techniques can be used for each stack within a single web application. *[8]*

➔ **Unlike the traditional horizontal team structure, micro frontends promote vertical teams working on a single full stack micro application.** *[8-9]*

➔ **There are several reasons why companies adopt the micro frontends architecture:**
- optimises feature development by including both frontend and backend developers within the same team
- narrow scope and smaller codebase leads to easier understanding and refactoring and reduce of coupling between modules
- simplify frontend upgrades, allowing independent changes by each team without interfering with others
- facilitates migration from an old application by allowing a new application to run alongside the old one. *[9]*

➔ **Single-SPA is a JavaScript framework for implementing micro frontends:**
- supports multiple different frontend frameworks
- consists of a root application and a number of micro applications
- the root application renders an HTML page and a JavaScript file that registers micro applications

- each micro-application is registered with a name, a function to load the code of the micro application, and a function that determines when the micro application should be active
- the micro-applications must know how to mount and unmount itself from the DOM, but do not have their own HTML page. *[28]*

➔ **Full Stack Teams Owning Features**

- feature service owned by a single team leads to in-depth knowledge due to the narrow scope, covering the service from presentation to data layer
- reduce the risk of tasks being forgotten and clear team responsibilities
- working on applications from the top layer to the bottom reduces potential misunderstandings as opposed to separate frontend and backend teams.
- developers would enjoy belonging to full stack teams for more varied work tasks
- teams owning specific features should be flexible with team members to avoid running out of tasks. due to the varying workloads of features
- virtual teams where members can switch between feature teams based on current requirements, allowing teams to be paused during periods of inactivity. *[44-45]*

➔ **Standalone Applications**

- isolation and separate deployment, preventing one micro app from breaking others
- individual micro-applications can experience issues without affecting overall functionality, minimising downtime and ensuring continued access to other features
- isolation effectiveness depends on the absence or minimal presence of dependencies between micro applications
- enabling small releases of specific web application parts is highlighted as a significant benefit. *[45-45]*

➔ **Separate Techniques Side by Side**

- flexibility in choosing techniques can lead to better performance and
- benefit of adjusting choices based on the team's competencies
- ability to implement micro applications using different techniques is a key feature of the micro frontends concept

- using multiple frontend frameworks could affect the performance of the web application remarkably since all frameworks have to be loaded into the web browser *[45–46]*

➔ **Effect on Further Development**

- incremental upgrades, allowing changes in implementation tools or versions for one micro application at a time
- supports small teams, allowing isolated development without affecting others
- isolated micro applications make it easier to work with smaller version control repositories, reducing merge conflicts and simplifying repository handling
- may lead to challenges when implementing changes due to disappearing competencies or outdated tools
- setting up the development environment is seen as the most challenging aspect, emphasising the importance of general agreements between teams on configuring setup tools *[46–47]*

➔ **Effect on Testing**

- isolated services with distinct responsibilities, making components easier to handle and test.
- testing many small services in microservices backend development is considered easier than testing one large monolith, although dependencies between services may complicate integration testing.
- complexity of testing in microservices architecture due to inter-service communication and dependencies.
- testing must ensure that all micro applications have access for example to user credentials at the correct time, requiring tests to be run on the final, composed MFA. *[47–48]*

➔ **Suitability in Certain Applications**

- application size is a crucial factor when considering the micro-frontends architecture
- the architecture is valuable for large, complex applications that are challenging for a single team to handle, maintain, and comprehend
- splitting an application into very few simple micro applications may not justify distributing them across different teams, especially if one team can handle the entire application
- micro-frontends may be suitable for future versions if the application is expected to expand, making it easier to add features side by side

- micro-frontends are fitting when a web application must perform various tasks, contain many services, or be provided by different companies
- benefits depend on complete isolation between micro applications; too many dependencies may negate advantages
- may not be suitable for tight deadlines or small budgets due to configuration overhead *[48-49]*

➔ **Challenges With Isolated, Non-Dependent Teams and Micro Applications**

- Communication challenges exist in backend microservices, leading to difficulties in defining data needs and potential over-dependencies
- Risk of too many dependencies in microservices causing unnecessary complexity and difficulty in bug detection
- Microservices may need access to the same data source, and coordinating multiple requests for the same data is challenging
- Isolated teams in micro frontends pose challenges for handling non-functional requirements, such as browser support
- Working with isolated teams may lead to variations in code quality across micro applications
- Suggested the need for a dedicated user experience team to ensure consistent UI design

Infographics:

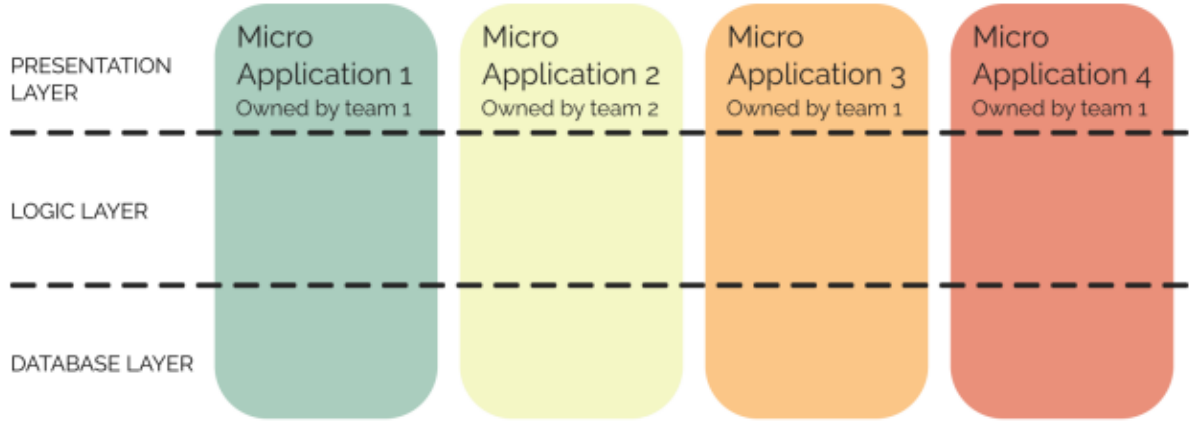

Figure1: Architectural overview of an MFA [9]