Report: Micro-frontends: application of microservices to web front-ends

Summary:

The article begins with an exploration into web development and various architectural approaches. It then delves into a practical case study illustrating the implementation of micro-frontends within a real-world project, addressing emerging challenges along the way. However, it falls short in providing detailed insights into the construction of the micro-frontend, focusing primarily on outlining its architecture, requirements, and component breakdown without specifying the development approach utilised. Furthermore, the article addresses critical issues such as routing, serving static assets, and project organisation, offering potential solutions without furnishing concrete examples. Ultimately, while it offers a brief explanation of relevant terminology, the article lacks substantial value in terms of actionable guidance and practical demonstration.

Keywords:

MF implementation | MF issues | MF organization | FE architectures | micro-services

Research Questions:

1. Explore the micro-frontends methodology within the framework of single-page applications (SPAs).

Key Points:

➔ **M. Geers outlines the key principles of micro-frontends as follows:**

- **Technology stack independence** – teams should be able to choose their technology stack without any impact from the others.
- **Code isolation** – the sharing of runtime is discouraged, and applications are advised to be self-contained.
- **Team Prefixes** – to prevent potential conflicts, components that cannot be isolated should be appropriately prefixed.
- **Native browser API** – the default browser's APIs should be preferred over a custom ways of inter-application communication. *[54]*

➔ **New issues introduced by micro-frontends are:**

- **Orchestration** – addressing the loading of applications as required.
- **Routing** – efficiently managing application routes.
- **Isolation** – establishing boundaries to prevent conflicts within the environment.
- **Communication** – determining effective methods for inter-application communication.
- **Consistency of UI/UX** – ensuring a uniform and cohesive UI across all applications.
- **Dependency management** – avoiding the loading of identical libraries multiple times. *[55]*

➔ **There exist five approaches for implementing micro-frontends:**

- **Server-side template composition** – composition of applications in the server during HTML page generation
  - old approach
  - HTML page must be generated after each user's request
  - lose of all SPA benefits
- **Build-time integration –** micro-frontends are delivered as Javascript packages and integrated during the final compilation of modules.
  - all micro-fronends must be recompiled if any of them is changed
- **Run-time integration via iframes** – isolation of applications is achieved through the utilisation of the iframe tags.
  - impossible to share common dependencies across different iframes
  - larger size
  - communication via iframe API
- **Run-time integration via JavaScript** – micro-frontends are delivered as separate bundles, loaded and mounted only when needed.
  - possible to pre-load common libraries and styles
  - each bundle can be deployed separately
- **Run-time integration via Web components** – Web Components enable run-time integration by harnessing new HTML standards, empowering the creation of custom HTML elements with defined behaviour and dynamic code loading. The browser inherently ensures isolation and offers better communication methods.
  - New standards, not yet fully implemented across all browser
  - This challenge can be addressed by employing polyfills, which implement new features using older versions of JavaScript. However this leads to an increase in the size of content and download time. *[55]*

➔ **SingleSPA library – used to manage actions such as:**
- Registration of micro-frontends
- Managing routes
- Mounting and unmounting micro-frontends applications into DOM-tree. *[60]*

➔ **There are 3 ways to organise micro-services and micro-frontends projects:**
- **Mono-repository** – All modules are located in one single repo, structured by folders
  - Each time whole codebase must be downloaded
- **Multi-repository** – Separate repositories for different micro-frontends and common parts.
  - More complex of developer tools setup and maintenance
- **Multi-repository with git-repo tool** – Multiple repositories managed by Google's git-repo tool. The tool uses a configuration file, what repositories to download and how they should be structured.
  - No need to download the whole codebase
  - No need for package registry
  - Little bit higher setup complexity *[62-63]*

➔ **There is no fully ready-to- use solution to bootstrap micro-frontends projects.**
- A lot of time must be spent on tool preparation and project setup.
- For micro-frontends to be accessible to small teams, there has to be comprehensive tool support to greatly simplify the setup. *[64]*

Infographics:

	Server-Rendered Pages	Single Page Application	SPA with Chunk Splitting	Micro-frontends
Static or dynamic GUI	Typically static with dynamic parts	Typically dynamic	Typically dynamic	Typically dynamic
Framework restrictions	Without framework	Single framework	Single framework	Any number of frameworks
Delivery to the client	Loads one page at a time	Loads fully, in the beginning	Loads partially, on request	Loads partially, on request
Simultaneous work	Possible	Usually, impossible	Usually, impossible	Possible
Setup complexity	Simple	Simple	Normal	Complex
Maintenance complexity	Normal	Normal/Complex	Normal/Complex	Normal/Complex

Table. 1: Comparison of approaches to building frontends [61]

	Mono-repository	Multi-repository	Multi-repository with git-repo tool
VCS	Any	Any	Git
Additional tools required	No	No	git-repo
Package registry required	No	Yes[1]	No
Partial codebase downloading	Impossible	Possible	Possible
Setup complexity	Simple	Complex	Complex
Checkout speed	Slow	Fast	Fast

Table 2: Comparison of approaches to project organisation [63]