LEVERAGING MICRO-FRONTENDS TO INCREASE THE MODULARITY OF WEB APPLICATIONS

KRISTI BRAHOLLI, KLESTI HOXHA

Department of Informatics, Faculty of Natural Sciences,

University of Tirana, Albania

e-mail: kristi.braholli@fshn.edu.al

Abstract

This paper explores the concept of micro-frontends in the development of web-based applications, as an architectural approach designed to address the challenges of scalability and maintainability in modern systems. Driven by the rapid growth and increasing complexity of web applications over the past fifteen years, there has been a need for better management, clarity, and structured architecture. Micro-frontends offer a solution by breaking down the frontend into smaller, independent units that can be developed, deployed, and maintained by different teams. Inspired by the success of microservices, developers have started to apply similar architectural principles to the frontend, giving rise to the micro-frontend approach. Although the concept is not entirely new, it has gained popularity in recent years due to the rise of single-page applications and the need for more efficient scaling and maintenance. This approach offers several benefits, including faster development cycles, improved scalability, and reduced maintenance costs. However, it also requires careful planning and coordination to be successfully implemented in real-world projects. To evaluate this architecture, in this work we design and implement a case study prototype that leverages the micro-frontend architecture combining React, NextJS, VueJS, and NodeJS using Webpack Module Federation. Despite its notable advantages, including independent deployment, this architecture was found to introduce additional maintenance complexity for small-scale applications.

Key words**:** *Micro-frontends, scalability, maintainability, modular design, software architecture.*

Përmbledhje

Ky punim trajton konceptin e mikrofrontendeve në zhvillimin e aplikacioneve të bazuara në ueb, si një qasje arkitekturore që synon të adresojë sfidat e shkallëzimit dhe mirëmbajtjes në sistemet moderne. Duke u nisur nga evolucioni i vazhdueshëm i teknologjisë dhe rritja e kompleksitetit të aplikacioneve njëfaqëshe, mikrofrontendet ofrojnë mundësinë për ndarjen e ndërfaqes së përdoruesit në njësi më të vogla dhe të pavarura, të cilat mund të zhvillohen dhe mirëmbahen në mënyrë të pavarur nga ekipe të ndryshme. Në këtë kontekst, punimi eksploron mënyrën sesi copëzimi i funksionaliteteve në komponente më të vegjël ndikon në fleksibilitetin, strukturën dhe shkallëzueshmërinë e aplikacioneve. Vihen në dukje përfitimet dhe sfidat që lidhen me adoptimin e mikrofrontendeve, duke përfshirë përmirësimin e cikleve të zhvillimit, reduktimin e kostove të mirëmbajtjes dhe nevojën për koordinim të mirë mes ekipeve të zhvillimit. Për të vlerësuar arkitekturën në fjalë u ndërtua një prototip rast studimi që kombinon teknologjitë React, NextJS, VueJS, dhe NodeJS nëpërmjet Webpack Module Federation. Përveç lehtësive të ndjeshme, duke përfshirë vendosjen në punë në mënyrë të pavarur, u vu re që për aplikime të vogla kjo arkitekturë krijon kompleksitet shtesë në mirëmbajtjen e aplikimeve në fjalë.

Fjalë kyçe: *Mikrofrontend-e, shkallëzueshmëri, mirëmbajtje, dizenjim modular, arkitekturë softueri.*

Introduction

Historically, web applications were divided into two main components: a back-end (server-side) responsible for processing, storing, and serving data via direct database calls or APIs, and a front-end (client-side) designed to present these data through a user-friendly interface (see Figure 1), enabling smooth user interaction (Nielsen, 2015).

With rapid technological changes, application structures are evolving. Modern clients demand increasingly complex functionalities, requiring sophisticated solutions and experienced development teams. While early-stage projects may not face significant challenges, large-scale applications pose difficulties in managing code complexity and meeting tight deadlines. Consequently, new approaches in code engineering have emerged.

Schäffer et al. (2019) suggest that adopting microservices and micro-frontends (Geers, 2020) can address these challenges. These concepts, although separate, can be integrated to build scalable, modular applications. Microservices (see Figure 2) divide the back end into small, independent, specialized components communicating via well-defined protocols, typically HTTP (Pavlenko, Askarbekuly, Megha, & Mazzara, 2020).

This architecture enables parallel development, reusability, and scalability, unlike monolithic architectures where all functionalities reside in a single program, limiting flexibility and requiring full system redeployment for changes (Nielsen, 2015; GeeksforGeeks, n.d.).

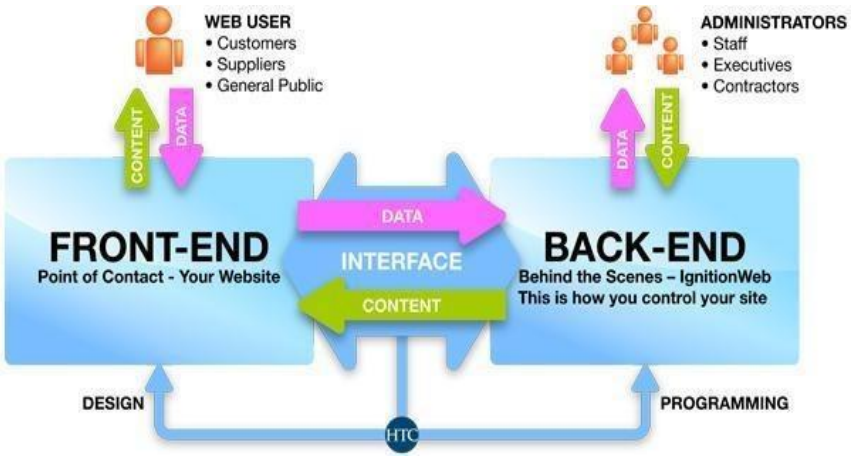

Figure 1. Front-end and Back-end (Front-end vs Back-end Developer Difference, 2015).

On the front end, application complexity has increased similarly. Inspired by microservices, the frontend community developed the micro-frontend architecture (Pavlenko, Askarbekuly, Megha, & Mazzara, 2020; Thoughtworks, n.d.). A micro-frontend divides a frontend application into independently deployable, team-owned modules, each focused on a specific business function.

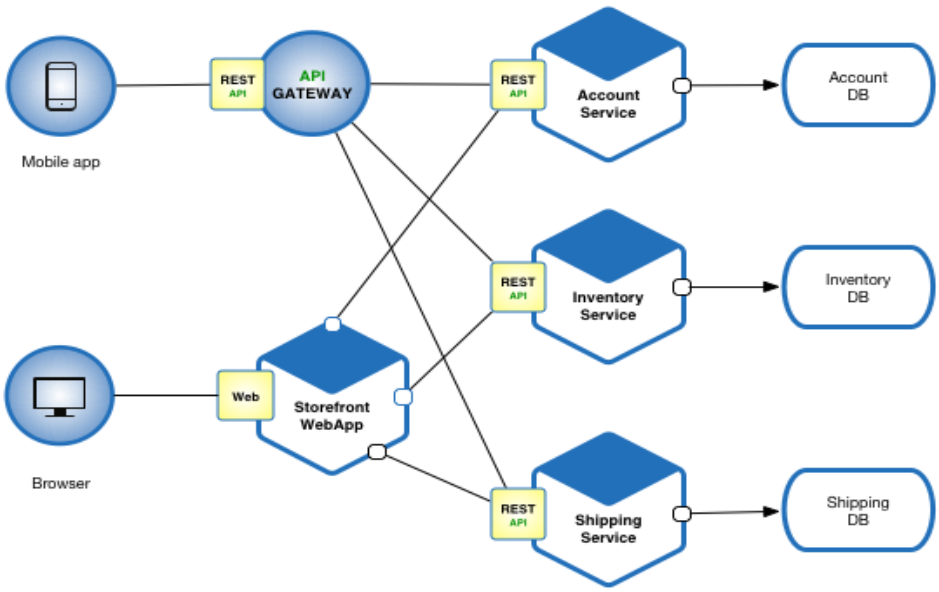

Figure 2. Microservices overview (Microservices, n.d.)

In this work, the main approaches of splitting the frontend layer of an application in terms of micro-frontends (microservices) are highlighted. We have developed a case study prototype that implements the horizontal splitting approach using WebPack, an important communication enabler. We investigate the extent to which Webpack Module Federation enables horizontal splitting within micro-frontend-based applications and evaluate the architectural advantages introduced by this decomposition strategy. The paper is concluded with our final thoughts and observation regarding the micro-frontend architecture.

Splitting methods

Although in principle they apply the same idea, there are several ways of partitioning these applications (see Figure 5). The most typical ones are horizontal, vertical, and composition-based divisions (Prajwal, Parekh, and Shettar, 2021).

Vertical split

Vertical split (see Figure 3) is implemented by splitting the application into subparts that are ultimately brought together again into a single "node" or web page. The separation is applied at the domain level, and each team adopts what is called Domain-Driven Design (DDD), within which it develops its independent work. So, each micro-frontend basically will consist of a single page or a set of pages, no further divisions between them.

Furthermore, Mezzalira (2021) mentions that the application is divided into several independent components, each responsible for a specific aspect of the application's functionality.

These micro-frontends are independent in the same way as microservices and can be developed and managed in full autonomy. This means that a different team can work on each micro-frontend and later decide to integrate them together into a complete application (Mezzalira, 2021).

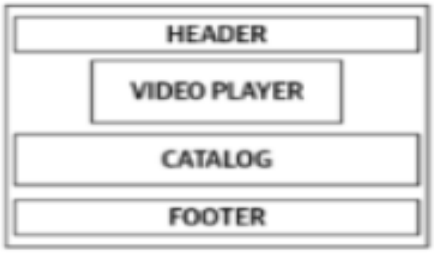

Figure 3. An example vertical micro-frontend (Mezzalira, 2021)

Again, according to Mezzalira, to divide an application into micro-frontends, the application must first be decomposed into small functional units. This division should be done in such a way that each micro-frontend contains only what is necessary to perform its specific functionality and is not tightly coupled with the rest of the application. This ensures that if one micro-frontend changes, only that part of the application will be affected, while the others remain unaffected.

Horizontal Split

The horizontal split of micro-frontends (see Figure 4) is a concept in which a view of a web application is divided into several independent groupings of horizontal functionalities. Horizontal division allows development teams to work on separate sets of functionalities and to focus exclusively on them. For example, one team may be responsible for developing the product details view of an application, while another team may focus on the footer part of it. Each team works independently and uses its own technologies and methodologies to develop and manage its specific functionalities (Mezzalira, 2021). This approach needs a composition strategy when rendering the full page.

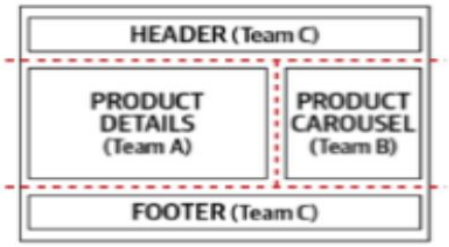

Figure 4. Horizontal split in micro-frontends (Mezzalira, 2021)

This horizontal separation of Micro-frontend functionalities aligns with the concept of Microservices division, where application components are split into small and independent services. Each Micro-frontend group can operate as an autonomous team that applies horizontal division to enable parallel development and clearly defined responsibilities for each specific functionality. Just like vertical division, the horizontal division of Micro-frontends is advantageous for scaling the development team, as it helps distribute the application's complexity and allows each team to concentrate on the specific areas in which they are most specialized.

To implement the horizontal division of micro-frontends, it is important to identify and define the horizontal groups of functionalities that are independent yet need to collaborate to create the final application.

Furthermore, proper coordination and communication among teams are essential to ensure that all micro-frontend components are connected and function together as a complete system.

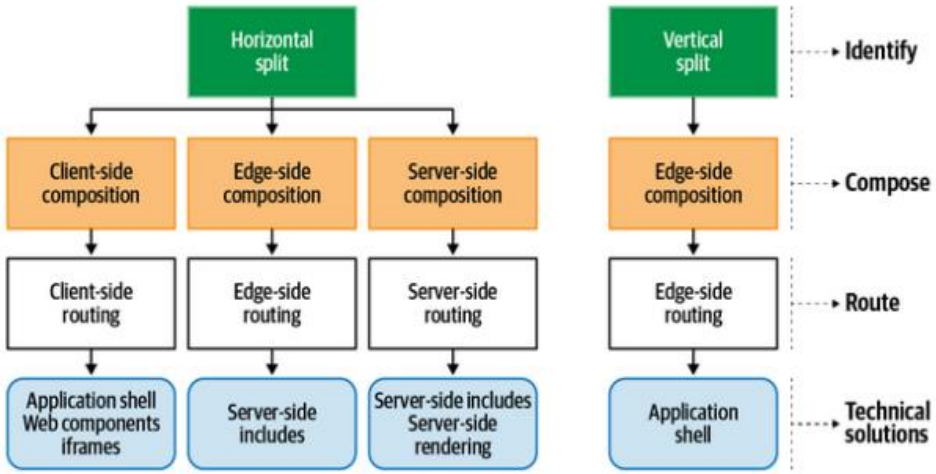

Figure 5. Horizontal split vs. vertical split (O'Reilly, n.d.)

Related Works

Despite their advantages, micro-frontends require careful planning for orchestration, routing, isolation, UI/UX consistency, and inter-module communication. They are not universally suitable; smaller, simpler applications may benefit more from monolithic approaches (Pavlenko, Askarbekuly, Megha, & Mazzara, 2020).

Recent research has examined how micro-frontend architectures can enhance modularity, scalability, and team autonomy in modern web applications. Mezzalira (2019) describes an application organized into multiple subdomains based on user interactions and following Domain-Driven Design (DDD) principles, where each subdomain was implemented as a micro-frontend,

except for the video component, which was treated separately. Micro-frontends were loaded and orchestrated by a bootstrap application that abstracted platform- and device-specific requirements, allowing them to run across different devices without modifying the code and enabling teams to work independently while maintaining delivery speed.

Yang et al. (2019) explore a content management system structured as function-based modules using a micro-frontend approach. Independent sub-applications were created for different business modules, with a main project based on the Mooa framework managing the loading, routing, and user access control dynamically. While this architecture offered benefits such as independent development and continuous deployment, it also posed challenges related to integrating diverse technology stacks, preventing conflicts, and optimizing resource usage.

Pavlenko et al. (2020) present a case study of an Education Hub system, which aggregates online courses from multiple providers to serve as a single-entry point and search engine for users. Their approach utilized the Backend-for-Frontend (BFF) model to ensure a unified access point for the frontend. In contrast, this thesis examines similar scenarios while combining multiple JavaScript frameworks into a single source, demonstrating how micro-frontends can be adapted to more complex and heterogeneous applications.

Together, these examples illustrate the potential of micro-frontends to increase modularity, reduce code duplication, and enhance team autonomy in web application development, while also highlighting the challenges of orchestration, integration, and maintaining a consistent user experience across modules.

Nikulina and Khatsko (2023) propose a structured approach on converting an existing monolith app into using the micro-frontends architecture. Similarly to the generic microservice architecture they propose starting with the identification of the business features using them as the first step towards a modular and later micro-frontend based approach.

Webpack as a Communication Enabler

Webpack (see Figure 6) is a tool for compiling and bundling resources in web applications. Its main purpose is to assist in organizing, transforming, and

optimizing the application's code and assets to prepare them for deployment in the production environment.

It is well-suited for projects that involve multiple interdependent resources, such as those using Node.js, React, Vue, or Next.js. It features a module-based architecture that allows developers to fully leverage modular development principles. It enables the importing and exporting of resources between modules, providing a flexible and efficient way to manage application dependencies and assets.

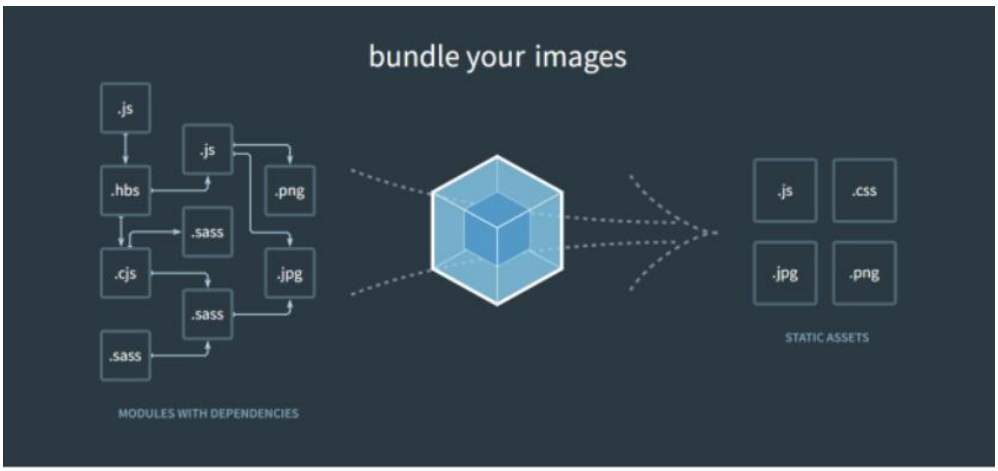

Figure 6. Webpack bundling (Webpack, n.d.)

Module Federation

Module Federation (see Figure 7) is a Webpack configuration that enables the sharing and integration of modules between different web applications. The process begins with the host (or producer) application, which exports the modules it intends to make available. Through the Webpack configuration and the use of the ModuleFederationPlugin, developers specify the modules to be exported, defining the module name, file location, and export settings (Webpack, n.d.).

The existence of numerous frameworks and libraries can create challenges in their integration and composition. Since Webpack is supported by most of the leading development frameworks, and moreover, offers the Module

Federation feature, it becomes a powerful tool for building micro-frontends and dividing applications into independent modules.

In addition, Module Federation enables the reuse of modules across different applications, reducing code duplication and increasing development efficiency. This architecture is particularly beneficial in large-scale projects that require frequent interaction between applications or the distribution of responsibilities among multiple development teams.

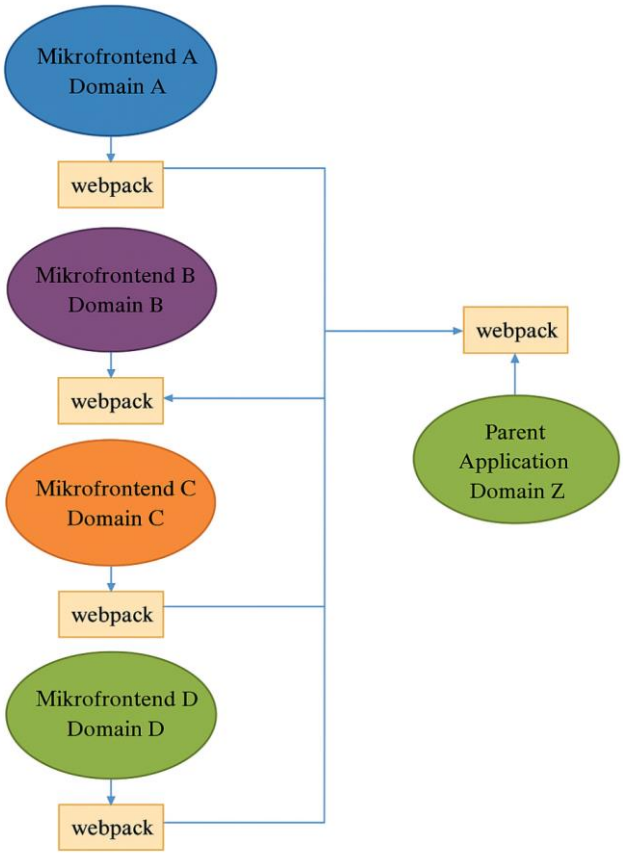

Figure 7. Visualization of how module federation works.

Case study prototype

Recent works on micro-frontends, being a relatively new and evolving concept, are somewhat preliminary and often focused on a single technology or framework. In the following case study, we leverage modules, plugins, or frameworks to create a bridge connecting multiple frameworks as micro-applications within a single parent application. This approach enables virtually unlimited development possibilities, allowing fully independent teams specialized in a single framework to develop and maintain specific functionalities at the module or component level.

Integration of multiple frameworks in a parent application

In the prototype presented below, the concept of linking the most widely used libraries and frameworks into a parent application is applied, along with the implementation of a micro-frontend architecture. The technologies used in this example include:

- NodeJS (no additional framework / library involved)
- React
- NextJS
- VueJS
- Webpack

These frameworks are invoked and managed by a parent application, or consumer, in our case NodeJS based.

Example Use Case: Currency Converter Web Application

The following prototype demonstrates the creation of a web application that provides currency conversion functionality along with historical information displayed as charts for the selected currency. The goal of it is to illustrate how pieces of code from different frameworks can be used together, while maintaining interdependency and communication capabilities between them.

This methodology highlights the potential for combining multiple frameworks into a unified micro-frontend system, showcasing how independent modules can interact seamlessly within a parent application to deliver complex functionalities.

The individual modules of it, the tab controller, the filter component, and the chart displayer have been developed through independent modules using the technologies mentioned above. Please refer to Figures 8, 9, and 10 for further details. We used module federation for the inter-communication between them. We observed that each individual micro-frontend can be developed, run, and deployed independently. Our prototype uses publicly available APIs, in case the backend part of the application would be developed by the same company (team), it would be advisable that the API (backend) and consumer frontend components be developed by the same team.

A notable benefit of the implemented architecture is the possibility of deployment of newer versions of portions of the user interface. It was also noticed that teams with different tech stack abilities can quickly collaborate in creating larger applications, reducing the learning curve needed to implement the same application with the same technology. This is in the same line as the traditional microservice (usually backend) based architectures. Out of the above outlined division approaches, we relied on the horizontal splitting and module federation. The benefits are not immediately noticeable considering the overhead related to deployment and bundling / composition. Same as with traditional microservices architectures, the micro-frontend approach seems more fit for carefully engineered larger applications.

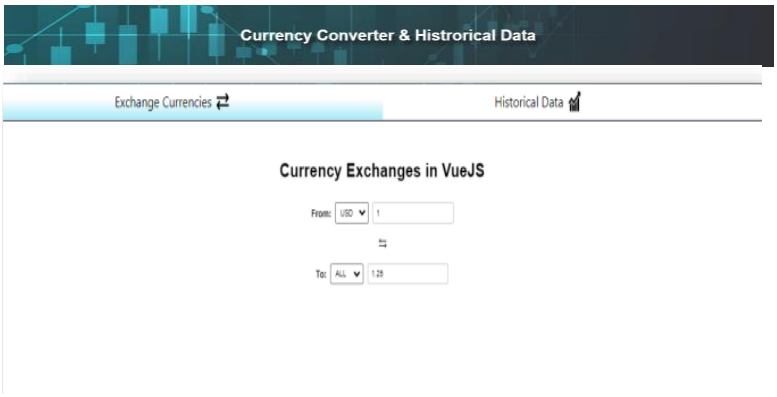

Figure 8. Micro-frontend developed on VueJS.

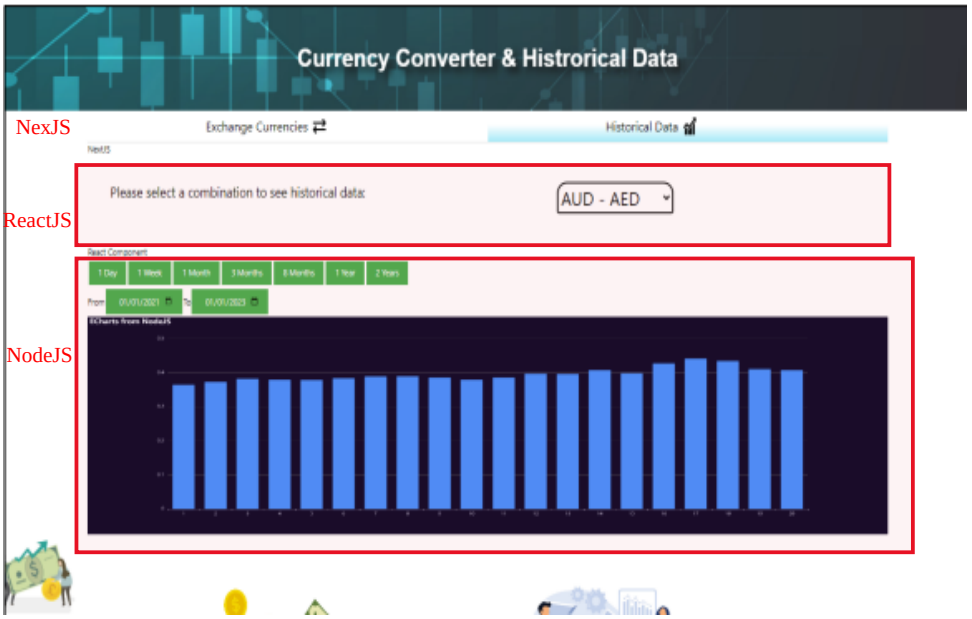

Figure 9. Micro-frontend splitting of the developed prototype.

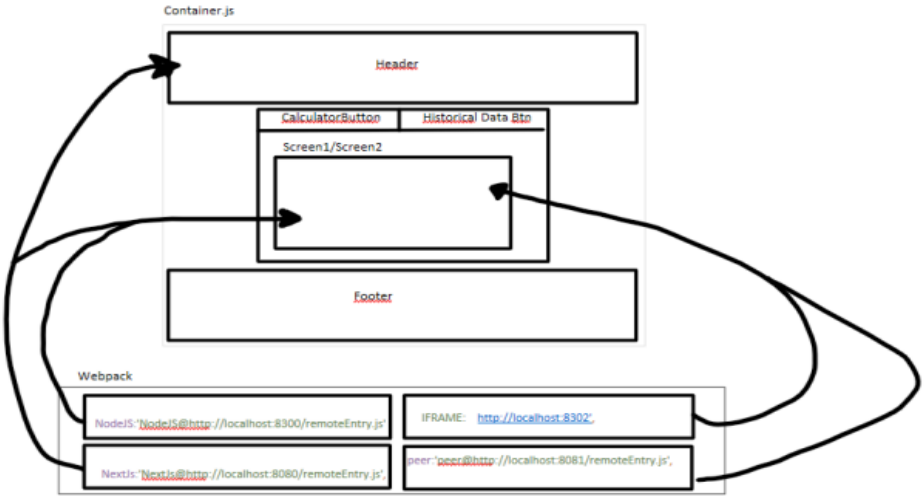

Figure 10. Composition diagram of the implemented use case.

Figure 11 shows an excerpt from the WebPack configuration of one of the micro-frontends (producer) and the host app of the implemented case study prototype. The filters component (micro-frontend, see Figure 9) has been exposed as usable remotely while it has been loaded (composed) by the parent app. This composition strategy is deployment agnostic, each individual micro-frontend may be deployed independently and redundantly if needed.

```
// Consumer (host)

plugins: [
    new ModuleFederationPlugin({
      name: 'ExchangeRateCalculator', // Name of the app
      remotes: {
        MyApp: 'IntervalFilters@http://localhost:3000/remoteEntry.js', //Producer
      },
    }),],

// Producer

plugins: [
  new ModuleFederationPlugin({
    name: "IntervalFilters",
    library: { type: "var", name: "peer"},
    filename: "remoteEntry.js",
    remotes: {},
    exposes: {
      './IntervalFilters': './src/IntervalFilters'
    },
    shared: {
      ...deps,
      react: {
        singleton: true,
        requiredVersion: deps.react,
      },
      "react-dom": {
        singleton: true,
        requiredVersion: deps["react-dom"],
      },
    },
  }),
```

Figure 11. Excerpts from the WebPack module federation configurations for the IntervalFilters micro-frontend.

We didn't encounter any shared dependency issues, while it was noted that for simple apps like the one of our case study prototype may create unnecessary complex configurations that would need to be maintained separately. Again, in the same line as most recent insights from industry adopters of microservice architectures, we wouldn't recommend a "micro-frontend first" approach for simple apps.

Conclusions

In this work, we explore micro-frontends, a new architecture in web application development that divides applications into small, independent modules. The research and case study implementation revealed several conclusions that were predominantly positive and promising.

One of the main advantages observed is the clear separation of responsibilities among developers and the division of the application into independent modules. Development teams can work autonomously on their respective modules, using preferred technologies and programming languages. This approach offers flexibility and freedom to adapt development according to team needs and preferences, while also improving code reusability. Independent modules allow the same functionalities to be used across different applications, reducing code duplication and enabling developers to leverage existing code without rewriting it. This results in faster development cycles and overall increased efficiency.

However, setting up the environment and architecture necessary to integrate these micro modules may not be worthwhile for small projects, either in terms of data volume or project complexity. In such cases, more time may be spent on package integration than on actual feature development.

Therefore, the choice of application architecture should always be carefully evaluated, considering available resources and the requirements of the project. Nevertheless, even for smaller projects, when micro-frontends are properly implemented by a professional team, they can bring a fresh and highly promising approach to development.

To strengthen the evidence base, future work should include empirical evaluations (preferably industry based) that examine not only the potential gains of micro-frontend-based architectures, but also the associated drawbacks and architectural trade-offs. These studies would help clarify when and how micro-frontends deliver measurable value.

References

Geers, M. (2020). Micro frontends in action. Simon and Schuster

Mezzalira, L. (2019, April 8). The “DAZN way.” Retrieved from

https://lucamezzalira.com/2019/04/08/adopting-a-micro-frontends-architecture/

Mezzalira, L. (2021). Building Micro-Frontends. O'Reilly Media, Inc.

Nielsen, C. D. (2015). Investigate availability and maintainability within a microservice architecture. *Aarhus: Department of Computer Science, Aarhus University*.

Pavlenko, A., Askarbekuly, N., Megha, S., & Mazzara, M. (2020). Micro-frontends: Application of microservices to web front-ends. Journal of Internet Services and Information Security, 10, 49–66

PraJwal, Y., Parekh, J. V., & Shettar, R. (2021). A brief review of micro-frontends. United International Journal for Research and Technology, 2

Schäffer, E., Mayr, A., Fuchs, J., Sjarov, M., Vorndran, J., & Franke, J. (2019). Microservice-based architecture for engineering tools enabling a collaborative multi-user configuration of robot-based automation solutions. Procedia CIRP, 86, 86–91

Yang, C., Liu, C., & Su, Z. (2019). Research and application of micro frontends. In IOP Conference Series: Materials Science and Engineering

Web Resources

AngularJS. (n.d.). Retrieved from https://docs.angularjs.org

Front-end vs Back-end Developer Difference. (2015). Retrieved from http://frontend-school.blogspot.com/2015/10/front-end-back-end-developer-difference.html

GeeksforGeeks. (n.d.). Monolithic vs Microservices architecture. Retrieved from https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/

Microservices (n.d.), Microservices.io. https://microservices.io/

Nikulina, O., & Khatsko, K. (2023). Method of converting the monolithic architecture of a front-end application to micro-frontends. Bulletin of National Technical University" KhPI". Series: System Analysis, Control and Information Technologies, (2 (10)), 79-84.

NPM. (n.d.). Retrieved from https://docs.npmjs.com/

NodeJS. (n.d.). Retrieved from https://nodejs.org/en/docs/

NextJS. (n.d.). Retrieved from https://nextjs.org/docs

O'Reilly. (n.d.). *Building Micro-Frontends*. Retrieved from https://www.oreilly.com/library/view/building-micro-frontends/9781492082989/ch04.html

React. (n.d.). Retrieved from https://react.dev/reference/react

Thoughtworks. (n.d.). Micro-frontends. Retrieved from

https://www.thoughtworks.com/radar/techniques/micro-frontends

VueJS. (n.d.). Retrieved from https://vuejs.org/guide/introduction.html

Webpack. (n.d.). Retrieved from https://webpack.js.org/concepts/module-federation/