Migrating stateful monoliths to Micro Frontends

1st submission

2nd of September, 2024

If we would define software as a composition of functionality expressed by code, architecture would be nothing more than a way for developers to explain and motivate the composition, interactions and hierarchy of these functionalities. (Sim, 2005)

1 Background, Context and Scope

The evolvability of architecture is essential to an application's survivability. The inability to effectively and reliable evolve your product means loss of business opportunities and potential stagnation of innovation (Hunyu Pei Breivold, 2012). Unfortunately, in most business cases, it is not feasible to just start a new project from scratch. This leads researchers to come up with strategies to improve the evolvability of existing codebases.

Historically, the main focus in the evolution of architectures was the level of dependency (coupling) in the code (Cătălin STRÎMBEI, 2015; Thomas Zimmermann, 2003). When architecture is mentioned in research, there is recurring theme expressed as: Architecture is only important for substantial codebases to justify the extra complexity it brings, where 'substantial' is measured in the amount of developers involved and codebase size (Sim, 2005; Thomas Zimmermann, 2003; Davide Taibi, 2020; Zateishchikov, 2023).

Following this trend, the newest development is decoupling the frontend into isolated components addressed as Micro Frontends (MFE). Research started around 2019 to further decouple frontend components after the evolution that changed stateful 'traditional' monoliths to stateless frontend/backends (Caifang Yang, 2019). By that time tooling was not readily available, which has changed completely over the last couple of years[1].

It is worth mentioning that a migration to MFE is not a 'silver bullet' solution (Mezzzalira, 2021). Smaller products might not gain any of the benefits of a decentralized codebase since the codebase (or team) itself is already small (Severi Peltonen, 2020). Therefore growing companies might initially neglect this architecture until the necessity occurs, but when it occurs, it might not feasible anymore for the company to migrate 'big bang'. This paper is dedicated to the reseach question of how a stateful monolith (Like Java Wicket, PHP etc.) can gradually migrate their frontend to a scalable decentralized architecture like MFE.

2 Problem Description

Considering the rapid growth of technology and user demands of new features, developers struggle to keep their growing codebases clear and understandable (Yosep Novento Nugroho, 2022). While companies start with monoliths (Cătălin STRÎMBEI, 2015) there are no ways to really decrease the codebase size except for removing code and thus functionality or abstracting away to libraries thus horizontally scaling the code (Biggerstaff, 1994). While other factors like the experience level of the maintainers and the quality of the codebase render it difficult to measure the correlated complexity, or even specify the connected implications of a complex codebase, research suggests that there is a strong correlation between code coupling and understandability for developers (Alenezi, 2016). A major downside of a monolithic application during runtime is the lack of flexibility and resilience. To be able to deploy new changes, the whole codebase requires to be redeployed and if something goes wrong, it could potentially result in downtime for the entire application (Cătălin STRÎMBEI, 2015). Furthermore, homogeneity in a monolithic approach limits the use of technologies or tools to the initially chosen language. If for some reason the support of the chosen language stops or the choice is made to use a different language, the team is forced to rewrite the whole codebase. This limits innovation preventing the usage of new tools (Severi Peltonen, 2020). Extensive research has been performed on microservices to mitigate these draw-

[1] Bit.dev, LitElement, Webpack5, single-spa etc.

backs (Grzegorz Blinowski, 2022) . The fragmented nature of small subservices allow for easy migration and isolated testing of new tools/frameworks without affecting the other services. To be able to decouple the frontend the same way as microservices did for the backend, some differences and extra complications should be take into account (Rodrigo Perlin, 2023).

2.1 Approaches to solve the problem

The evolution of MFE started with html fragments and a way to aggregate microservices using a Gateway API[2] or Backend for Frontend (BFF)[3] design patterns on server level as described by Harms et al. (Holger Harms, 2017). Tilak et al. (P Yedhu Tilak, 2020) stated that after the wave of micro-services, the next bottleneck in line were the monolithic frontends. Ferracaku (Ferracaku, 2021) presented in his Msc. thesis around 2021 the state of micro frontends accompanied by a small experiment on how developers perceive the new paradigm. The results suggested that even though tooling was in a primitive state, the technique sounded promising for large projects. The results of the experiment also implied that the best way to implement MFE is to migrate from an existing Singe Page Application (SPA).

In an attempt to standardize MFE, some books (Geers, 2020; Rappl, 2021; Mezzzalira, 2021) have been written that define which options are available to setup a micro frontend environment: The first choice to make is whether to go for a horizontal or vertical split as can be seen in Fig 1.

Horizontal split means composing the webpage by injecting grouped functionality as an isolated (micro) frontend in a 'shell' application while vertical means creating a component per ‘route' and thus injecting a (micro) frontend based on the URL. The next step is to choose where to compose the MFE. Peltonen et al (Severi Peltonen, 2020) presented the different ways to create MFE, mainly the difference between client-side, server-side and edge-side composition (Fig. 2). Another product of their research was a list of motivations, benefits and issues regarding MFE in general. Finally, Petcu et al. (Adrian Petcu, 2023) highlighted the difference between a horizontal split architecture and vertical split architecture, complimented by a small performance analysis between a monolith, module federation and IFrames solution. It is important to note that the application used in this benchmark was very simple and small.

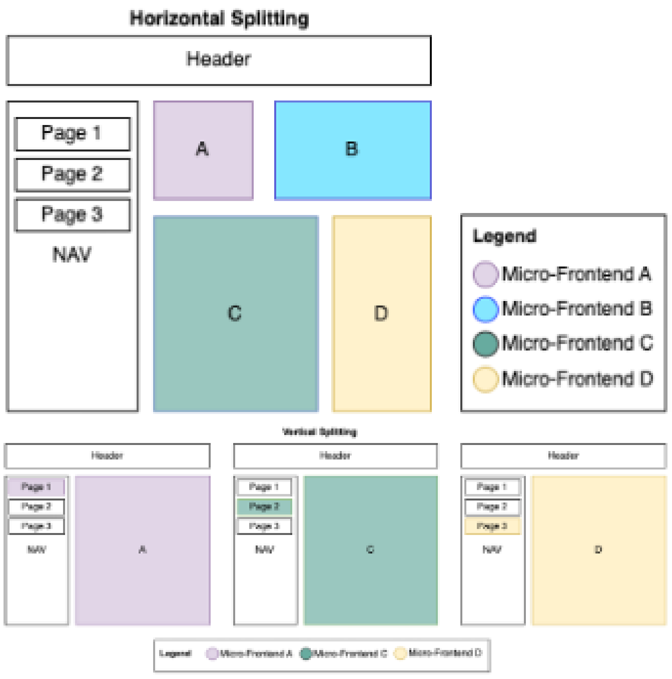

Figure 1: *Horizontal vs Vertical splitting*

Therefore the test fails to emphasize the potential benefits of MFE when used in bigger applications. Kaushak et al. (Neha Kaushik, 2024) performed the experiment again using a bigger application. the results suggest a slight improvement of performance on heavy usage although it is still hard to compare the 2 setups without seeing the codebases.

While research advices to start using micro frontends whenever the codebase is of considerable size[4] or whenever multiple teams work on the same codebase to see the real benefits (Adrian Petcu, 2023; Zateishchikov, 2023; Severi Peltonen, 2020), Männistö et al. (Jouni Männistö, 2023) suggested that potential benefits could already be achieved on smaller codebases. A small custom framework was developed leveraging the library LitElement[5] to load isolated web components in the shell application based on which tenant was signed in. Perlin et al (Rodrigo Perlin, 2023) showed a way to load different frontend frameworks like React, Angular and Vue into a single shell application using Webpack[6]. Zateischchikov developed a guide on how to rewrite your SPA to a MFE which also leveraged Webpack. Most of these solutions could be defined as “client-side composition” with the exception of Männistö since it is not clear what technique is used for the shell, the use of module federation however, suggests client-side composition.

[2]https://microservices.io/patterns/apigateway.html
[3]https://bff-patterns.com/
[4]The term “considerable size” is left unquantified on purpose because no research has been able to reliably define and quantify big codebases, rendering the term subjective.
[5]https://lit.dev/
[6]https://webpack.js.org/concepts/module-federation/

The benefits of modularization have not yet been reliably proven in current MFE research. However Martini et al presented a way to quantify the benefits of modularization which could benefit the sustainability of a codebase. (Antonio Martini, 2016, 2018) A nuance to be addressed is that modularity can be achieved within monolithic applications. The MFE architecture however, enforces modularity by breaking the code up into multiple sub-applications which reduces the possibility of coupling dramatically.

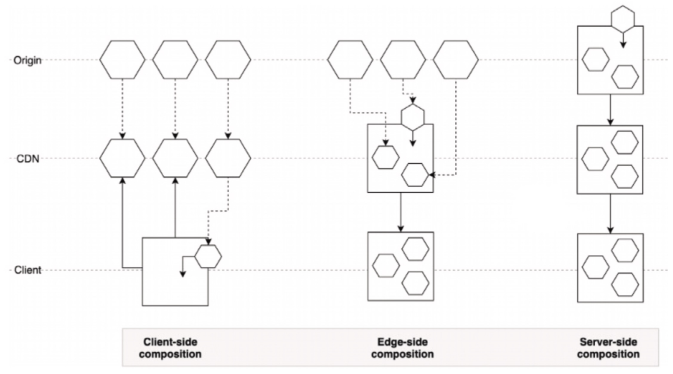

Figure 2: *Composition types*

Kaushik et al. (Neha Kaushik, 2024) used machine learning to measure the response time and throughput in an attempt to quantify the resource benefits of MFE. This is hard to proof considering that a drawback of MFE is potential increased code duplication resulting in an increase of resource usage (Zateishchikov, 2023). To mitigate these drawbacks, load-time optimization can be achieved by premature composition of the web components at CDN or server-side level (Severi Peltonen, 2020). Combined with caching this can significantly improve the load times. This optimization can be taken a step further by applying Server Side Rendering (SSR) to pre-render MFE on the server. Borello (Borello, 2024) dedicated his thesis to researching the use of SSR in combination with React server components[7] which could allow component-level SSR. However this technique was not mature enough to be used in a corporate environment, which is why he neglected the technique in his experiment. For server-side composition, it is worth mentioning that in early research around 2021, an old technology called Server-side includes (SSI) is mentioned as a way to compose micro frontends at server level while never used in later research. Perhaps because this technique is unfortunately not without security risks. SSI could potentially be exploited by malicious users, which should be taken into account in further research (Muhammad Zulkhairi Zakaria, 2021).

2.2 Gaps in Research

Since current research is focussed on migrating from a stateless SPA frontend monolith to stateless MFE (Jouni Männistö, 2023; Rodrigo Perlin, 2023; Adrian Petcu, 2023). This paper aims to close the gap on migrating from a stateful monolithic architecture to a stateless MFE architecture.

What sets this research apart is the focus on server-side composition, SSR and Isomorphic components. SSR and Static Site Generation (SSG) are mentioned in research but hardly used in the experiments (Andrej Simeunovic, 2023; Tokuc, 2023; Borello, 2024; Zateishchikov, 2023). The given motivation was 'lack of time and available resources'. This renders the topic valuable for further research since SSR could potential be very beneficial for monoliths that are SSR like PHP, Ruby or Java Wicket (Taufan Fadhilah Iskandar & Lubis, 2020). Recent developments within the field of MFE like React server components, Angular's focus on improving SSR (*Introducing Angular v17*, 2023). new task automation updates from Nx[8] and the inclusion of module federation in Vite[9] allow for micro frontend setups that include hydrated[10] SSR frontends which could proof useful for this particular use-case.

3 Research Question

RQ1: Is it possible to integrate SSR Micro Frontends in a stateful Monolithic application?

RQ2: Are there any performance benefits of using SSR Micro Frontends over the baseline CSR Micro Frontends in stateful Monolithic applications.

3.1 Goal

The goal of this research paper is to explore the potential benefits of SSR in Micro Frontends. In previous research, the main focus was put on migrating Single Page Applications to Micro Frontends but currently no research backs the edge case of having a stateful monolith. To find the optimal setup for this usecase, the server-side composition setup will be benchmarked against the common practice setup (client-side composition).

3.2 Null Hypothesis

Micro Frontends is a technique used gradually after migrating your stateful monolithic application to a stateless backend, preferably with micro services, and

[7]https://react.dev/reference/rsc/server-components
[8]https://nx.dev/concepts/inferred-tasks
[9]https://github.com/originjs/vite-plugin-federationc
[10]https://angular.io/guide/hydration

a separate frontend. A migration to Micro Frontends requires some sort of JavaScript shell application that manages injection and cohesion.

3.3 Alternative Hypothesis

When migrating a Java Wicket monolith to Angular micro frontends, if micro frontends are injected directly in a stateful monolith shell using server-side composition then the removal of a JavaScript shell application combined with prerendered components will improve the overall performance of the application.

4 Design and Implementation

The setup will consist of a stateful SSR monolith (Java Wicket) as baseline application referred to as the 'shell', to maintain consistency with current research and the community the tractor-store[11] was chosen as example. And a Monorepo MFE setup referred to as the 'Framework' that supports building, packaging and deploying (Angular) Micro Frontends to be integrated in the baseline application. The project setup will use the MFE Decision Framework invented by Mezzalira to discover which path is most feasible for this particular Use Case (Mezzzalira, 2021). Although the chosen techniques (Java Wicket and Angular) are predetermined, the choice of technology is rather trivial. The focus will be on the techniques and the overal use-case of integrating (SSR) MFE in a stateful Monolith.

In the research phase, frameworks like bit.dev, simple-spa, Lerna, openComponents, Piral, Qwik, Nx and similar Monorepo frameworks are to be explored. An important design principle for MFE is uniformity to create a smooth developer experience (Davide Taibi, 2020). Monorepos are proven to benefit fragmented multi-application setups like Google because code and dependencies from (internal) libraries is easier to find and manage (Ciera Jaspan, 2018). Vite and Webpack will be used in the benchmark to research the benefits of sharing common modules and the use of module federation, a technique to load JavaScript modules from external sources.

The components will have to be able to communicate to the 'shell' (baseline app) and each other. Therefore a reliable way of communication has to be researched, there are a couple of ways of establishing this using the 'shell' application (Mezzzalira, 2021). However a constraint of the current scope is to minimalize coupling on the shell application which is why a custom solution should be explored.

To be able to deploy Micro Frontends in a non-javascript 'shell', A method or technique should be researched that takes care of the integration of the MFE. Web components provide a technology agnostic way of loading JS applications in a web application. Furthermore it is possible to write a minimalistic JS wrapper that uses Module Federation to fetch and integrate the Micro Frontends. Finally, a brand new open source project called Picard.js[12] will be explored to load MFE in the baseline application. All mentioned tools will be compared to find the best setup for this particular use-case. Current research is investing in standardizing Frontend Discovery which will also be included in the integration part[13].

Additional research will have to be performed on combining SSR with Web components. The combination of isomorphic Web components, (JavaScript components that can be rendered server-side as well as client-size), started off a bit unstable (Lorenz, 2020). Development on Isomorphic rendering (Karolina Kowalczyk, 2024) and SSR with Hydration has improved over the years with for example Angular 17. The choice to include the use of isomorphic rendering is to figure out if micro frontends can be created initially as SSR, but migrated to CSR in a later stadium, allowing for more freedom in future development. Once the components are rendered on the server, the viability of server-side composition in combination with these SSR components will be researched because of concerns of scalability on server-side composition setups (Severi Peltonen, 2020).

Another potential concern of this particular experiment is the combination of stateless MFE in a stateful shell. A solution has to be discovered to transfer the stateful context to a MFE that has no access to the state. A common solution is to write an isolated Micro Service complementing the MFE. Writing Micro Services however exceeds the scope of this experiment. Therefore in this setup a small wrapper acting as a BFF will suffice to handle REST requests from the MFE.

Once the framework is developed, the framework will be benchmarked against a “client-side composition” setup. Since components or rather sections from the baseline architecture will be extracted using Domain Driven Design (DDD) (Evans, 2004) in the future, a mock functionality will be developed in both setups. The main idea of DDD is to draw boundaries around aggregated components that contain similar business-functionality. That means domains like for example ‘account management’ can potentially be transformed to micro frontends, isolating functionality based on ‘similarity’. The same technique has

[11]https://micro-frontends.org/tractor-store/
[12]https://picard.js.org/
[13]https://github.com/awslabs/frontend-discovery

been used a lot over the last couple of years to gradually migrate monolithic backends to micro services (Victor Velepucha, 2023).

Finally, a uniform simple plug-and-play methodology to inject these SSR micro-frontends will be researched with as product the 'Framework' setup. The conditions are; minimal change in the host application and minimal interference with the current baseline application. The implementation should feel seamless and should be easy to use for developers of similar use-cases. MFE are proven to increase complexity during implementation (Ferracaku, 2021). A poor developer experience will create resistance and thus lack of support as suggested by the results of (Fabio Antunes, 2024) rendering the benefits of MFE practically useless. Although the specifics have yet to be explored, the implementation will consist of a Java Wicket wrapper component that is able to load a micro frontend in the monolith. The MFE, provided by a Node server or CDN, require an uniform way of being initialized and injected. This is possible using a settings JSON file (Rappl, 2021) through html properties in the component (Mezzzalira, 2021) or a novel implementation like using JavaScript signals[14].

5 Evaluation

Due to the significant differences in the paradigms SSR and CSR, server, network and browser load should be taken into account. It is important to note that Hydration and Isomorphic components could blur the lines between what is considered CSR and SSR, hence these techniques will be incorporated in the server-side composition setup. The main idea is to fill the gaps of creating a setup that renders and composes the micro frontends on the server which has not been done before (Karolina Kowalczyk, 2024).

The addition of rendering components on the server-side does mean that additional computation will be moved to the server instead of the browser. Therefore it is unfair to only measure the load on the browser. After creation of the BFF micro-service, a monitoring tool like Grafana will be used to visualize the difference in load on the server to accommodate for the server-side part of the benchmark.

The browser part will be measured browser tools like lighthouse and Core Web Vitals in order to quantify the user experience (Roy Hanafi, 2024). By leveraging predefined metrics like Largest Contentful Paint (LCP) and First Contentful Paint (FCP), the differences in the rendering techniques could be compared and bechmarked.

The alternative hypothesis is accepted if the JavaScript bundle size is decreased or if the average FCP/LCP metric improved using a threshold of 10%. Server CPU and RAM differences will be presented in the benchmark and comparison to highlight the benefits of both paradigms. Because of time limitations, it is impossible to test the performance differences in a big application, therefore the threshold was intentionally lowered to accomodate for the resource overhead that MFE produce.

[14]https://github.com/tc39/proposal-signals

6 Activities

The outline of this experiment will consist of 4 phases: Research, Development, Benchmarking and Evaluation. Fig 3. Shows these phases including the subproducts to be delivered at the end per phase. The document will be outlined and initialized in the first week and maintained during the process.

In the final weeks during the conclusions, the document will be revised. The expected products to be delivered are a framework that shows how to migrate a trivial SSR Monolith to Micro Frontends (using Angular) with a complementary benchmark providing detailed insight in if and when this method is beneficial.

References

Adrian Petcu, D. A. S., Madalin Frunzete. (2023). Benefits, challenges, and performance analysis of a scalable web architecture based on micro-frontends. In *U.p.b. sci. bull., series c, vol. 85, iss. 3* (pp. 319–334). University Politehnica of Bucharest.

Alenezi, M. (2016). Software architecture quality measurement stability and understandability. In *International journal of advanced computer science and applications, vol. 7, 2016* (pp. 550–560). thesai.org.

Andrej Simeunovic, U. T. (2023). *Managing micro frontends across multiple tech stacks - sharing, finding & publishing* (Unpublished master's thesis). PLTH — LUND UNIVERSITY.

Antonio Martini, N. M., Erik Sikander. (2016). Estimating and quantifying the benefits of refactoring to improve a component modularity: A case study. In *42th euromicro conference on software engineering and advanced applications* (pp. 92–100). IEEE.

Antonio Martini, N. M., Erik Sikander. (2018, January). Software architectures – present and visions. In *Information and software technology vol. 93* (pp. 264–279). Elzevier.

Biggerstaff, T. J. (1994). The library scaling problem and the limits of concrete component reuse. In

Proceedings of 1994 3rd international conference on software reuse (pp. 102–109). IEEE.

Borello, D. (2024). *Micro frontends, server components and how these technologies can provide a paradigm shift with architectural changes in modern enterprise web app development* (Unpublished master's thesis). POLITECNICO DI TORINO.

Caifang Yang, Z. S., Chuanchang Liu. (2019). Research and application of micro frontends. In *Iop conference series: Materials science and engineering vol. 490, iss. 6* (pp. 1–6). IOP.

Ciera Jaspan, A. K. C. S. E. K. S. C. W. E. M.-H., Matthew Jorde. (2018). Advantages and disadvantages of a monolithic repository. In *40th international conference on software engineering: Software engineering in practice* (pp. 255–265). ACM/IEEE.

Cătălin STRÎMBEI, R.-M. S. A. N., Octavian DOSPINESCU. (2015, April). Software architectures – present and visions. In *Informatica economică vol. 19, no. 4/2015* (pp. 13–27). AL.I.Cuza University.

Davide Taibi, L. M. (2020, April). Micro-frontends: Principles, implementations, and pitfalls. In *Acm sigsoft software engineering notes, volume 47, issue 4* (pp. 25–29). ACM.

Evans, E. (2004). *Domain-driven design: Tackling complexity in the heart of software.* Addison-Wesley Professional.

Fabio Antunes, M. A. P. A. D. T. M. K., Maria Julia Dias Lima. (2024). Investigating benefits and limitations of migrating to a micro-frontends architecture. *https://arxiv.org/*.

Ferracaku, J. (2021). *Sthe state of micro frontends: Challenges of applying and adopting client-side microservices* (Unpublished master's thesis). University of Oulu.

Geers, M. (2020). *Micro frontends in action.* Manning.

Grzegorz Blinowski, A. P., Anna Ojdowska. (2022, February). Monolithic vs. microservice architecture: A performance and scalability evaluation. In *Ieee access, vol. 10* (pp. 20357–20374). IEEE.

Holger Harms, L. L. I., Collin Rogowski. (2017). Guidelines for adopting frontend architectures and patterns in microservices-based systems. In *11th joint meeting on foundations of software engineering* (pp. 902–907). ACM.

Hunyu Pei Breivold, M. L., Ivica Crnkovic. (2012, January). A systematic review of software architecture evolution research. In *Information and software technology, vol. 54, 2012* (pp. 16–40). Elsevier.

Introducing angular v17. (2023). `https://blog.angular.dev/introducing-angular-v17-4d7033312e4b`. (Accessed: 2024-05-12)

Jouni Männistö, M. R., Antti-Pekka Tuovinen. (2023). Experiences on a frameworkless micro-frontend architecture in a small organization. In *Ieee 20th international conference on software architecture companion* (pp. 61–67). IEEE.

Karolina Kowalczyk, T. S. (2024, January). Enhancing seo in single-page web applications in contrast with multi-page applications. In *Ieee access vol. 12* (pp. 11597–11614). IEEE.

Lorenz, D. (2020). *A deep analysis into isomorphic, autonomous cross-framework usage microfrontends.* `https://itnext.io/a-deep-analysis-into-isomorphic-autonomous-cross-framework-usage microfrontends-364271dc5fa9`. (Accessed: 2024-05-12)

Mezzzalira, L. (2021). *Building micro-frontends: Scaling teams and projects, empowering developers.* O'Reilly UK Ltd Clockhouse Dogflud Way Farnham, GU9 7UD: O'Reilly.

Muhammad Zulkhairi Zakaria, R. K. (2021). Risk assessment of web application penetration testing on cross-site request forgery (csrf) attacks and server-side includes (ssi) injections. In *International conference on data science and its applications (icodsa)* (pp. 79–85). IEEE.

Neha Kaushik, V. R., Harish Kumar. (2024, April). Micro frontend based performance improvement and prediction for microservices using machine learning. In *Journal of grid computing, vol. 22, art. 44* (pp. 1–26). Springer.

P Yedhu Tilak, S. D. D. N. B., Vaibhav Yadav. (2020). A platform for enhancing application developer productivity using microservices and micro-frontends. In *Ieee-hydcon* (pp. 1–4). IEEE.

Rappl, F. (2021). *The art of micro frontends.* Packt.

Rodrigo Perlin, V. M. G. D. A. M., Denilson Ebling. (2023). An approach to follow microservices principles in frontend. In *Ieee 17th international conference on application of information and communication technologies (aict)* (pp. 1–6). IEEE.

Roy Hanafi, N. A., Abd Haq. (2024, January). Comparison of web page rendering methods based on next.js framework using page loading time test. In *Vol. 13 no. 1 (2024): Maret 2024* (pp. 102–108). Teknika.

Severi Peltonen, D. T., Luca Mezzalira. (2020, August). Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review. In *Information and software technology, vol. 136* (pp. 1–19). Elsevier.

Sim, S. E. (2005). A small social history of software architecture. In *13th international workshop on program comprehension* (pp. 341–344). IEEE.

Taufan Fadhilah Iskandar, T. F. K., Muharman Lubis, & Lubis, A. R. (2020). Comparison between client-side and server-side rendering in the web development. In *Iop conference series: Materials science and engineering vol. 801* (pp. 1–7). IOP.

Thomas Zimmermann, A. Z., Stephan Diehl. (2003). How history justifies system architecture (or not). In *Sixth international workshop on principles of software evolution* (pp. 73–83). IEEE.

Tokuc, K. (2023). *Suitability of micro-frontends for an ai as a service platform* (Unpublished master's thesis). University of Applied Science Hamburg.

Victor Velepucha, P. F. (2023, July). A survey on microservices architecture: Principles, patterns and migration challenges. In *Ieee access vol. 11* (pp. 88339–88358). IEEE.

Yosep Novento Nugroho, M. J. A., Dana Sulistyo Kusumo. (2022). Clean architecture implementation impacts on maintainability aspect for backend system code base. In *10th international conference on information and communication technology (icoict)* (pp. 134–140). IEEE.

Zateishchikov, K. (2023). *Scaling a software platform using micro frontends* (Unpublished master's thesis). VAASAN AMMATTIKORKEAKOULU UNIVERSITY OF APPLIED SCIENCES.

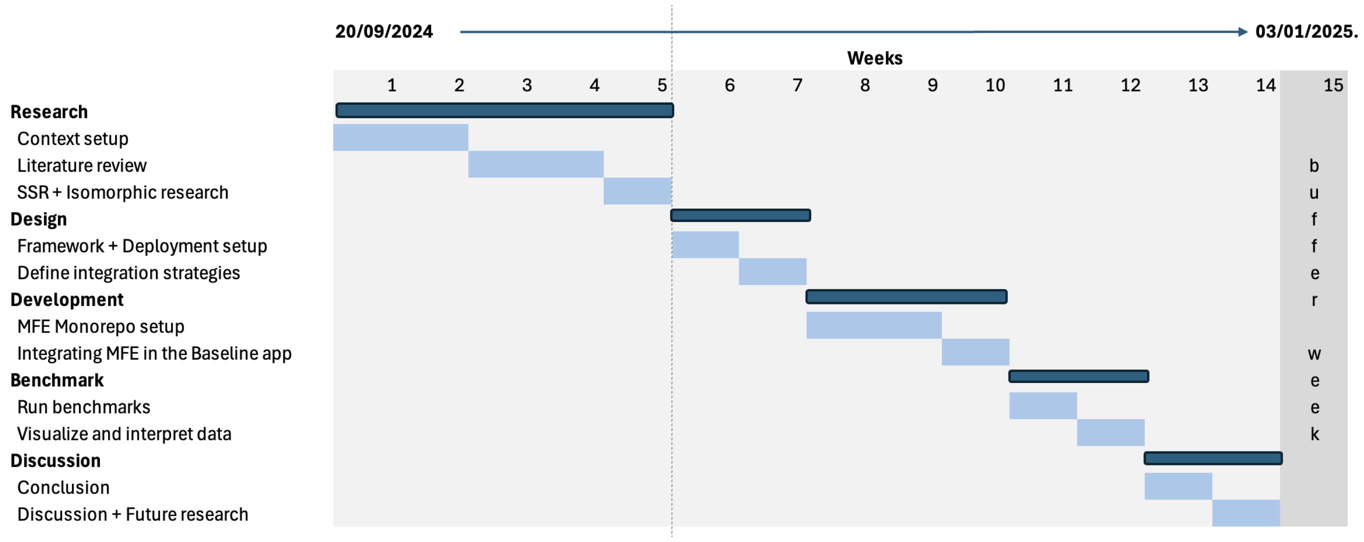

Figure 3: *Gantt chart*