ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 9, No. 1, 2024

FEATURES OF THE IMPLEMENTATION OF MICRO-INTERFACES IN INFORMATION SYSTEMS

Oleksandr Stepanov, Halyna Klym

Lviv Polytechnic National University, 12, Bandera Str, Lviv, 79013, Ukraine.
Authors' e-mail*: oleksandr.v.stepanov@lpnu.ua, halyna.i.klym@lpnu.ua*

https://doi.org/10.23939/acps2024.01.054

Submitted on 18.04.2024

***Abstract*: Microservices are a software development technique, or variant of the service-oriented architecture structural style, that organizes an application as a collection of loosely coupled services. The purpose of the work is to study the methodology for the design and implementation of information systems using micro-interfaces in order to improve the quality and speed of their development and facilitate their use. The article proposes a method of transforming the software system architecture from monolithic to microservice architecture. A brief review of existing architecture reengineering research has been provided and the advantages of a microservice approach have been identified. At the second stage, a transition to a modular architecture with the allocation of functionality into separate modules has been proposed. An experiment with a typical external single page application demonstrates the performance of the proposed algorithm.**

***Index Terms*: architecture, interface, micro-frontend, microservices, monolithic structure, software applications**

I. INTRODUCTION

Until recently, the absolute majority of software applications has not differed in the particular complexity of their development and the complexity of their software content. However, in recent years, the sophistication of software creation in general and the relative cost of its development have begun to rise rapidly, which has had a negative impact on the composition of software in general. This impact is an increase in complexity, which in turn is reflected in the complexity of the user interface and, as a result, the tools are required for its development.

Over time, the amount of JavaScript code used in applications increases, which leads to an increase in the following parameters in most projects: development time due to the high level of code complexity, time of testing, time interval between application releases.

Large applications that have these problems are often referred to as monoliths because of the architectural approach used.

Monolithic architecture is an architectural approach in which all the main logic of the program is collected in one place as a whole component, in other words, monolithic software consists of a single-layer combination of various components into a single whole [1,2].

This trend necessitated the emergence of a concept that would be able to simultaneously increase the functionality of applications and make the development process easier. Such a concept appeared, and it acquired the name micro-frontend, apparently for front-end development, and the concept of microservices, in turn, in the field of back-end development. The principle of a micro-frontend is to break large monolithic software applications into smaller, independently deployed units called micro-interfaces, each of which is responsible for a specific function, but is nevertheless part of one larger system. By decoupling the interface components, this architecture enables development teams to work independently on different parts of the application, contributing to faster development cycles and easier maintenance [3-5].

Within the scope of this study, the main focus will fall on ways of implementing the concept of micro-frontends in the process of designing and implementing information systems. As such, in digital terms, they represent electronic systems – sets of interconnected components that collect, process, store and disseminate data to support decision-making and organizational operations. In connection with the expected scale of these systems, the implementation of a technique aimed at offloading software components may prove to be the right solution and contribute to more efficient processing and distribution of data [6].

The main goal of this work is to propose an optimal micro-interface architecture that combines the advantages of different architectural approaches.

II. LITERATURE REVIEW AND PROBLEM STATEMENT

In the scientific and research space of today, there are works dedicated to the invention and analysis of the methodology for the design and implementation of information systems using micro-interfaces in order to improve the quality and speed of their development and facilitate their use.

Research [7] was aimed at developing a method of applying the micro-interface approach for monolithic single page applications (SPA). The article proposes a method of transforming the software system architecture from monolithic to microservice architecture (microservice architecture - MSA). The proposed three-stage method differs from the methods by the allocation of an additional stage of transformation, which allows to gently

change the connections between the parts of the monolithic application, which were implemented in the initial monolithic architecture [8,9]. The first stage is reverse engineering, in which it is proposed to shift the focus from the search for obsolete code to the functional analysis of the program as such. At the second stage, a transition to a modular architecture with the allocation of functionality into separate modules is proposed. At the end of the third stage, several separate programs (micro-interfaces) were obtained, which are connected to the main program [8]. An experiment with a typical external SPA demonstrates the performance of the proposed algorithm. The resulting system is compared to the original on the following measurable parameters: production build time, master build size, and average first page load time. All comparisons showed the advantages of the system obtained as a result of the transformation. As a result, the architecture transformation algorithm made it possible to obtain a better result, taking into account the limitations of the SPA interface.

However, taking into account the above-mentioned scientific documentation, the issue related to the methodology for the design and implementation of information systems using micro-interfaces still remains insufficiently researched and requires further elaboration.

III. SCOPE OF WORK AND OBJECTIVES

The purpose of the work is to study the methodology for the design and implementation of information systems using micro-interfaces in order to improve the quality and speed of their development and improve the user experience.

IV. MICROSERVICE ARCHITECTURE FROM THE FRONTEND PERSPECTIVE

Microservices is a software development technique, or variant of the service-oriented architecture (SOA) structural style, that organizes an application as a collection of loosely coupled services. In the microservice architecture, the components themselves are isolated from each other and interact using interfaces. From the above, it can be concluded that each microservice must be an independent component. Since each microservice works in its own process, it must clearly define the interaction interface (API) with it. Other vertices can interact with the service only through the API, so minimizing connections is one of the most important processes in planning such an architecture [10].

The main advantages of the micro-frontend approach to the development of large applications include its modular architecture, where individual widgets or pages function as completely independent programs. This modularity enhances the speed of testing, as changes in one widget or page can be tested in isolation within the application, eliminating the need for testing all other functionalities. Additionally, the micro-frontend approach facilitates parallel deployments, allowing individual widgets or pages to be deployed independently for greater efficiency.

In addition to the obvious advantages of this approach, it also has significant disadvantages. Duplication of code - each application is developed by a separate team that makes its own technical decisions. This leads to re-downloading the same frameworks libraries and general duplication of code that could have been reused. The JS bundle of a monolithic application will always be smaller than the set of bundles in the micro-frontend architecture, there may be possible problems with caching and versioning of applications. Global variables or CSS styles are things to forget about in a micro-frontend architecture if applications are completely isolated.

Using this architectural approach on small projects and in small teams brings more challenges and additional development complexity than benefits. However, large projects with distributed teams, on the contrary, benefit more from creating micro-frontend applications. That is why today micro-frontend architecture has been already widely used by many large companies in their web applications [11]. Fig. 1 shows migration schema from monolithic to micro-frontends.

Currently, there are several options for providing a micro-frontend architecture. The easiest way to organize micro-frontends is to create several independent applications. This requires having one main application with hyperlinks to other micro-frontend applications. Clicking on the hyperlink takes the user to another application with a different URL.

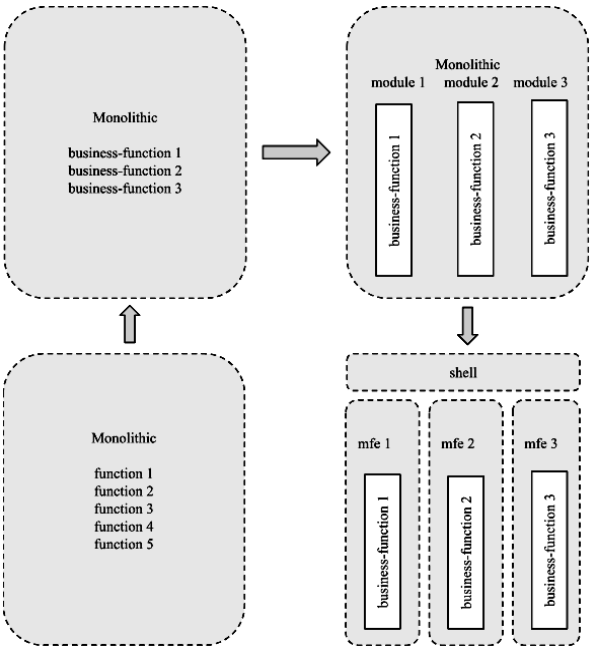

Fig. 1. Architecture when transitioning to a micro-frontend

Another, often used option is to use the single-spa framework. The idea is to create a framework-specific shell for each micro-frontend application to integrate them into a one single-spa application. The main disadvantage of this approach is the need to follow clear rules of the single-spa framework for each micro-frontend in order to

organize integration with other micro-frontends. If there is a ready-made application, then there is a risk of error and it should be rewritten taking into account the rules of single-spa.

One of the most popular mechanisms is the use of iframes. All necessary widgets must be placed in I-frames, which load the corresponding micro-frontend, which is placed on a separate hosting. Data exchange between them is carried out using a POST message. The main disadvantage of this approach is the need to download the full microprocessor package. Another significant disadvantage is the risk of reloading libraries using microprocessor packages.

The most modern way of working with micro-frontends is to use the function of combining modules in the Webpack module bundler. This approach ensures both good communication of micro-frontends and the possibility to avoid the duplication of code. The main idea of the approach is to configure the shell program to import only the necessary module from the micro-frontend program.

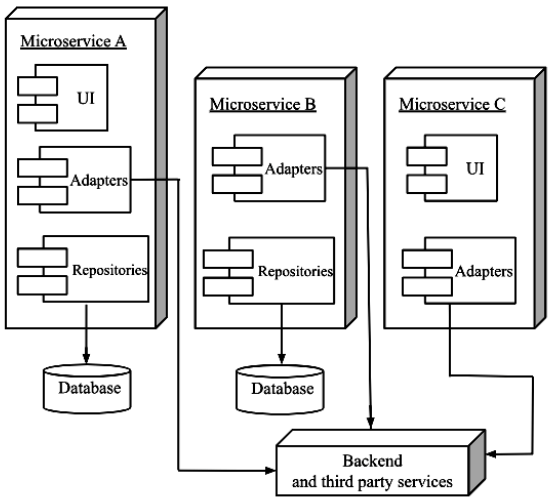

Fig. 2. The scheme of the test environment in the microservice architecture

In information system performance testing, the following directions are distinguished: load, stress, stability testing, failure and recovery testing, volume testing, and spike testing [1]. Fig. 2 shows possible integrated test environment implementation. These types of testing mainly differ in the purpose they aim to achieve.

Load tests are designed to determine the speed, scalability and stability of the system. Scalability describes the ability of the system to adapt to increasing workloads by using additional resources. Load tests evaluate behavior of the system under normal and peak load conditions. Stress testing evaluates the behavior of the system under loads that significantly exceed the expected maximum. Stability testing checks whether the system can withstand the expected load for a long time.

Volumetric testing is performed with an increase in the amount of used data that is stored and used in the application. Spike testing is performed suddenly, increasing the load for a short time, created by a very large number of users, and observing the behavior of the system. Regardless of the type of stress testing, there are different levels at which performance testing can be performed.

V. PRESENTATION OF THE MAIN MATERIAL OF THE RESEARCH

A. THE PROS ANS CONS OF MFE DEVELOPMENT

Understanding the main advantages of the micro-frontend approach in the development of large enterprise applications plays a fundamental role in fully realizing the opportunities for further development of software applications provided by the implementation of the micro-frontend concept modular architecture. Individual widgets or pages are completely independent programs with a speed of testing. Changes in one widget or page can be tested separately and only in this application without wasting time testing all other functions in parallel deployment. Individual widgets or pages can and should be deployed independently.

However, like all currently known conceptual technologies, in addition to the obvious advantages of the studied approach, it also has significant disadvantages increasing the overall complexity of the program duplication of code - each application is developed by a separate team that makes its own technical decisions. This results in re-loading the same frameworks libraries and general duplication of code that could have been reused. The JS package of a monolithic application will always be smaller than the set of packages in the micro-interface architecture, there may be possible problems with caching and program versioning impossibility of using global variables or CSS styles due to incomplete isolation of programs.

A big role in the issue of integration of this concept in the development of the application is played by the direction of the latter and the initial estimated volume of development. Using this architectural approach for small projects and small teams turns out to be more problematic and adds more development complexity than benefits. On the other hand, large projects with distributed teams, on the contrary, get more benefits from creating micro-interface applications. For this reason, the micro-frontend architecture is currently widely used by many large companies in their software applications.

At the heart of micro-interfaces, from the point of view of their influence on the developed product, there are the following fundamental components independent of the technological stack Teams must decide on their own set of technologies without any influence from other code isolation which consists in the fact that the execution time should not be shared and the programs should be autonomous. All parts that cannot be isolated must be prefixed to avoid collisions of identical identifiers taking advantage of the native browser API to develop non-standard ways of communicating between applications stability of the developed function which consists in the possibility of its operation autonomously without JavaScript.

B. HORISONTAL & VERTICAL SEPARATION

However, the implementation of this architectural style is associated with a number of nuances that should be given due attention. First of all, it is necessary to decide how the micro-frontend will be identified, it is possible to have several Micro-Frontends in one view (horizontal distribution) or only one micro-frontend loaded per unit of time (vertical distribution), in which case it is represented by SPA or by one representation exported as an HTML or JavaScript file. Fig. 3 shows horizontal separation architecture approach with multiple micro-frontends. Horizontal distribution better serves static pages such as catalogs or e-commerce, instead of more interactive projects such as a surface streaming platform or internal applications that require vertical distribution.

C. ORCHESTRATION

The next important aspect of implementing micro-frontends is orchestration - effectively managing the loading and unloading of micro-interface modules based on user interaction or application requirements [12]. Fig. 4 shows architecture with vertical separation, each micro-frontend loads on separate route. When horizontal distribution is adopted, it is possible to assemble the server-side view by creating the final HTML page with HTML fragments, otherwise using the Edge side includes markup language, which uses the concept of transclusion, where the placeholder is re-placed for valid HTML tags.

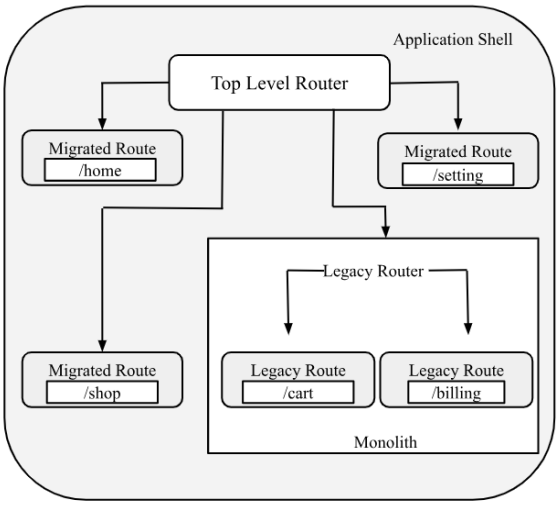

Fig. 3. Routing mechanism implementation example with migration from monolith to vertical architectural approach

Switching occurs at the edge via a CDN provider or at the edge using a Varnish reverse proxy. The main disadvantage of this approach is that the ESI specification is not fully available in every CDN provider, often only a subset of the specification is present, making this technique difficult to use. In the case of vertical distribution, the most common way to load a micro-frontend is to use a shell application that is responsible for loading each micro-frontend one at a time. Technically, the application shell is represented by an HTML file that is always present during the user's session and contains a small JavaScript library that is used to load various micro-frontends and orchestrate view changes.

Routing – developing mechanisms to manage routes and decide which micro-interface program to load based on a requested URL or user interaction. Isolation – micro-frontend modules work independently, but they must also coexist in the same environment without causing conflicts or collisions. Fig. 3 shows partly migrated monolithic application to micro-frontend with separate routing.

D. MFE COMMUNICATION

Communication - establishment of effective communication channels between micro-interfaces while maintaining weak communication. Both vertical and horizontal distribution require a way to share data. Web storage or cookies can be used as client-side approaches to manage communication. In the case of horizontal distribution, an event emitter can be applied, which is introduced into each micro-interface. Fig. 4 shows possible approach example of how to share data between different micro-frontends with event emitter or etc.

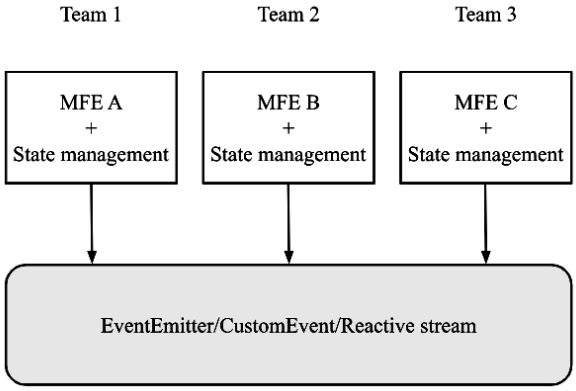

Fig. 4. Methods of data transmission between micro-frontend entities

With this approach, each micro-interface will be completely independent from the others, which reduces coupling and enables independent deployment. When a micro-interface emits an event, other micro-interfaces subscribed to that particular event respond appropriately. In the case of vertical distribution, it is important to understand how to exchange information between micro-interfaces. Both horizontal and vertical approaches need to consider how views interact as they change. It is possible that variables can be passed via the query string or by using a URL to transfer small amounts of data. In addition, you can use web storage to store information temporarily (session storage) or permanently (local storage) for sharing with other micro-interfaces.

Additionally, consider the importance of UI/UX consistency – managing common styles, layouts, and interactions to ensure a cohesive and intuitive user experience; and dependency management – the micro-interface

architecture can lead to duplicate dependencies when the same library or resource is loaded multiple times in different modules. This can lead to increased page load times and unnecessary resource consumption.

I. INTEGRATION METHODS

In the space of front-end application development, there are currently a number of methods related to the implementation of the idea of micro-interfaces. Each offers its own trade-offs for flexibility, performance, complexity, and developer experience. The choice of method depends on factors such as specific project requirements, desired level of isolation between micro-interfaces, available infrastructure and tools.

Method of integration during software development. The principle of this approach is mostly that each micro-interface is built separately into a separate bundle during the development process. The bundles are then integrated into the shell of the main program. The main disadvantage of this approach is the need to recompile the micro-interfaces in case of a change in one of them.

Runtime integration. This approach provides greater flexibility as micro-interfaces can be added, removed or updated without the need to completely rebuild the application [13].

Both methods of integration can be implemented in several ways. JavaScript, which assumes that each program is built as a separate bundle and is loaded and mounted on the page only when needed. In this case, common libraries and styles can be preloaded, which will reduce the final size of the application. Additionally, each bundle is deployed separately, allowing teams to update functionality independently of the others.

Webpack module federation is a feature of the webpack module bundler that allows developers to share code and modules across multiple applications or micro-frontends. Fig. 5 shows how with module federation remote components can be dynamically loaded from other applications at runtime, making it easy to compose and scale complex, distributed systems with ease.

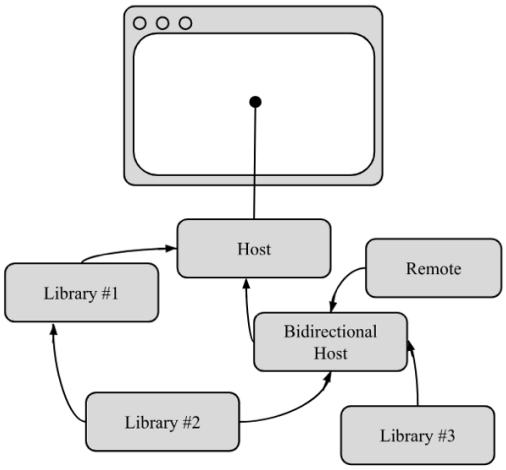

Fig. 5. Module Federation schema

Web components technology, which is a set of APIs that allow you to create reusable user interface components encapsulated in their own elements. It allows you to create your own HTML elements, define their behavior and load their code dynamically. In this case, the isolation is maintained by the browser. This approach allows you to choose an arbitrary technology stack for each component, then work on development and deployment independently within the same team or with different teams. Developers can use polyfills (Fig. 6), which are small utility libraries that implement new JS features, to make them work, but this will increase the size of the downloaded content and increase the load time;

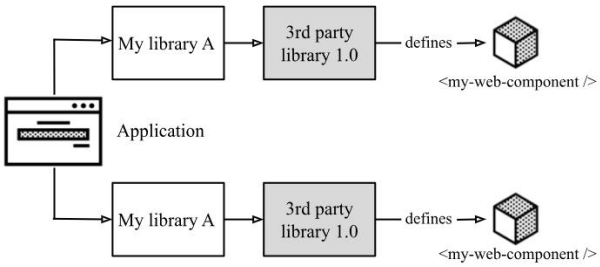

Fig. 6. Web components implementation schema

The single-spa framework which makes it possible to combine various applications regardless of the used library or framework, into one whole. It is possible to build a set of applications using the micro-frontend approach and single-spa both by creating the entire infrastructure and applications from scratch, and on the basis of an existing application.

Application Shell – a SPA container or a single HTML page or an entry point to micro-frontends. It's an HTML file with some JavaScript logic present throughout the user's session. It is responsible for loading JavaScript or HTML files that represent the entry point of micro-interfaces. Using HTML files as an entry point, it is possible to implement a server-side rendering engine inside the CI/CD pipeline, creating a page skeleton for loading, improving the user experience. This approach is closer to traditional UI development experience and fits well with the Strangler pattern.

F. EMPIRICAL PERFORMANCE COMPARISON OF DIFFERENT ARCHITECTURES

Let’s consider an example of a monolithic project that is refactored and rewritten to the architecture of the MFE by means of the browser. We measure the number of requests, the speed of loading and rendering of content. Table 1 and Fig. 7 show the measurement results [14]. It is obvious that the Module Federation showed the fastest load time, with approximately the same content rendering time and resource size compared to the monolith.

Table 1

Performance comparison

Type/Criteria	Monolith	Iframe	Module Federation
First paint	418ms	1222ms	540ms
Requests	13	34	26
Resources	5,4MB	16,6MB	6,6MB
Load Time	1,35s	1,12s	0,774s

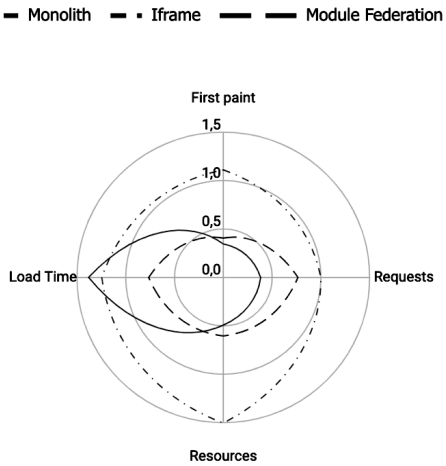

Fig. 7. Diagram with performance comparison of different architectures

VI. SUGGESTED OPTIMAL ARCHITECTURAL APPROACHES

Let us consider the comparative characteristics of various architectural approaches in the implementation of micro-interfaces (see Table 2).

Table 2

Comparison of different architectural approaches

Feature	Build-time Integration	Client-side Composition	Server-Side Rendering (SSR)
Integration Point	During the build process	In the user's browser at runtime	On the server, per request
SEO	Good	Requires extra effort, less ideal	Excellent
Initial Load	Often fastest	Potentially fast initially, depends on complexity	Fast, but usually slightly slower than build-time
Dynamic Updates	Requires re-build and deployment	Easy, components change independently	Possible, but adds complexity, might need full re-renders
Team Autonomy	High, some coordination needed	Highest	Moderate (potential coordination on server-side)
Runtime Complexity	Low	Moderate (orchestration & communication logic)	Low
Server Complexity	Low	Low	Medium to High
Technology Choice	Flexible	Most Flexible	Can be constrained by server-side framework

Finding the best architecture, of course, depends on a lot of conditions. In any case, optimal architecture is a search for a golden mean.

Analyzing the table (see Table 2), we can conclude that one of the optimal architectural solutions for preserving SEO, fast initialization loading, and fast interactivity provided by single-page applications is a hybrid architecture based on server-side rendering (SSR) (separate the components that are necessary for the initial loading and SEO optimization), and client side composition (CSC) (for fast interactivity) (Fig. 8).

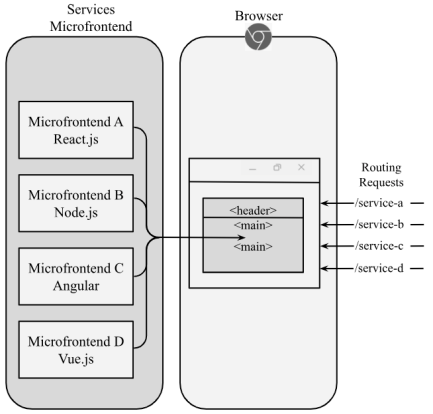

Fig. 8. Proposed hybrid architecture approach with SSR(Node.js) and CSC (React.js, Angular, Vue.js)

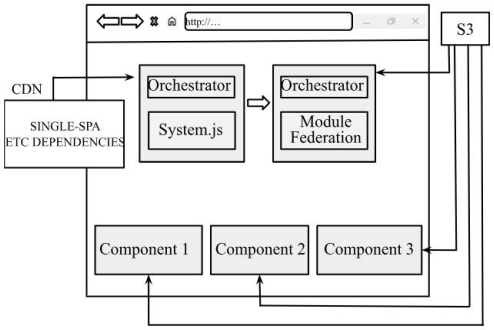

Fig. 9. Proposed hybrid architecture approach with build time integration (Module Federation) and CSC (Single-SPA)

The next possible optimal solution offered is a hybrid combination of client-side compositions with build-in integration, which also allows you to combine SEO optimization, fast initial load and fast interactivity of single-page applications. The idea is based on the single-spa architecture, next to the orchestrator, which is responsible for dynamically determining which programs are loaded based on their activation functions, replace the system.js (which is responsible for bundling and combining components) by a Module Federation with all its advantages as a build time integration (Fig. 9).

VII. CONCLUSION

In general, the emergence of an architectural style, which consists in the implementation of micro-interfaces in software applications, is a kind of natural reaction to the

increasing complexity of software. For long-term development and a large development team, this approach is a convenient solution with an eye toward accelerating the product creation process. In this case, all the complexity added by the tools that help support the development process and connect all the micro-interfaces will be compensated by the reduced coupling, i.e. the dependency of the modules on each other, and the management effort, since the development can be separated into separate teams.

However, the obvious disadvantage of this approach is the inconvenience of its use at all stages of software development. The argument for this statement is the fact that for small projects or projects with a small number of developers, such an architecture turns out to be a factor of additional load, since most of the efforts of the developers are spent on maintaining the architecture, not on the development of features, and the overall design time increases.

The proposed optimal hybrid combinations of architectures: SSR along with CSC and the Module Federation with CSC will increase the speed of systems that will be built on the basis of the mentioned solutions besides with good SEO optimization and fast interactivity as a single page application.

The provided research will be continued with further investigation of practical implementation of the proposed hybrid architecture solutions using different UI stack of technologies with performance testing, research on information transfer between micro interfaces will also be conducted.

References

[1] Blinowski, G., Ojdowska, A., & Przybylek, A. (2022). Monolithic vs. Microservice Architecture: A performance and scalability evaluation. *IEEE Access*, *10*, 20357–20374. DOI: 10.1109/access.2022.3152803.

[2] Cruz, P., Astudillo, H., Hilliard, R., & Collado, M. (2019). Assessing migration of a 20-year-old system to a microservice platform using Atam. *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. DOI:10.1109/icsa-c.2019.00039.

[3] Di Francesco, P., Lago, P., & Malavolta, I. (2018). Migrating towards microservice architectures: An industrial survey. *2018 IEEE International Conference on Software Architecture (ICSA)*. DOI: 10.1109/icsa.2018.00012.

[4] Terdal, Dr. S. (2022). Microservices enabled e-commerce web application. *International Journal for Research in Applied Science and Engineering Technology*, *10*(7), 3548–3555. DOI: 10.22214/ijraset.2022.45791.

[5] Zhou, J., Yang, L., & Wu, J. (2023). Micro-frontend architecture base. *Sixth International Conference on Computer Information Science and Application Technology (CISAT 2023)*. DOI: 10.1117/12.3003818.

[6] Pontarolli, R.P., Bigheti, J.A., de Sá, L.B.R., Godoy, E.P.L. (2023). Microservice-Oriented Architecture for Industry 4.0. Eng 2023, 4, 1179-1197. DOI: 10.3390/eng4020069.

[7] Perlin, R., Ebling, D., Maran, V., Descovi, G., & Machado, A. (2023). An approach to follow microservices principles in frontend. *2023 IEEE 17th International Conference on Application of Information and Communication Technologies (AICT)*. DOI: 10.1109/aict59525.2023.10313208.

[8] Auer, F., Lenarduzzi, V., Felderer, M., & Taibi, D. (2021). From Monolithic Systems to microservices: An assessment framework. *Information and Software Technology*, *137*, 106600. DOI: 10.1016/j.infsof.2021.106600.

[9] Homay, A., Zoitl, A., de Sousa, M., & Wollschlaeger, M. (2019). A survey: Microservices Architecture in Advanced Manufacturing Systems. *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. DOI: 10.1109/indin41052.2019.8972079.

[10] Marco, V., & Farias, K. (2024). Exploring the technologies and architectures used to develop micro-frontend applications: A systematic mapping and emerging perspectives. *SSRN Electronic Journal*. DOI:10.2139/ssrn.475066.

[11] Abdellatif, M., Shatnawi, A., Mili, H., Moha, N., Boussaidi, G. E., Hecht, G., Privat, J., & Guéhéneuc, Y.-G. (2021). A taxonomy of Service Identification Approaches for Legacy Software Systems Modernization. *Journal of Systems and Software*, *173*, 110868. DOI: 10.1016/j.jss.2020.110868.

[12] Chen, K. C. (2021, August 24). *Micro Frontend Framework Guide: Technical Integrations*. Trend Micro. https://www.trendmicro.com/en_us/devops/21/h/micro-frontend-guide-technical-integrations.html.

[13] Nikulina, O., & Khatsko, K. (2023). Method of converting the monolithic architecture of a front-end application to microfrontends. *Bulletin of National Technical University "KhPI". Series: System Analysis, Control and Information Technologies*, (2 (10)), 79–84. DOI:10.20998/2079-0023.2023.02.12.

[14] Petcu, A., Frunzete, M., & Stoichescu, D. A. (2023). Benefits, challenges, and performance analysis of a scalable web architecture based on micro-frontends. *University Politehnica of Bucharest, Scientific Bulletin., Series C*, *85*(3), 319-334.

Oleksandr Stepanov – graduated from Lviv Polytechnic National University in 2004. He received the B.S. and M.S. degrees in Electronics. He has been working in IT field as a front-end developer for over eight years. On most large projects, he faced with problem of aging technology, maintaining a large project, and migrating to new architecture approaches.

His research interests include client-server highly loaded information systems, performance scalability of micro-interfaces, migration from monolith architecture to micro-frontend, design and implementation of scalable systems.

Halyna Klym - doctor of technical sciences, professor, professor of the department of Specialized Computer Systems of the Institute of Computer Technologies, Automation and Metrology of Lviv Polytechnic National University.

In 2008, she received a degree of Doctor of Philosophy in the specialty: Physical and Mathematical Sciences at Ivan Franko Lviv National University.

In 2016, she received a Doctor of Science degree in Technical Sciences at Lviv Polytechnic National University. She conducts lecture courses on the design of ultra-large integrated circuits and methods and means of automated design of computer systems. She is an author of more than 170 scientific articles in international publications.