DOI: 10.20998/2079-0023.2023.02.12
UDC 004.9

O. M. NIKULINA, Doctor of Technical Sciences, Full Professor, Head of the Department of Information Systems and Technologies of the National Technical University "Kharkiv Polytechnic Institute",
Kharkiv, Ukraine, e-mail: elniknik02@gmail.com; ORCID: https://orcid.org/0000-0003-2938-4215
K. O. KHATSKO, Senior Lecturer of the Department of the Department of Information Systems and Technologies of the National Technical University "Kharkiv Polytechnic Institute", Graduate Student,
Kharkiv, Ukraine, e-mail: kyrylo.khatsko@khpi.edu.ua, ORCID: https://orcid.org/0000-0003-3315-1553

METHOD OF CONVERTING THE MONOLITHIC ARCHITECTURE OF A FRONT-END APPLICATION TO MICROFRONTENDS

Web systems have existed for a long time and quite a lot of them have been created. Modern development uses new microservice architectural to improve performance, portability, and other important characteristics. This necessitates the transformation of legacy systems from a monolithic architecture to a microservices one. Such a process is complex and costly, so improving the methods for converting old systems to a new platform is relevant. This research aims to develop a method of applying microfrontends approach for monolithic single page applications (SPA). The article proposes a method of transforming the software system architecture from monolithic to microservice architecture (MSA). Since the client part of the system is considered, the term microfrontend is proposed, as an analog of microservers in the server part of the software systems. A brief review of existing architecture reengineering research is made and the advantages of a microservice approach are identified. The proposed three-stage method differs from the methods by the selection of an additional stage of conversion, which allows to gently change the connections between parts of the monolithic application, which were implemented in the initial monolithic architecture. The first stage is reverse engineering, it is proposed to shift the focus from the search for outdated code to the functional analysis of the program as such. 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, we have several separate programs (microinterfaces) that are connected to the main program. An experiment with a typical external SPA demonstrates the operation of the proposed algorithm. The system obtained as a result of the transformation is compared with the original one according to the following measurable parameters: production builds building time, size of the main bundle, and first page average load time. All comparisons showed the advantages of the system obtained as a result of the conversion. As a result, the architecture transformation algorithm allows you to obtain a guaranteed better result, taking into account the limitations of the interface SPA, which were not considered by the authors of previous articles.

Keywords: information system, software architecture, algorithm, monolith model of an information system, software development process, software migration, microservice architecture, single page application, method of converting to microfrontends.

Introduction. To create scalable, future-oriented software systems in modern industrial programming, the microservice architectural approach is increasingly used [1, 2]. Microservices break traditional monolithic applications into a set of smaller services that can be independently developed, tested, and deployed [3]. Due to highly decoupled software modules, microservice applications are easy to debug, update, use third-party code, therefore, in a professional environment, they believe that the future is theirs [4].

However, many applications have already been developed as monolithic or modular, so in order to improve these applications, it is necessary to migrate them to a microservice architecture. Such actions have become the preferred solution for software upgrades [5] than new development.

With the development of browser-based client applications, as well as the requirements for them, the same problems that occur in monolithic backend applications become more and more relevant, this is especially acute in single page applications (SPA), which were originally conceived as a single monolith.

We use a microservice approach to break a monolithic SPA application into separate microfrontends. The topic of this paper is the method of converting the monolithic architecture of front-end applications to microfrontends.

Lets' talk about motivation. Just imagine that you're the developer on some great project with a beautiful microservice architecture. Each service is developed by separate team, services are tested and deployed in isolation. But let's back from imagine and see what we have on frontend. Here is the typical SPA application. It's designed with modern framework, but it is monolithic by its nature with all the disadvantages of this architecture. So we think may be we can do something like this – apply microfrontends.

In the second section we analyse existing articles and papers related to the strategies of the migration to the microservice architecture. We highlight that this migration could be successfully applied to solve an existing problems that could be occurred in applications with monolithic architecture approach. We also point that despite of the existence of the fact that all of the problems of the monolithic backend applications are inherent in front-end applications, approach with dividing has been less reflected in front-end development.

In next section we describe exiting methods of migration to the microservices. We propose to use a microservice approach to break a monolithic SPA application into separate microfrotends. In this section we also highlight limitations of the front-end SPA applications that could not allow to apply existing methods directly for converting its monolithic architecture into separate independent units similar to microservices. Here the additional motivation of such migration is described. Finally in the main part of the section, we propose new determination of the existing steps and describe all the changes to be done on every step.

The next sections describes the existing technical approaches to organize microfrontends. Here we mention the main advantages and disadvantages of these approaches and the ability to be used for SPA.

The results section contains a description of an experiment to prove the method proposed. Here we highlight the requirements to the application that should be the subject of the experiment of the architectural transformation to the microfrontends. We point the detailed descripttion of the changes in code and architecture according to the previously explained method steps. In this section the choice of a technical solution for the organization of microfrontends is justified. New microfrontends are hosted and the final application is evaluated.

In the last section we point the questions that are still not covered in the current experiment or could be improved in future works.

Analysis of Migration Strategies. Many business applications have been in use for many years, their development does not stop, and a lot of unsuccessfully fixed bugs have accumulated [6]. It would be useful for such applications to get a second life with a new architecture without this accumulated set of bugs. There is reason to believe that migrating to a microservice architecture will help overcome the existing problems. Particular reasons for migrating older applications are the fact that microservices improve maintainability over traditional monoliths due to a smaller code base, strong isolation of components, and organization of microservices around business functionality. In addition, the development company has the ability to create autonomous teams of employees, which should reduce coordination efforts and increase team productivity.

However, the introduction of microservices can complicate the quality assurance of systems [7]. From an architectural point of view, quality assurance is considered a key issue when migrating or developing systems based on microservices [8]. Most of the existing research on microservices is focused on architectural principles and the application of architectural patterns [9-11] in microservice migration practices, which can provide an analytical view of the common patterns and methods used for MSA, and can be considered the starting point of our work. Many researchers have contributed to the development and quality improvement of systems based on microservices [2, 3, 12].

As a result, MSA has also become the preferred path for software upgrades based on the architecture [13].

There are many examples of successful rewriting of applications based on microservices [14], when, next to the original, applications are made immediately in the execution of microservices [13, 15, 16].

While MSA has gained a lot of popularity as an architectural style for back-end development of web applications, this architecture has been less reflected in front-end development. Web applications have been around for a long time and many large systems have accumulated that have a monolithic architecture. This statement applies to both the server side and the browser side. For several years, research has been published on the transformation of the back-end from a monolithic or modular architecture to a microservice one [4, 17, 18].

Migration method. Since the microservice architecture primarily touched server applications, we will first consider the published methods for migrating to MSA.

The process of moving from an existing system to microservices, based on earlier work on systems reengineering, is described in three steps: reverse engineering, architecture transformation, and forward engineering [19].

The described migrations were motivated by the need to partially or completely modernize the system, to some extent such a system was considered legacy, so the system that existed before the migration was called pre-existing, and the target microservice system was called new system. At the reverse engineering step, the system was analyzed to identify obsolete code, which became a candidate for transferring it to services. Further, this transformation was a restructuring of the code with the transformation of the current architecture to a microservice one, but maintaining the same level of abstraction. At this step, the architecture, business model and business strategy are changed. At the stage of backward engineering, the system is being finalized, implemented and deployed.

However, the browser part of the system, the so-called front-end, has a number of limitations that make such a conversion a difficult task. Such limitations include the need to work with a single environment. It is executed on the client side, so within one application there is always only one address bar, one global BOM object, and, accordingly, the DOM that is part of it. It is around this problem that the main limitations of microfrontends are built.

The authors present the adapted process of transition of a monolithic SPA application to microfrontends as follows on fig. 1:

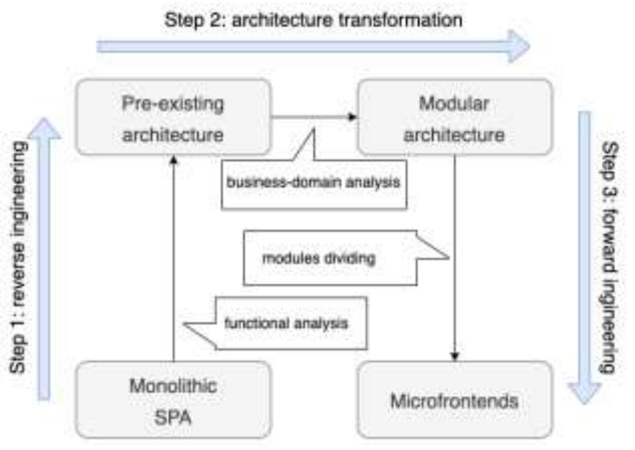
Step 2: architecture transformation
Pre-existing architecture
Modular architecture
business-domain analysis
modules dividing
functional analysis
Monolithic SPA
Microfrontends
Step 1: reverse ingineering
Step 3: forward ingineering

Fig. 1. Migration to microfrontends

The SPA approach has become popular relatively recently, so the motivations for migrating to microfrontends are caused not so much by outdated architecture, but by the non-functional benefits that microfrontends can provide. That is why the authors propose to revise the above transition steps and specify them as more appropriate in the context of working with client applications. So, for example, at the stage of reverse engineering, it is proposed to shift the focus from the search for legacy code to the functional analysis of the application as such. At this step, functions are grouped, unified, large functions can be separated. At the end of this step, our architecture (pre-existing) is still monolithic and requires further analysis before moving on to the next step.

The purpose of the stage of architecture transformation is to analyze the current application to determine the main business functions of the application and, based on them, to identify potentially separate parts of the application that should not depend on each other as much as possible. For these purposes, you can use the Strategic Design Domain-driven design approach. In the context of DDD, the main application domains are identified. To successfully solve this problem, all stakeholders can be involved: developers, architects, product owners; the project documentation is studied, compared with the main business requirements.

Based on the allocated domains, the application modules are created, which allows us to move to the modular architecture stage. It should be noted that at this stage we have already solved some problems inherent in monoliths: code is more structured and less coupled. And although we still have one application with bundles hosted on the one server, this architecture allows us to organize lazy loading of modules. This leads to a decrease in the size of the main bundle, and hence to a decrease in the initial load time of the application.

On the forward engineering step, the coupling between the components of the different modules are finally broken. At this stage, a technical solution for organizing microfrontends should be selected. Due to the limitations of client applications, in any division into separate parts we still need one main application to manage other microservices. All existing technical solutions come down to solve the problem of how individual microfrontends connect to the main application and how it orchestrates them.

Step by step architecture changes are described on fig.2.

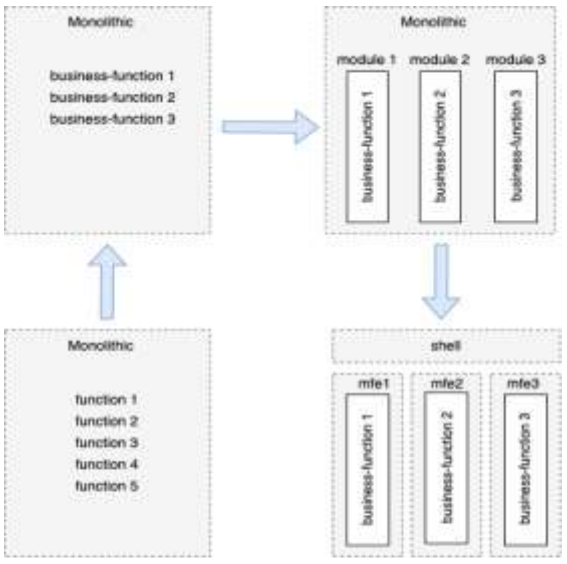

Fig. 2. Architecture transformation during migration to microfrontends

On the step of the backward engineering step there is no architecture changes. Here we can see changes on low level: grouping, unifying or removing unused functions. On this step we still have the same business models as in pre-existing system. On the next step the transformation of the architecture is finally take place. By the end of the step we have modular monolithic architecture. Business models could be changed to ensure better isolation of the modules. These modules are candidates to be moved into separate micro-frontends on the last step of the forward engineering step. By the end this step we have several separate applications (microfrontends) that are connected to the main application (shell).

Technical solutions for microfrontends. Currently there are several variants how to provide microfrontend architecture. The simplest way to organize microfrontens is to create several independent applications. We need to have one main application with hyperlinks to other microfrontend applications. Clicking on such hyperlink the user is navigated to the other application with other URL. The only benefit of this approach is its simplicity, but bad user experience is the biggest price for this advantage.

Another commonly used variant is applying single-spa framework. The idea is to create framework-specific wrapper for every microfrontend application to integrate them in one single-spa application. The main disadvantage of this approach is the necessary to follow strict single-spa framework rules for every microfrontend to organize integration with other microfrontends. If there is a ready-made application, then it is a bug risk that it should be rewritten taking into account the single-spa rules.

One of the most popular mechanisms is to apply i-frames. All necessary widgets should be placed in i-frames that load the corresponding microfrontend hosted on separate host. Data is exchanged between them using POST messages. The main disadvantage of the approach is the necessary of the loading full bundle of the microfrontend. This fact limits ability to use i-frames only for good isolated applications. Another downside is the risk of reloading libraries with the microfrontend bundles.

The most modern way to work with mircrofrontends is to apply the Module Federation feature of the Webpack module bundler. This approach allows both the good communication of the microfrontends and the ability to avoid code duplication. The main idea of the approach is configurate the shell application to import just the necessary module from mircofrontend application.

Results. The authors took for consideration the previously created Chess Tutorials application, on the client part of which experiments were carried out. When writing, the authors, recognizing the problem, chose an application that has a large number of internal communications, in order to maximally reflect the problems that developers face in the process of solving real problems. The client part is typical monolithic SPA created on the Angular framework with state management organized with NgRx. The application is an educational platform for learning the game of chess. The application is designed for two types of clients – teachers and students.

To prove the ability to apply the proposed method of converting the pre-existing monolithic SPA was refactored according to all the necessary steps of migration: reverse engineering, architecture transformation, and forward engineering.

At the stage of backward engineering, the interface elements were unified, large analyzed, reused components were identified. Functions related to authorization and student entities were separated; an application routing was changed by adding new route for home page, Angular SDK was upgraded to higher version etc.

At the next stage, an analysis of the business functions of the application was carried out. Business requirements defined the following abilities for different types of users: tutors could invite students to the system, create lessons and manage study groups, including tracking learning statistics for students; students could complete tasks from the lessons available to them, view the study groups. All available business documentation was studied including vision and existing prototypes. The example of the documentation that could be applied to identify business domains is application user path is displayed on fig.3.

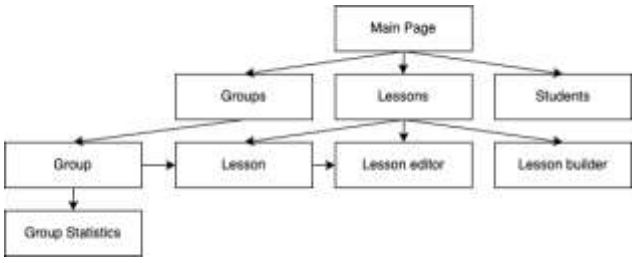

Fig. 3. Chess Tutorials documentation: path of the user with role tutor

Summarizing all the abilities and data from documentation, we identified such basic business functions as managing students, working with lessons, working with study groups. Based on the selected business functions, the following domains was determined: Lessons, Students, Groups and subdomain for Groups – Group Statistics. Schematic view of the modular application is displayed on fig. 4. Since domains must have a separate model valid only within their bounded context [21], at this stage, the application state (store) was restructured.

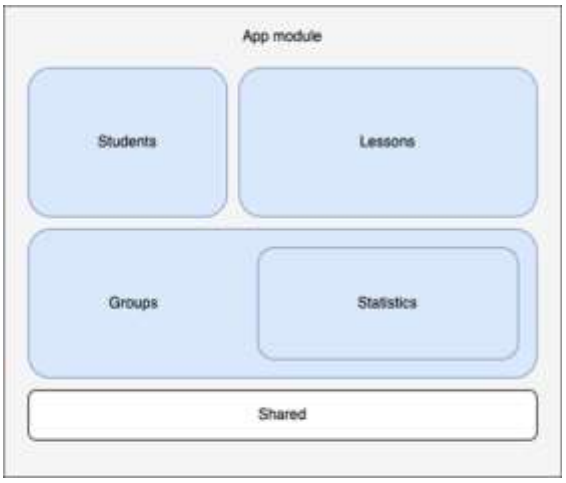

Fig. 4. Application structure by the end of the architecture transformation step

All support functions of the centralized storage have been broken into separate modules according to belonging to a certain domain. The models of User and Student have also been separated, since they belonged to different domains. In reality, it is not always possible to achieve complete isolation of domains. This is exactly the situation that the authors are faced with. To solve this problem, the data obtained during functional analysis were used – the identified reusable components were taken out into separate shared modules. It should be noted that creating a single shared module is a bad solution for large systems.

Since not all functions are reused in each of the above domains, it is recommended to organize several shared modules to prevent unnecessary functionality from being imported.

All the domain logic was moved to separate independent modules. All smart and dump components and support services responsible for working with students were collected in the Students module; the components of the lesson builder, view and list of lessons (support services and other structures) were moved to the Lessons module; everything related to group management, including a separate module of Statistics, was moved to the Groups module. After the necessary transformations, the Chess Tutorials application still looked like one big monolithic application, but consisting of as much as possible separated domain modules, as well as shared modules. At this stage a lazy loading of the modules was applied. Domain modules also had their own internal routing. The above domain modules were the candidates for separate microfrontends. Module Statistics at this stage did not look independent enough to be moved to a separate domain, and, accordingly, was not a candidate for moving to a separate microfrontend. With the development of the application and with the addition of a new functions, such a transfer may become relevant in the future, so such a decision may be postponed for this stage.

On the forward engineering step, all possible links between domains was broken down as much as possible, since each domain will be placed in a separate application. Orchestration by these applications was done by a shell application that was built from the main application module. Orchestration itself was done using routing. On the last stage we finally created microfrontends. One of the problems that we faced at this stage is the correct development of individual microfrontends. Since they should be separate independent applications, duplication of a large amount of code, at least the framework itself and styles, cannot be avoided. Our domain modules still had some common functions placed in shared modules. We also had state manager and common data used in several modules. All this led to the fact that duplicate parts of the code would be loaded several times, for example, when using the i-frame approach. To avoid code duplication in bundles of the future microfrondents we applied Webpack Module Federation approach. Microfrontends was still separate independent applications with duplicated code, but webpack allowed to load only the necessary (declared) modules of microfrontend applications in the resulting application on the client side. Shared module was divided into separate shared libraries. New microfrontend applications were created and main domains were moved from main application to them. After that we set up shell application to import just domain modules from our microfrontends and deployed all the applications separately. As a result we had shell application, mfe1 (Students), mfe2 (Lessons) and mfe3 (Groups).

Comparison of the pre-existing system and refactored system is displayed on table 1.

Table 1 – Pre-existing and target systems comparison

Measurable indicator	Pre-existing system (monolithic SPA)	Target system (Microfrontends)
Production builds building time, ms	20290	15363
Size of the main bundle, KB	540.5	82.7
First page average load time, ms	644	269

Because of the ability to run build process of every single microfrontend and shell application in parallel, the result build time could be equal to the build time of the largest application – mfe2. The size of the final bundle of the target system (microfrontends) has become smaller, which reduced the load time of the first page of the application.

Conclusions and Future Work. In current work, existing methods of migration to microservices are adopted to be acceptable for microfrontends. Steps of converting and new states of the application are defined and described to take into account the limitations of the front-end SPA which was not covered by authors of previous articles. Experiment with typical front-end SPA Chess Tutorials interface proved that the proposed conversion method shows a good result, in terms of the quality of the resulting software. Further research will be related to the concept of DDD for more efficient domain identification and microfrontend separation. Better understanding of conceptions of the strategic design and the boundary context will allow to design better domain models and as a result better isolated microfrontends.

References

1. Furrer F. J. *Future-Proof Software-Systems*. Springer, 2019. 376 p.
2. Gidey H. K., Marmsoler D., Eckhardt J. Grounded Architectures: Using Grounded Theory for the Design of Software Architectures. *IEEE International Conference on Soft-ware Architecture Workshops*. URL: https://doi.org/10.1109/ICSAW.2017.41 (access date: 23.10.2023).
3. Terdal S. Microservices Enabled E-Commerce Web Application. *International Journal for Research in Applied Science and Engineering Technology*. URL: https://doi.org/10.22214/ijraset.2022.45791 (access date: 23.10.2023).
4. Francesco P. D., Lago P., Malavolta I. Migrating towards microservice architectures: An industrial survey. *International Conference on Software Architecture,* URL: https://doi.org/10.1109/ICSA.2018.00012. (access date: 23.10.2023).
5. Cruz P., Astudillo H., Hilliard R., Collado M. Assessing Migration of a 20-Year-Old System to a Micro-Service Platform Using ATAM. *2019 IEEE International Conference on Software Architecture Companion* URL: https://doi.org/10.1109/ICSA-C.2019.00039. (access date: 23.10.2023).
6. Auer F., Lenarduzzi V., Felderer M., Taibi D. From monolithic systems to microservices: An assessment framework. *Information and Software Technology,* URL: https://doi.org/10.1016/j.infsof.2021.106600. (access date: 23.10.2023).
7. Li S., Zhang H., Jia Z., Zhing C. et al. Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology*. URL: https://doi.org/10.1016/j.infsof.2020.106449. (access date: 23.10.2023).
8. Soldani J., Tamburri D.A., Van Den Heuvel W.J. The pains and gains of microservices: a systematic grey literature review. *Journal of Systems and Software*. URL: https://doi.org/10.1016/j.jss.2018.09.082. (access date: 23.10.2023).
9. Kazman R., Woods S. G., Carrie`re S. J. Requirements for integrating software architecture and reengineering models: Corum II. *Proceedings Fifth Working Conference on Reverse Engineering*. URL: http://doi.org/10.1109/WCRE.1998.723185. (access date: 23.10.2023).
10. Razavian M., Lago P. Understanding SOA migration using a conceptual framework. *Journal of Systems Integration*. URL: https://core.ac.uk/download/pdf/15455794.pdf (access date: 23.10.2023).
11. Steyer M. *Enterprise Angular: Micro Frontends and Moduliths with Angular*. URL: https://www.angulararchitects.io/en/book/. (access date: 23.10.2023).
12. Homay A., Zoitl A., de Sousa M., Wollschlaeger M. A Survey: Microservices Architecture in Advanced Manufacturing Systems. *IEEE 17th International Conference on Industrial Informatics*. URL: http://doi.org/10.1109/INDIN41052.2019.8972079. (access date: 23.10.2023).
13. Abdellatif M., Shatnawi A., Mili H., Moha N. et al. A Taxonomy of Service Identification Approaches for Legacy Software Systems Modernization. *Journal of Systems and Software*. URL: https://doi.org/10.1016/j.jss.2020.110868. (access date: 23.10.2023).
14. Hasselbring W., Steinacker G. Microservice Architectures for Scalability, Agility and Re-liability in E-Commerce. *IEEE International Conference on Software Architecture Workshops*. URL: http://doi.org/10.1109/ICSAW.2017.11. (access date: 23.10.2023).
15. Patil M., Prajapat, S. Microservice Architecture for Scalability and Reliability in E-Commerce. *International Journal of Advanced Research in Science, Communication and Technology*. URL: http://doi.org/10.48175/IJARSCT-2050. (access date: 23.10.2023).
16. Asrowardi I., Putra S., Subyantoro E. Designing microservice architectures for scalability and reliability in e-commerce. *Journal of Physics: Conference Series*. URL: http://doi.org/10.1088/1742-6596/1450/1/012077. (access date: 23.10.2023).
17. Evans E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. URL: https://www.amazon.com/Domain-Driven-Design-Tackling-Complexity-Soft-ware/dp/0321125215. (access date: 23.10.2023).
18. Blinowski G., Ojdowska A., Przybylek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access*. URL: https://doi.org/10.1109/access.2022.3152803. (access date: 23.10.2023).
19. di Francesco P., Lago P., Malavolta I. Migrating Towards Microservice Architectures: An Industrial Survey. *IEEE International Conference on Software Architecture*. URL: https://doi.org/10.1109/ICSA.2018.00012. (access date: 23.10.2023).

References (transliterated)

1. Furrer, F. J. *Future-Proof Software-Systems*. Springer, 2019. 376 p.
2. Gidey H. K., Marmsoler D., Eckhardt J. Grounded Architectures: Using Grounded Theory for the Design of Software Architectures. *IEEE International Conference on Soft-ware Architecture Workshops*. Available at: https://doi.org/10.1109/ICSAW.2017.41 (accessed 23.10.2023).
3. Terdal S. Microservices Enabled E-Commerce Web Application. *International Journal for Research in Applied Science and Engineering Technology*. Available at: https://doi.org/10.22214/ijraset.2022.45791 (accessed 23.10.2023).
4. Francesco P.D., Lago P., Malavolta I. Migrating towards microservice architectures: An industrial survey. *International Conference on Software Architecture*. Available at: https://doi.org/10.1109/ICSA.2018.00012. (accessed 23.10.2023).

5. Cruz P., Astudillo H., Hilliard R., Collado M. Assessing Migration of a 20-Year-Old System to a Micro-Service Platform Using ATAM. *2019 IEEE International Conference on Software Architecture Companion*. Available at: https://doi.org/10.1109/ICSA-C.2019.00039. (accessed 23.10.2023).
6. Auer F., Lenarduzzi V., Felderer M., Taibi D. From monolithic systems to microservices: An assessment framework. *Information and Software Technology*. Available at: https://doi.org/10.1016/j.infsof.2021.106600. (accessed 23.10.2023).
7. Li S., Zhang H., Jia Z., Zhing C. et al. Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology*. Available at: https://doi.org/10.1016/j.infsof.2020.106449. (accessed 23.10.2023).
8. Soldani J., Tamburri D.A., Van Den Heuvel W.J. The pains and gains of microservices: a systematic grey literature review. *Journal of Systems and Software*. Available at: https://doi.org/10.1016/j.jss.2018.09.082. (accessed 23.10.2023).
9. Kazman R., Woods S. G., Carrie`re S. J. Requirements for integrating software architecture and reengineering models: Corum II. *Proceedings Fifth Working Conference on Reverse Engineering*. Available at: http://doi.org/10.1109/WCRE.1998.723185. (accessed 23.10.2023).
10. Razavian M., Lago P. Understanding SOA migration using a conceptual framework. *Journal of Systems Integration*, Available at: https://core.ac.uk/download/pdf/15455794.pdf (accessed 23.10.2023)
11. Steyer M. *Enterprise Angular: Micro Frontends and Moduliths with Angular*. Available at: https://www.angulararchitects.io/en/book/. (accessed 23.10.2023).
12. Homay A., Zoitl A., de Sousa M., Wollschlaeger M. A Survey: Microservices Architecture in Advanced Manufacturing Systems. *IEEE 17th International Conference on Industrial Informatics*. Available at: http://doi.org/10.1109/INDIN41052.2019.8972079. (accessed 23.10.2023).
13. Abdellatif M., Shatnawi A., Mili H., Moha N. et al. A Taxonomy of Service Identification Approaches for Legacy Software Systems Modernization. *Journal of Systems and Software*. Available at: https://doi.org/10.1016/j.jss. 2020.110868. (accessed 23.10.2023).
14. Hasselbring W., Steinacker G. Microservice Architectures for Scalability, Agility and Re-liability in E-Commerce. *IEEE International Conference on Software Architecture Workshops*. Available at: http://doi.org/10.1109/ICSAW.2017.11. (accessed 23.10.2023).
15. Patil M., Prajapat, S. Microservice Architecture for Scalability and Reliability in E-Commerce. *International Journal of Advanced Research in Science, Communication and Technology*. Available at: http://doi.org/10.48175/IJARSCT-2050. (accessed 23.10.2023).
16. Asrowardi I., Putra S., Subyantoro E. Designing microservice architectures for scalability and reliability in e-commerce. *Journal of Physics: Conference Series*. Available at: http://doi.org/10.1088/1742-6596/1450/1/012077. (accessed 23.10.2023).
17. Evans E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Available at: https://www.amazon.com/Domain-Driven-Design-Tackling-Complexity-Soft-ware/dp/0321125215. (accessed 23.10.2023).
18. Blinowski G., Ojdowska A., Przybylek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access*. Available at: https://doi.org/10.1109/access.2022.3152803. (accessed 23.10.2023).
19. di Francesco P., Lago P., Malavolta I. Migrating Towards Microservice Architectures: An Industrial Survey. *IEEE International Conference on Software Architecture*. Available at: https://doi.org/10.1109/ICSA.2018.00012. (accessed 23.10.2023).

Received 06.11.2023

УДК 004.9

О. М. НІКУЛІНА, доктор технічних наук, професор, завідувачка кафедри інформаційних систем та технологій Національного технічного університету «Харківський політехнічний інститут», Харків, Україна; e mail: elniknik02@gmail.com; ORCID: https://orcid.org/0000-0003-2938-4215

К. О. ХАЦЬКО, старший викладач кафедри інформаційних систем та технологій Національного технічного університету «Харківський політехнічний інститут», аспірант, Харків, Україна; e-mail: kyrylo.khatsko@khpi.edu.ua, ORCID: https://orcid.org/0000-0003-3315-1553

МЕТОД ПЕРЕТВОРЕННЯ МОНОЛІТНОЇ АРХІТЕКТУРИ FRONT-END ДОДАТКУ НА МІКРОФРОНТЕНДИ

Вебсистеми існують давно і їх створено досить багато. В сучасній розробці використовуються нова архітектура мікросервісів для підвищення продуктивності, переносимісті та інших важливих характеристик. Це зумовлює необхідність трансформації застарілих систем від монолітної архітектури до мікросервісної. Процес трансформації складний і дорогий, тому удосконалення методів перетворення старих систем на нову платформу є актуальним. Це дослідження спрямоване на розробку методу трансформації для монолітних односторінкових програм (SPA). У статті запропоновано метод трансформації архітектури програмної системи від монолітної до мікросервісної архітектури (MSA). Оскільки розглядається клієнтська частина системи, пропонується термін мікрофронтенд, як аналог мікросерверів у серверній частині програмних систем. Зроблено короткий огляд існуючих досліджень реінжинірингу архітектури та визначено переваги мікросервісного підходу. Запропонований метод з трьох етапів відрізняється від інших методів виділенням додаткового етапу перетворення, що дозволяє м’яко змінювати зв’язки між частинами монолітного додатку, які були реалізовані в початковій монолітній архітектурі. Перший етап – реверс-інжиніринг, пропонується перенести фокус з пошуку застарілого коду на функціональний аналіз програми як такої. На другому етапі пропонується перехід до модульної архітектури з виділенням функціоналу в окремі модулі. Наприкінці третього етапу ми маємо кілька окремих програм (мікроінтерфейсів), які підключаються до основної програми. Експеримент із типовим зовнішнім SPA демонструє роботу запропонованого алгоритму. Система, що отримана в результаті трансформації, порівнюється з вихідною за такими вимірюваними параметрами: час створення виробничих збірок, розмір основного пакету, що надсилається в браузер, та середній час завантаження першої сторінки. Усі порівняння показали переваги системи, отриманої в результаті перетворення. У результаті алгоритм трансформації архітектури дозволяє отримати кращий результат, враховуючи обмеження інтерфейсного SPA.

Ключові слова: інформаційна система, архітектура програмного забезпечення, алгоритм, монолітна модель інформаційної системи, процес розробки програмного забезпечення, міграція програмного забезпечення, мікросервісна архітектура, односторінковий додаток, метод перетворення на мікроінтерфейси.

Повні імена авторів / Author's full names

Автор 1 / Author 1: Нікуліна Олена Миколаївна, Nikulina Olena Mykolaivna
Автор 2 / Author 2: Хацько Кирило Олександрович, Khatsko Kyrylo Olexandrovych