Performance comparison of monolith and microservices based applications

Andrzej Barczak
Faculty of Exact and Natural Sciences, Siedlce University of Natural Sciences and Humanities
Siedlce, Mazowieckie, Poland

Michał Barczak
Mettler Toledo
Warsaw, Mazowieckie, Poland

ABSTRACT

In the following work analysis of the performance of microservices and monolith web applications are done. The article's primary goal is to analyze the behavior of two similar web applications created in microservices and monolith architecture. In the beginning, both monolith and microservices architectures are described. The following section concerns two web applications that were made to examine their performance. One of them is based on monolith architecture, whereas the second on microservices. Analysis are done using Application Insights Azure Module. To compare the performance of applications different metrics are taken into consideration. Very crucial metric is RAM Memory availability on the server. Another indicator taken into consideration is the percentage of CPU usage. One of the most important performance indicators, from the end-user perspective, is response time. All the tests of applications were done automatically using JMeter tool. Both simple requests and more complicated scenarios were tested.

Keywords: Performance analysis, Monolith applications, Microservices applications, Web applications

1. LITERATURE REVIEW

Several researchers were analyzing many aspects of microservices performance, including comparing efficiency of monolith and microservises applications. In this section we would like to present some of the articles concerning mentioned topic.

Freddy Tapia et al. in the article “From Monolithic Systems to Microservices: A Comparative Study of Performance” made very profound analysis of microservices and monolith application. Monolith application was hosted on KVM Virtual Machines, while microservices on EC2 cluster. Authors performed not only analysis of performance of monolith and micorservices Node.JS applications, but as well proposed mathematic models describing the performance factors [7]. Authors compared performance indicators like CPU usage, memory usage, disc reading speed, and disc writing speed. The stress tests were performed. Taking into consideration mentioned factors. Despite higher usage of CPU, authors consider the micorservices architecture more efficient [7].

Similar studies were performed by Alexis Saransig an Freddy Tapia in “Performance analysis of Monolithic and Microservice architectures – Containers technology” . Authors as well compared monolith application hosted on KVM with E2C hosted microervices application. Performance indicators were similar to ones used in previous article. However the results were deferent comparing to ones presented in “From Monolithic Systems to Microservices: A Comparative Study of Performance”. Authors indicates that both CPU and memory usage are higher while using microservices [2].

Wojciech Zabierowski in his “ The Comparison of Microservice and Monolithic Architecture” compared response times for monolith application and microservices application. Author proposed microservices architecture with load balancer, allowing to split the requests between several instances of the same microservices. In case of big amount of request microservices architecture with load balancer proved to be most efficient [16].

Omar Al-Debagy and Peter Martinek in “A Comparative Review of Microservices and Monolith Architectures” made the performance evaluation of microservice an monolith application under high and low load [10]. Authors took into consideration metrics like response time or number of processed requests per second. As well comparison of efficiency of microservices using deferent service discovery technologies (Eureka and Consul) was performed. The presented results shows that under regular load performance of microservices and monolith applications are similar. However, when the number of requests grows monolith application turn out to be more efficient. Performance of microservice that use Consul service discovery tool occurred to be higher [10].

Marco Amaral et al. published “Performance Evaluation of Microservices Architectures using Containers”. Authors analyzed performance of microservices hosted in the containers. They examined both master slave container approach and nested containers approach to hosting microservices [8]. In the master slave architecture there exists one container, called master, that coordinate and manage other slave containers. In nested containers approach we can determine nested containers being a child containers of main container. Authors compared CPU usage, network traffic and overhead of creating containers. Researchers get into conclusion that performance of nested containers is higher.

In a comparison with described studies, we analyzed performance of two web applications having exactly the same business logic. Both of applications were developed by authors. One of the applications is created using monolith architecture, while second is consuming microservices. Applications are hosted as Web Apps in Microsoft Azure cloud, not like in other works, on KVM or E2C.

2. MONOLITH APPLICATIONS AND MICROSERVICES APPLICATIONS

2.1 Monolith applications

Monolith architecture is one of the oldest software architecture. The concept of this architecture is to create an application containing all necessary components. All components are

dependent on each other and very often cannot run or even compile separately. Internally, monolith applications contain many different libraries. However, externally, it is usually represented as a single container [4]. We can say that monolith applications are applications, which modules and functionalities cannot be installed separately [12].

There are three major types of monolith applications. Most simple are one process monolith applications. Such applications can consist of many components that run as one process. This type of monolith system is the most common one. There exists as well distributed monolith systems. Distributed monolith systems may contain several distributed components that cannot work separately and have to be installed together. Without doubt, third party applications, so called black box applications, can be considered, by users, as monolith applications. The main feature of black box applications is the lack of possibility to modify their code by a customer.

Monolith architecture, for sure, has got a couple of advantages. The main of them is simplified process of installation. Monolith applications in most cases (excluding distributed ones) are installed as one service [12]. As monolith applications are very often a single process, monitoring them is relatively simple. Unquestionable, it is easy to perform end-to-end tests of monolith applications. Developers working on monolith applications do not need to analyze aspects of securing communication between several services, what is as well a benefit [5]. For sure lower costs of hosting monolith applications can be a pro. The performance of monolith applications usually is higher than the microservices ones. The main reason for that is the lack of need for communication with another process or service. In this article, we would like to proof that single process monolith applications, are using fewer resources like CPU or memory. Monolith architecture is perfect for small applications, having minimal number of functionalities.

Thinking of the disadvantages of monolith applications, we should consider a lack of possibility to scale it easily. Horizontal scaling is limited to duplication of the whole application. Of course, vertical scaling is doable by adding more resources. However, possibilities of scaling microservices applications are way bigger. Major withdraw of monolith architecture is a more challenging process of code updates in comparison to microservices applications. The codebase of complex monolith applications usually is large. Adding new functionality requires a deep analysis of dependencies between all application components. Due to connections between components, it is hard for programmers to follow the Open-Closed Principle while creating new functionalities. The Open-Closed Principle is one of the basic rules, developers should follow while creating software. The principle says that the application should be open for extension but closed for modification. It means that software allows extending its' functionality without modification of existing code [14]. One of the biggest drawbacks of monolith applications is their vulnerability. It is common that one issue is heavily affecting the full application, not just a small part of it. Monolith applications are technology dependent as well. A full application needs to be created using the same technology stack. It can lead to problems with updating a software's code to a newer version of the framework. Another problem with monolith applications may occur while trying to integrate it with other software. Table 1 presents the advantages and disadvantages of monolith architecture.

Table 1. Advantages and disadvantages of monolith applications. Source: own work

Advantages of monolith applications	**Disadvantages of monolith applications**
Simplified installation	They are hard to scale
Easy monitoring	The challenging process of code updates within the complex application
Simple for testing (mostly end-to-end tests)	Very often big codebase
No need to secure communication between several processes or services	Harder to follow the Open-Closed Principle
Lower costs of hosting	They are very vulnerable to issues and bugs
Usually higher performance than microservices applications	Technology dependency
Perfect for small projects with limited functionality	Problematic update to a newer version of the programming framework
	Problems with integration with other software

2.2 Microservices application

Microservices architecture is based on small distributed applications communicating with each other using Web calls [13]. One of the main features of microservices architecture is the autonomy of services. Microservices are independent of each other and should not share the same data source. More than that, microservices can be deployed separately. Individual service is responsible for limited logic that is connected around the same business domain [9]. The question is how small the microservice should be. In our opinion, the perfect answer to it was given by Jon Eaves from RealEstatet.co.aum, who defined microservice as an application that can be rewritten in two weeks [12].

Using small independent applications providing particular functionality has got its' positives and negatives. The main advantage of microservices architecture is scaling. Services can be scaled both horizontally and vertically in an independent way. Developers can scale a particular service having efficiency problems without the need to change other services. Isolation of microservices makes the system more bug-proof. An issue in one of the microservices usually does not affect the full system, but just a smaller part. The microservice architecture allows splitting the development of services between different programmers' teams. As communication between services is done via network, mostly by REST-full API-s returning JSON files, developers can use deferent technology stack for deferent microservices. Network communication standards allow applications created in two totally different programming languages, like Node.JS and .NET, to cooperate. Using Microservices allows hiding the technical details of implementation. REST-full API services are providing only functionality. A client using API is not aware of such technical details as used technology, coding, or database schema. Microservices architecture supports three of five main good practices of object-oriented programming (SOLID Principles). By concentrating on the limited business domain, microservice applications usually follow the Single Responsibility Principle. The rule says that a single class should have only one business responsibility, which is easy to achieve in a microservices architecture. As services are independent, their functionality can be extended easily without doing significant modifications to the full system. This features for sure allow fulfilling the Open-Close Principle in an easier way than within monolith systems. The last principle that can be easily implemented with microservices is the Dependency Inversion Principle.

Dependency Inversion contains two rules. "A. High-level modules should not depend on low-level modules. Both should depend on abstractions. B. Abstractions should not depend upon details. Details should depend upon abstractions." [15]. A high-level module like user interface, in a microservice architecture, is not depending on service implementation, but on abstraction being API contract. Another advantage of microservices is the possibility to deploy the services separately. In comparison to monolith systems code base of microservices is relatively smaller.

Unfortunately, microservices architecture is not flawless. Installing a full system might be more problematic than in the case of monolith applications. It needs deploying several services, having independent data sources, not one application. Of course, if developers are using Continues Integration/Continues Deployment processes (CI/CD), deployment is automated. However, we should remember that CI/CD deployment pipelines very often require complex and complicated configuration. Network communication between services needs to be secured, what as well costs developers additional effort. Monitoring several distributed microservices might be more problematic and challenging than monitoring the single process of monolith application. Very often performance of microservices based applications is worse than the performance of monolith systems. Commonly microservices are not sharing the same resources, as they are installed on individual servers, what leads to higher usage of CPU and RAM memory than in monolith applications. Of course, microservices can be installed on one machine and share resources. However, in such a case, developers need to ensure the separation of microservices. Not well isolated microservices can affect on performance or availability of each other. To ensure isolation containerization of microservices is commonly used [11]. Table 2 presents the advantages and disadvantages of Microservices architecture.

Table 2. Advantages and disadvantages of microservices architecture. Source: own work

Advantages of microservices architecture	**Disadvantages of microservices architecture**
Ease to scale	Hard for monitoring
An issue in one microservice usually does not have a big effect on other	The deployment process might be challenging
Allows splitting development between several teams	The need for securing the connection between services
Is not technology dependent	Lower performance
Promoting good practices of development	
Smaller code base	

3. IMPLEMENTATION OF MONOLITH AND MICROSERVICES APPLICATION

3.1 Applications

For the purpose of this article, two web applications and five microservices were created. All of them were developed using ASP .NET Core 3.1 and are deployed on Azure as WEB Apps. One application is based on monolith architecture, and the second one is consuming developed microservices. Both applications are the implementation of an online shop and share similar business logic. Each of the applications has got references to WebShopModels and WebShopInterfaces libraries. WebShopModels library defines a data structure that is used within applications. WebShopInterfaces contains interfaces, defining access methods that need to be provided by database managers, and generic abstract classes to access the database. For every data model class, a separate interface was created. BaseShopProvider is as well part of the WebShopInterfaces component. This is an abstract class consisting of instances of data access interfaces. Monolith applications and web app that uses microservices have got separate classes inheriting from BaseShopProvider. The mentioned class allows providing the proper implementation of interfaces for both applications. Implementation of BaseShopProvider is injected in the startup of applications using IServiceProvider, that is default .NET Dependency Injection Container [2].

Data sources for microservices and monolith applications are Microsoft SQL databases hosted in Azure. To communicate with databases, the EntityFramework library is used. Details regarding the implementation and architecture of the mentioned web application are presented in the following subsections.

3.2 Monolith Application

WebShop monolith application is a .NET Core MVC application. Except for standard MVC modules like Model View and Controler, it contains modules like Managers, Helpers, and DB. Managers are responsible for database operations. For each data model from WebShopModels separate manager is defined. All of the managers provide CRUD (create, read, update, delete) methods for proper data model class and implement correspondent interface defined in WebShopInterfaces library. DB module contains one data context class responsible for communication with the database. In the Helpers component, we can find MonoShopProvider class. This class is the implementation of BaseShopProvider, allowing access to the managers. It is injected in a startup and is used by the controllers to perform actions on the database. Figure 1 presents the package diagram of the monolith application.

3.3 Microservices Application

To compare the monolith application's performance and microservice one, we developed five microservices WebShopClientService, WebShopItemsService, WebShopOrdersService, WebShopProducersService, and WebShopStockService. As well client web application WebShopMicroservice consuming mentioned services was created. Each of the microservices has got a separate database, so they can be deployed individually. Needed communication between microservices and the client application is done via REST API calls. More other WebShopItemsService is calling WebShopProducersService to get the list of producers. Every microservice has got references to WebShopInterfaces and WebShopModels.

The architecture of microservices is very simple. Each of them contains modules like Managers, DB, and Controllers. Using the DB module, we are able to connect to the database.

Managers module is responsible for performing database actions, while in the Controllers module, there are implementations of REST methods.

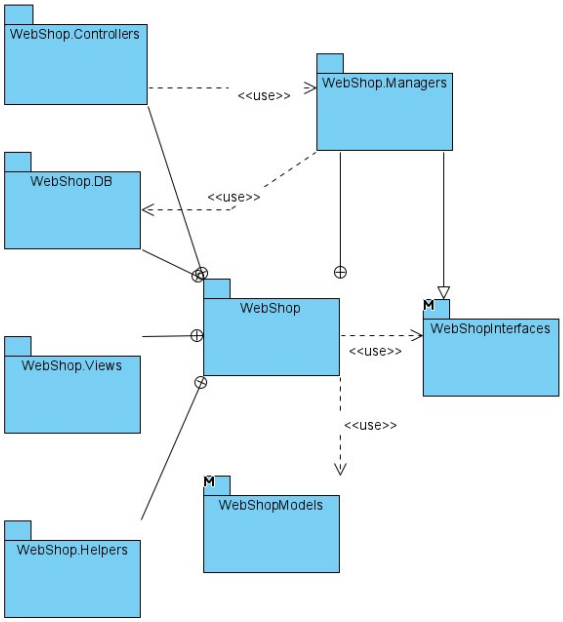

Fig. 1. Package diagram of monolith application. Source: own work

WebShopClientService is responsible for CRUD functionality related to Clients and Addresses data entities. WebShopItemService provides methods enabling manipulation of Item and ItemType entities. Using WebShopOrdersService, it is possible to perform actions on Orders and OrderItems data models. WebShopProducersService enables manipulation of Producers entities, while WebShopStockService provides methods related to Stock entity. Each microservice is using its own database managers independent form managers implemented in monolith application.
In the client web MVC application WebShopMicroservice, as in the monolith application, managers and helpers modules were added. Managers contain classes used for communication with microservices and implementing corresponding interfaces from WebShopInterfaces. In the Helpers modules, we can find MicroservicesShopProvider class, which is the implementation of the BaseShopProvider abstract class, providing access to microservices managers. MicroservicesShopProvider is injected on a startup of an application. Figure 2 presents the package diagram of the microservices application.

4. PERFORMANCE TESTS OF MONLITH AND MICROSERVICES APPLICATIONS

4.1. Test scenarios

Test scenarios for both monolith and microservices applications are the same. Authors defined two scenarios. The first one is relatively simple and consists of three steps. In the beginning, all items available in the shop are presented. After it, we filter out items having the letter "L" in the name.

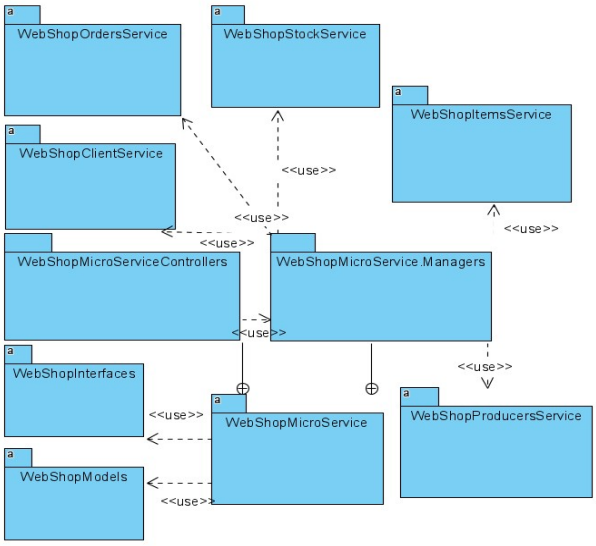

Fig. 2. Package diagram of microservices application. Source: own work

Finally, the list of items is ordered by the producers' name. The second scenario is more complicated. In the beginning, user is entering the page having lists of items available in the web shop. Next, one of the items is selected. The application verifies if the selected item exists in stock. If yes, the "success" popup message appears, and the user is able to fulfill the order. Before proceeding with the creation of a new order, the user needs to log in. After providing proper credentials, he is redirected to the order page. The application automatically fills in the users' addresses. After confirming the order by the user, it is saved in a database. Finally webpage with details of all users' orders is shown.
Monolith application contains all needed functionality to fulfill the test scenario. The microservice client application is calling several microservices during the test. Information about items in the store is gathered from WebShopItemsService. Details regarding producers are taken from WebShopProducersService microservice. Process of checking if the selected item exists in the stock takes place in WebShopStockService. Address data can be accessed using the WebShopClientService. Final saving of an order and gathering order details is done via WebShopOrdersService.

4.2 Testing tool

To perform tests of microservices and monolith applications, the Apache JMeter tool is used. JMeter is an open-source testing tool that performs complicated performance tests and simulates heavy network traffic [1].
With the BlazeMeter Chrome browser plugin, we could record webpages usage, and base on that prepare test scripts. Performance of applications is compared base on results presented in Azure Monitoring/Application Insights. To emulate network traffic, we set the number of users performing the JMeter test to 50 for the first test scenario and 20 for the second. Test duration was around 2 minutes, and the requests are evenly spread in time.

4.3 Test results

Test results of the microservices and monolith application will be presented in the form of a tables. Test duration was between 2 and 3 minutes. Each row of a table presents values per minute of a test. To evaluate and compare both applications' performance, we took into consideration metrics like response time, available RAM memory, and percentage of CPU usage. In the case of the first test scenario, there is a need to analyze results for the following three services, WebShop service, being a monolith application, WebShopMicroservice, which is the client for microservices applications, WebShopItemsService and WebShopProducersService microservices. Table 3 presents information regarding the average server response time in milliseconds during tests.

Table 3. Server response time for first scenario. Source: own work

Minute of a test	WebShop	WebShop Microservice	WebShop ItemsService	WebShopProducersService
1	32.83	95.2	78.53	2.45
2	32.28	58.46	53.16	2.44
3	No requests	62.46	No requests	2.45

We can notice that the microservices-based application's response time is between 2 and 3 times higher than the monolith application response time. The reason for that is the need to call microservice for gathering data.
The second important performance indicator is the usage of memory. Table 4 shows memory usage in MB for services while a simple test scenario is performed.

Table 4. Memory usage for first scenario. Source: own work

Minute of a test	WebShop	WebShop Microservice	WebShop ItemsService	WebShopProducersService
1	138.80	176.98	163.23	102.93
2	138.84	174	163.24	103.24
3	End of a test	160	End of a test	103.20

Usage of RAM memory of each of the services is similar. Values are in a range between 138 MB and 177 MB. Of course, cumulative usage of memory of client application and microservice is higher than the usage RAM of monolith application. However, considering the amount of RAM available on production servers, not mentioning cloud services like Azure, that should not be a big problem. We would like to emphasize that the tests were done on relatively simple applications. Nevertheless, we assume that usage of memory will not be dramatically higher on production microservices.
The third performance measure we analyzed is the percentage of the used CPU. In table 5, you can find exact values for the mentioned gauge.

Table 5. Percentage of CPU usage for first scenario. Source: own work

Minute of a test	WebShop	WebShop Microservice	WebShop ItemsService	WebShopProducersService
1	5.46	1.74	0.61	0.19
2	2.63	2.22	1.25	0.2

The CPU usage in the first minute of a test was twice higher for the monolith application than the summarized usage of the processor for WebShopMicroservice and microservices. We can notice a considerable decrease in monolith application values during the second minute, while the numbers for the system based on microservices architecture increased.
Same metrics were analyzed during performing, more complex, second test scenario. To fulfill the requests, WebShopMicroservice is calling four microservices WebShopItemService, WebShopProducerService, WebShopStockService, and WebShopOrdersService. Their performance is taken into consideration during analyses. Table 6 shows the average application response time in milliseconds.

Table 6. Server response time for second scenario. Source: own work

Minute of a test	Web Shop	WebShopMicroservice	ItemsService	ProducersService	StockService	ShopOrdersService
1	49.39	51.18	64.49	1.94	7.54	No requests
2	19.04	43.53	61.17	2.25	6.37	18.33
3	40.48	35.54	64.54	2.11	3.77	19.43

Like in the first scenario, the average response time of microservices application is higher than the monolith one. In the first minute of a test, the difference was minimal (around 2 ms). However, during the second minute, it increased significantly. Surprisingly, during the last minute of the test, the monolith application's average response time was slightly higher than the response time from the microservice application. What is worth mentioning the average response time for WebShopItemsServic is higher than the response time of WebShopMicroservice. The reason for that is the big difference between the minimum and maximum response time of services. The maximum response time for WebShopMicroservice is over 460 ms, while the minimum is 0.5 ms. For WebShopItemsService, deference is lower. The maximum response time is 215 ms, while the minimum is 1.53 ms. The fact that, WebShopMicroservice is calling as well relatively fast WebShopStockService and WebShopOrdersService, is certainly not without significance.
The next analyzed performance gauge is the usage of RAM memory during the second test. Memory was measured in MB. Table 7 shows memory usage by services.

Table 7. Memory usage for second scenario. Source: own work

Minute of a test	Web Shop	WebShopMicroservice	ItemsService	ProducersService	StockService	OrdersService
1	138.97	65.78	150.11	65.16	60.45	100.26
2	138.70	65.79	150.09	65.27	60.47	100.17
3	138.17	65.79	150.05	65.30	60.47	100.16

Similar to the first scenario, we can notice that cumulated usage of a memory of microservices is much higher than RAM needed by monolith application. Corresponding to the first scenario, during the second test, usage of memory by WebShopMicroservice is lower, as microservices perform more actions.
The last considered metric is the percentage of CPU usage. Test results for this indicator are presented in Table 8.

Table 8. CPU usage for second scenario. Source: own work

Minute of a test	Web Shop	Web Shop Microservice	Items Service	Producers Service	StockService	OrdersService
1	0.14	0.05	0.14	0.14	0.04	0.41
2	1.44	0.5	0.16	0.13	0.1	0.55
3	1.66	1.5	0.4	0.12	5.1	0.66

The same as in the first scenario percentage of CPU use is very low. It increased diametrically
at the end of the test due to comparatively higher CPU usage by WebShopStockService.

5. SUMMARY

Recently microservices architecture is becoming more popular. Market leaders like Netflix, Uber, or Amazon decided to migrate some of their monolith applications to microservices [6]. The reasons for such a situation are the undeniable advantages of microservice architecture. Certainly, the fact that microservices are more issue-proof than monolith application is one of this architecture's significant favors. Without a doubt, microservices can be scaled quite easily. It is crucial that microservices systems are technology independent and can be easily developed by distributed teams. Often, as a major drawback of microservices performance is considered.

In this article, we analyzed the performance of two web applications. One of the developed applications was the monolith, while the second was a web client consuming five microservices. Both applications were published on Azure and have similar business logic. For analyzing applications, we have chosen metrics like response time, usage of RAM memory, and CPU usage. Two test scenarios were defined. Using the first one, we were able to check performance during the simple call to the database. The second scenario allows testing more sophisticated logic of the applications. The results of tests show that the monolith application's performance is better than the performance of microservices. During the first test, the difference between microservices' response times and monolith application was very high. It was caused by number of calls that need to be sent from client application to one microservices. The solution for that would be duplicating microservice and using the load balancer. While a more complicated scenario took place, the results were not so unambiguous. For three minutes of a test, only during one minute, the average response time of monolith application was significantly lower than microservices' response time. Cumulated usage of RAM memory for the client application and microservices is higher than RAM usage of monolith application. Numbers are still relatively low, and considering that most of the production servers are powerful, it should not be a problem. Similar results we could observe for the usage of CPU. The usage of computing power might be a problem when companies want to host applications on a cloud. Very often, the cloud provider is charging money for the usage of CPU and RAM. In such a case, microservices applications for sure will be more expensive than monolith ones.

Nevertheless, in our opinion, microservices architecture will be even more popular in the near future, becoming soon standard.

6. REFERENCES

[1] **Apache JMeter**. https://jmeter.apache.org/. Accessed 21.03.2021

[2] A.Saransig, F.Tapia. **Performance analysis of Monolithic and Microservice architectures – Containers technology**. Proceedings of the 7th International Conference on Software Process Improvement

[3] **Dependency injection in ASP.NET Core.** https://docs.microsoft.com/pl-pl/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-3.1. Accessed 21.03.2021

[4] **Monolithic applications**,. https://docs.microsoft.com/pl-pl/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/monolithic-applications. Accesses 21.03.2021

[5] **Monoliths vs. microservices — benefits and drawbacks.** https://medium.com/transparent-data-eng/monoliths-vs-microservices-benefits-and-drawbacks-a-comparision-9e7a462b8e3a. Accessed 21.03.2021

[6] **Why and How Netflix, Amazon, and Uber Migrated to Microservices: Learn from Their Experience**. https://www.hys-enterprise.com/blog/why-and-how-netflix-amazon-and-uber-migrated-to-microservices-learn-from-their-experience/. Accessed 21.03.2021

[7] F.Tapia et al. **From Monolithic Systems to Microservices: A Comparative Study of Performance**. Applied Science 2020, 10(17)

[8] M.Amaral et al. **Performance Evaluation of Microservices Architectures using Containers**. IEEE 14th International Symposium on Network Computing and Applications. 2015

[9] N.Ford, R.Parson, P.Kua**. Building Evolutionary Architectures: Support Constant Change**. pp. 99-109. O'reilly 2017.

[10] O. Al-Debagy, P.Martinek. **A Review of Comparative Microservices and Monolith Architectures.** 18th IEEE International Symposium on Computational Intelligence and Informatics. 2018

[11] S.J.Fowler. **Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization**, pp. 89-109. O'reilly 2016.

[12] S.Newman. **Building Microservices**, pp. 13-29. O'reilly 2015

[13] S. Newman. **Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith**, pp. 13-26. O'reilly 2019

[14] R.C.Martin. **Agile Software Development, Principles, Patterns and Practices**, pp. 113-116. Prentice Hall 2002.

[15] R.C.Martin. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**, pp. 107-111. Pearson Education 2017.

[16] W.Zabierowski. **The Comparison of Microservice and Monolithic Architecture.** IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH).2020