Design of Modern Distributed Systems based on Microservices Architecture

Isak Shabani[1], Endrit Mëziu[2], Blend Berisha[3], Tonit Biba[4]
Department of Computer Engineering
Faculty of Electrical and Computer Engineering
University of Prishtina
Prishtina, Kosovo

***Abstract*—Distributed systems are very commonplace nowadays. They have seen an enormous growth in use during the past few years. The idea to design systems that are robust, scalable, reliable, secure and fault tolerance are some of the many reasons of this development and growth. Distributed systems provide a shift from traditional ways of building systems where the whole system is concentrated in a single and indivisible unit. The latest architectural changes are progressing toward what is known as microservices. The monolithic systems, which can be considered as ancestors of microservices, cannot fulfill the requirements of today's big and complex applications. In this paper we decompose a monolithic application into microservices using three different architectural patterns and draw comparisons between the two architectural styles using detailed metrics that are generated from the Apache JMeter tool. The application is created via .NET framework, uses the MVC pattern and is fictive. The two comparable apps before testing with Apache JMeter, will be deployed in almost identical hosting environment in order to gain results that are valuable. Using the generated data, we deduce the advantages and disadvantages of the two architectural styles.**

***Keywords*—Distributed systems; microservice; monolithic; web services; JMeter**

I. Introduction

Microservices are a new development, coming into light just a few years ago. They offer many advantages compared to the old monolithic architectures. That is why many of the big tech companies have successfully made the switch to microservices. Currently, the monolithic architecture is the default model for creating a software application. Its trend is decreasing as it cannot keep up with the demands and the challenges of the new applications that are now quite big and complex.

In the monolithic architecture, application is built as a single indivisible unit. This usually means that the application has three core components that interchange information with each other: a user interface, a server-side and a database [1]. This architecture is characterized by a huge code base and has almost no modularity. Because they have a single code base, they can become so large and hence difficult to maintain. The whole application will need to be redeployed from a single small change in the code. More crucial is the fact that it is not very reliable since a bug in any part of the code can bring down the whole application [1].

Monolithic architecture, however, has some subtle advantages and with some tweaks it can still be useful to many modern applications. These include: the easiness of deployment (since only one file needs to be deployed), the easiness of development (compared to the microservices) and the network latency and security which are more noticeable in the microservices architecture. Monolithic architecture is also very easy to test. We can do so by simply launching the app and testing the UI with Selenium. However, some of the drawbacks of this architecture have made the switch to microservices a necessity [2].

Because today's apps are big and complex, in order to be useful, they need to be robust and reliable. The resources must be utilized efficiently so the users can get a seamless experience while surfing the app. Many components of the app might have different resource requirements. Some might need more CPU cycles, some others more memory etc. This imposes the need to scale the different components, independently. Scaling in the monolithic architecture is done by creating copies of the app. This means that all of these copies will access all of the data which in turn makes caching less effective and increases memory consumption and I/O traffic [2].

As authors in [3] put it, one of the problems that can arise from the monolithic applications is the evolvement into a "big ball of mud" state, which is a situation in which none of the developers understand the entire application. To overcome the obstacles, microservices provide a very reasonable and effective architectural style, which as mentioned, are increasingly being used and deployed in many modern applications. In fact, microservices are considered as the future of distributed systems.

On the other hand, despite its name, microservices are by no means, small. In this architectural style, the application is made up of a suite of small devices, all of which have their own unique codebases.

Microservices use lightweight mechanisms, somewhat like an API, to communicate between different services. Contrary to monolithic architecture, these services can be deployed together or separately. These services are loosely coupled (or headless) making this architectural style mostly decentralized [3].

It must be understood that a microservice is not a layer within a monolithic application. It has its self-contained functionalities with clear interfaces, and through its own internal components, must implement a layered architecture. According to the author in [4] this architecture follows the Unix philosophy of "do one thing and do it well". In the following sections we will explore some of the main advantages of microservices and whether it is a good idea to fully deploy an application into microservices.

The research questions we will try to answer from our experiment and analysis of literature, are

- Does decomposing into microservices impact the system's average response time?
- Is it always adequate to develop an app using the microservices logic?
- To what extent should the monolithic application be decomposed into a microservice?

A. *Design and Structure of Monolithic Applications*

A monolithic application describes a single-tiered software application in which the user interface and data access code are combined into a single program from a single platform. Schematically, this can be seen Fig. 1.

It is self-contained, and independent from other computing applications.

The design philosophy is that the application is responsible not just for a task but can perform every step needed to complete a particular function. Layered architecture is a common pattern seen in monolithic applications. This architecture allows for the technical capability to be changed fairly easily, especially if they are isolated to a particular layer [5].

The main idea behind this architecture is the separation of concerns, the main monolithic application components which include authorization, presentation, business logic and database are organized into four main categories or layers:

- The presentation layer contains all of the classes responsible for presenting the UI to the end-user or sending the response back to the client.
- The application layer contains all the logic that is required by the application to meet its functional requirements.
- The domain layer represents the underlying domain, mostly consisting of domain entities and, in some cases, services.
- The infrastructure layer (also known as the persistence layer) contains all the classes responsible for doing the technical stuff, like persisting the data in the database including DAOs or repositories.

An example of monolithic system architecture of real-world application is shown on Fig. 2. The diagram shows main components needed to build an Ecommerce application which authorizes costumer, takes an order, checks products inventory, authorizes payment and ships ordered products [6].

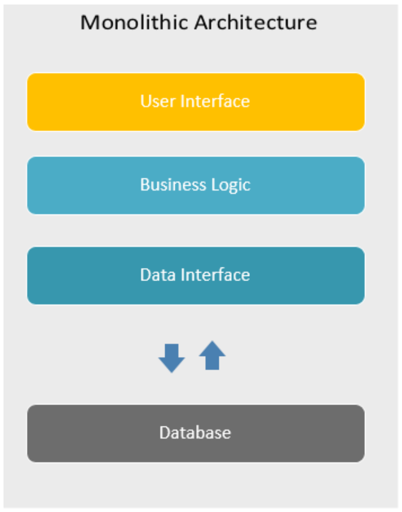


Fig. 1. Monolithic Application Architecture.

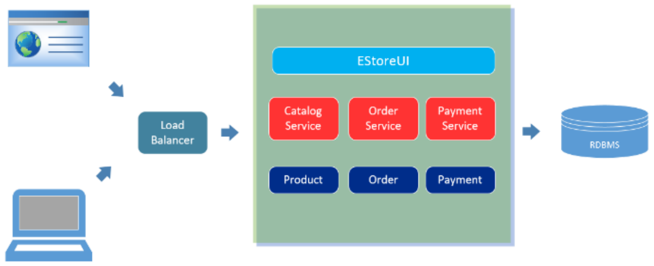


Fig. 2. Monolithic Architecture (Ecommerce Application).

Despite having many components which are independent from each other the system as shown in Fig. 2 is build and deployed as one application. With issues regarding maintenance, response time and scaling, monolithic architecture should be avoided when designing large and complex applications which may be used in different environments with different configurations or in applications which may change and need to be frequently updated.

This paper is structured as follows: Section II presents the state of the art, Section III methodology and results, Section IV case study and Section V conclusions.

II. State-of-the-Art

As mentioned previously, over the last decades, industry demands have pushed software design and architectures in various directions. The ever-growing complexity of enterprise applications, along with change and evolution management ushered in the rise of different architectures with an aim to replace or improve the traditional unified software designing model known as monolithic architecture.

Various architectures (besides the eminent ones) have been designed, researched, and used in industry, in recent years there has been a lot of hype regarding the new architectural model called microservice architecture. Considered new, microservice architecture has found itself being researched and compared a lot with existing architectures including SAO, serverless and monolithic architectures. Most of research studies were oriented on performance analysis, cost, and resource usage.

In a research that was done by Singh and Peddoju, the performance of a monolithic application is compared to a microservices application, the applications that were built were tested for their response time and throughput. Obtained results made it clear that microservices architecture has a better performance especially when it is used for a large number of requests [7].

Similar approach was used by the IBM research team in Tokyo, they compared the performance of the monolithic and microservices applications in different environments and configurations. The results were compared for throughput, scalability, number of CPU instructions for request and number of clock cycles to complete one instruction. The results showed a significant performance boost in monolithic architecture applications in many configurations and environments, which in a way contradicts the results shown by Singh and Peddoju [8].

Microservices are often compared to Service Oriented Architecture. The research paper done by Cerny, Donahoo and Pechanec compares and analyses microservices, service-oriented architecture and self-contained systems in terms of service and architecture, characteristics, integrations, capabilities, and flexibility. The drawn conclusion presented at the end of the paper favorizes SOA for large systems with many shared components and suggests using microservices for medium distributed systems which may need to scale in the future [9].

A different approach was used on research paper done by Chen, Li and Zheng from Nanjing University. This paper discusses ways to decompose a monolithic application to microservice architecture. Throughout the paper the researchers used a top-down analysis approach and developed a dataflow-driven decomposition algorithm. They defined a three-step procedure for process decomposition involving business requirement analysis, usage of dataflow-driven algorithm and individual modules extraction [10].

According to the fourth annual Developer Ecosystem Survey conducted by JetBrains, about 85% of 19,696 developers who were surveyed in the beginning of 2020, use the microservices-based system design [11]. The programming languages of choice for building microservices are JavaScript and TypeScript; REST APIs are used for communication between microservices the most, whereas the favorite cloud provider for microservices is Amazon Web Services, as shown in [12].

Improving scalability and improving performance are two of the most important topics when it comes to microservices. In the State of Microservices 2020 research project [12], over 650 developers (CTOs, Lead Developers, and Senior Developers) were asked to rate in scale 1-5 how they enjoy working with microservices when it comes to different aspects.

As shown in [2] Table I, most experts are happy with microservices for solving scalability issues, whereas maintenance and debugging seem to be a challenge for them.

TABLE I. WORKING WITH MICROSERVICES

Category	**Average rating (1-5)**
Setting up a new project	3.8
Maintenance and debugging	3.4
Efficiency of work	3.9
Solving scalability issues	4.3
Solving performance issues	3.9
Teamwork	3.9

Regarding security, there are still many challenges due to the complexity of the developments, the hardness of monitoring, and debugging and auditing of the full application in foreign environments [13].

Before moving to microservices, we should be aware of the architectural challenges. Some of the main architectural challenges, as presented in [14], are:

1) Dispersed business logic – microservices approach distributes the operating logic and execution flow of complex features among many applications.

2) Lack of distributed transactions – attempting to maintain consistency among many microservices involved in business transaction is extremely complicated.

3) Inconsistent dynamic overall state – it is related to lack of distributed transactions. Overall consistency gets more complicated with data that is geographically distributed data within the same domain because of sharding and data replication.

4) Difficulty in gathering composite data – joining data for analytics of the overall system in a microservices architecture is not straightforward.

5) Difficulty in debugging failures and faults – attempting to pinpoint the source of an error might require debugging multiple applications. Identification of the root cause of the problem is difficult primarily because of deep hierarchies of microservices (AC1) and the inability to determine the exact state of the system (AC3).

6) Difficulty in evolving – software evolution is a hard concept in an environment different where parts of the system evolve continuously, in parallel.

III. Methodology and Results

This section gives an overview on which methods and tools were used.

The goal of this section is to offer a way of passing between monolithic architecture to microservices approach and comparing them. So, we are going to demonstrate how to identify key design issues of monolithic applications and how they should be reflected in microservices approach. For that purpose, we will use a monolithic application that is developed in Model-View-Controller approach, which is based on monolithic architecture, and we will try to offer a way of decomposing it in microservices approach.

We are aware that there are a lot of design patterns that exists for developing web applications. But based on usage we have decided to use MVC as one of most used architectural patterns for developing web applications that are based on monolithic architecture and not only.

A. *E-Shop Monolithic Application*

As we said earlier, we will use an application that uses MVC approach, which is developed in Asp.NET Core with MVC approach. Before analyzing this application, we want to make purely understood that the term “monolithic”, in this context refers to the fact that these applications are deployed as a single unit, not as a collection of interacting services and applications [15].

Application that we have developed for this paper is based on application of Microsoft [16], for e-shop. The main reason why we have chosen to develop an e-shop application is to demonstrate how to pass between monolithic to microservices is because there is an almost perfect example that microservices should be used there.

In Fig. 3, we have presented schematically controllers of the application that are developed.

As is can be seen there are four controllers that monolithic application currently has. First controller, Order, is for handling requests that are for ordering items on application. Second controller, Product, it is used for managing products. The third controller, Home, is for main and privacy terms. The last controller which is default controller for authentication and authorization is Identity, it used to manage accounts and roles. In Fig. 3 we have presented Identity as a Controller, but in latest version of Identity Microsoft uses Razor pages for this module, but we will abstract this, and we will consider as a controller.

In Fig. 4 we have presented schematically structure of application that is developed as a monolithic application in Visual studio.

As it can be seen from Fig. 4, all application logic, including presentation, business and data access logic is in one place.

In the next section we will offer a way of passing microservices approach and how we should identify parts of application that should be microservice itself.

B. *Decomposing to Microservices*

In this section we will try to offer a way of how to decompose E-shop application to microservices approach.

Before starting to identify microservices we want to make purely understood that there is no general method that can be applied to every monolithic application. This means that we need to study very deeply application before architecting to microservices.

For E-Shop application, the first thing that must be transformed to microservice is Identity service, which is used for authentication and authorization purpose. One the most important services in E-Shop application, and in most applications, is security. Identity service is an IdentityServer4 [17], which is a typically used for managing authentication and authorization in microservices environment. Typically, IdentityServer4 acts as a middleware [18] that adds the spec compliant OpenID Connect and OAuth 2.0. With IdentityServer4 all access to microservices can be managed and this service is responsible for generating access token for clients.

Other important microservice for E-Shop application is product microservice, which is responsible for managing products for this application. So, this microservice can register new products, edit them, or see details about products. So, this microservice does only one thing but it does in a perfect way.

Last microservice is responsible for handling orders of customers. So, this microservice is focused only on processing orders, and offers a payment for orders.

For testing purpose is developed a client which will use microservices over RESTful API [19]. A schematic presentation of E-Shop application decomposed to microservices is displayed in Fig. 5.

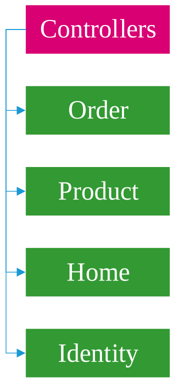
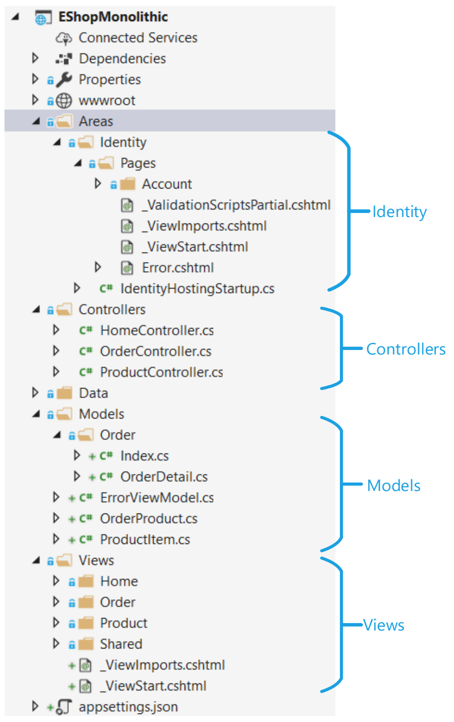


Fig. 3. Controllers for E-Shop.



```
◢ EShopMonolithic
    Connected Services
  ▷ Dependencies
  ▷ Properties
  ▷ wwwroot
  ◢ Areas
    ◢ Identity
      ◢ Pages
        ▷ Account
          _ValidationScriptsPartial.cshtml
          _ViewImports.cshtml
          _ViewStart.cshtml
        ▷ Error.cshtml
      ▷ IdentityHostingStartup.cs
  ◢ Controllers
    ▷ HomeController.cs
    ▷ OrderController.cs
    ▷ ProductController.cs
  ▷ Data
  ◢ Models
    ◢ Order
      ▷ Index.cs
      ▷ OrderDetail.cs
    ▷ ErrorViewModel.cs
    ▷ OrderProduct.cs
    ▷ ProductItem.cs
  ◢ Views
    ▷ Home
    ▷ Order
    ▷ Product
    ▷ Shared
      _ViewImports.cshtml
      _ViewStart.cshtml
  ▷ appsettings.json
```

Fig. 4. E-Shop Application with Monolithic Architecture.

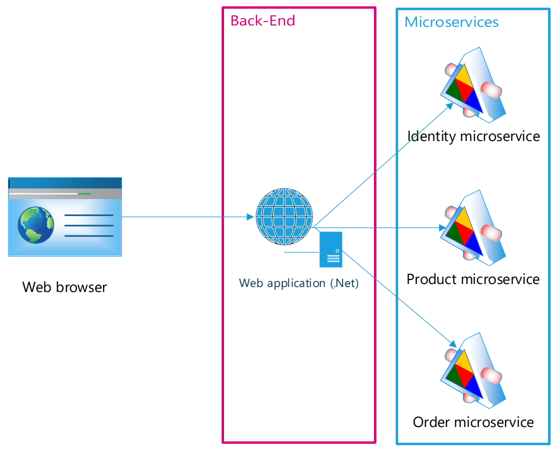


Fig. 5. E-Shop Application Decomposed to Microservices.

As it can be seen from Fig. 5, in this case we have 3 microservices, which we have described before. This decomposition offers a very good way to deal with scenarios where ordering a product is not possible, still application can offer service by listing all product that are there. So, with this decomposition we have archived a good way to handle problems with no function of order product, but order product currently contains functionality for checkout and payment. As part of comparison is this model of decomposition with E-Shop monolithic system, and other types of microservices architecture that will be presented.

As it can be seen from Fig. 5, the main problem with decomposition of E-Shop application in microservices architecture that is offered, is Order microservice, which needs to be decomposed to three microservices. These 3 microservices that will be derived from Order microservice are:

- Order microservice.
- Checkout microservice and.
- Payment microservice.

Schematically this decomposition is presented in Fig. 6.

With decomposition of Order microservice, are archived many things.

The last feature that will be applied when decomposing to microservices, in Fig. 6, is adding an API Gateway. Schematically this is presented in Fig. 7.

Decomposition that has been displayed in Fig. 7, contains an API Gateway, which acts as reverse proxy, hides functionality of microservices that are currently implemented in E-Shop application. This is a very good place to implement security for microservices.

C. Load test Comparison

In this section we will compare monolithic application with microservices for our fictive application. Comparison is made by using Apache JMeter [20] with different parameters. To have results that are comparable with each other we have hosted to Docker, with Linux container, all microservices, monolithic application and Client which consumes microservices is hosted in Internet Information services for Windows. For this purpose, we have deployed to test environment which is identic for microservices and monolithic application. Database is in Microsoft SQL server and contains same tables for both applications. Architecture of infrastructure for monolithic and microservices is presented in Fig. 8.

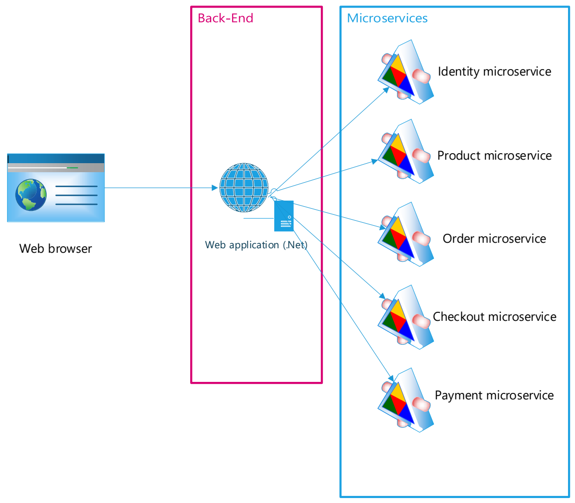


Fig. 6. E-Shop Application Decomposition Second Version.

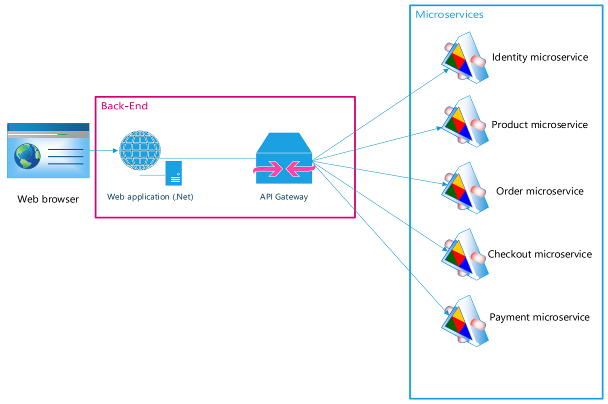


Fig. 7. Decomposition that has API Gateway.

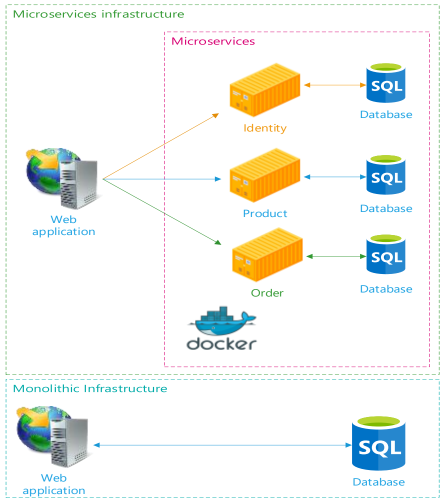


Fig. 8. Infrastructure of Microservices and Monolithic.

The first scenario will perform Get request to home page, then to list of products and finally to edit product page. All three requests are Get requests. Parameters of testing are set same for all applications. Parameters in Apache JMeter are:

- Number of Threads (users) = 100.
- Ramp-up period(seconds) = 50.
- Loop count = 5.

After creating test plan in Apache JMeter, we have gained results as can be seen in Table II.

In second comparison, as additional will be added post request which is responsible for adding new products to database. Parameters for Apache JMeter are same as above. After creating test plan in Apache JMeter, we have gained result as can be seen in Table III.

The final comparison will be made to order part. There will be added get request for checkout, order detail for specific product, update to database number of orders and finish payment.

After creating test plan in Apache JMeter, we have gained result as displayed in Table IV.

TABLE II. RESULTS FOR FIRST TEST

Parameter\Application	Monolithic	Microservices First	Microservices Second
Request	Get	Get	Get
Samples	1500	1500	1500
Average	6	10	9
Min	2	6	6
Max	41	159	98
Std. Dev.	5.14	8.83	5.33
Error %	0.00 %	0.00	0.00
Throughput	30.2/sec	10.1/sec	10.1/sec
Received KB/sec	247.14	81.38	81.53
Sent KB/sec	3.78	1.17	1.17
Avg. Bytes	8368.6	8268.6	8273.7

TABLE III. RESULTS FOR SECOND TEST

Parameter\Application	Monolithic	Microservices First	Microservices Second
Request	Get, Post	Get, Post	Get, Post
Samples	2000	2000	2000
Average	872	1851	1219
Min	2	7	8
Max	5024	7931	6361
Std. Dev.	1161.98	1858.53	1428.10
Error %	0.00 %	0.00 %	0.00 %
Throughput	22.5/sec	18.0/sec	20.7/sec
Received KB/sec	2197.87	1855.62	2154.23
Sent KB/sec	4.50	3.54	4.08
Avg. Bytes	100061.3	105852.1	106604.9

TABLE IV. RESULTS FOR THIRD TEST

Parameter\Application	Monolithic	Microservices First	Microservices Second
Request	Get, Post	Get, Post	Get, Post
Samples	2000	2000	2000
Average	7	22	21
Min	3	6	6
Max	113	127	319
Std. Dev.	5.38	15.52	16.53
Error %	0.00 %	0.05 %	0.15 %
Throughput	40.4/sec	39.8/sec	40.1/sec
Received KB/sec	220.37	3576.13	2270.08
Sent KB/sec	7.61	8.90	8.95
Avg. Bytes	5591.9	91920.7	57980.4

Very important statistic that can be derived from Table IV, is average response time that is from First and Second microservice. Decomposing to Microservices of course that has many benefits, but sometimes benefits that can be archived from decomposing might hurt performance of the system. This is proved by results displayed in Table IV.

IV. CASE STUDY

In case study will be discussed for complex system, which is implemented in Kosovo, which is Health Insurance Fund Information System of Kosovo. Because of data sensitivity we have decided to not use this system to decompose to microservices approach, so we have used a fictive application.

Results that are archived by using fictive application are very important and there can be draw parallel with Health Insurance Fund Information System and other systems.

Based on results that are archived there should be made a tradeoff between current architecture that has this system, which is monolithic application and is developed in Asp.Net, to decompose to Microservices approach. Again, based on results from Results for First Test Table II, Table III and Table IV, is evident that decomposing to microservices would decrease average response time, but benefits that could be archived from microservices, especially for this system, are bigger than the average response time. Benefits that will be archived are same as mentioned in Section C of III.

V. CONCLUSIONS

It is obvious that microservices offer a lot of advantages compared to the traditional monolithic architecture. Many of the core functionalities of microservices were described throughout the paper. Our approach in this paper, was to analyze and then compare the same application but developed with the two architectural styles. From the results obtained we saw that microservices can increase the system’s average response time since there are different services that need to communicate and exchange information with one another. Testing for different parameters with Apache JMeter we saw the differences in response times between them. Results from Apache JMeter, for three cases, also told that not only response time, but also error rate is better than architecture based on microservices. On the other hand, architecture based on microservices performs better in number of Kilo Bytes send and received per second, in case when test scenario contains post method as can be seen from Table IV.

One big advantage of microservices, is that they are not tied to a programming language. They also overcome the cumbersomeness of dealing with databases as we saw while developing our fictive application. To conclude, choosing whether to use the monolithic or the microservices architecture is not always clear cut. It all boils down to the type of application and what the developer wants to achieve. Big applications will benefit from the robustness, efficiency, and the well-organized code that the microservices make possible.

REFERENCES

[1] R. Amen, "Monolithic vs Microservices architecture," 2017.

[2] A. Kharenko, "Microservices Practioner Analysis," January 2019. [Online]. Available: https://articles.microservices.com/. [Accessed 03 June 2020].

[3] T. Jack, C. Bredley and L. Casey, "Content Stack," 03 02 2018. [Online]. Available: https://www.contentstack.com/cms-guides/decoupled-cms/monolithic-vs-microservices-cms-architectures. [Accessed 27 05 2020].

[4] K. Telai, "Medium," 09 04 2019. [Online]. Available: https://medium.com/@kenlynterai/microservices-and-distributed-systems-36a90d5d8ce. [Accessed 26 05 2020].

[5] [Online]. Available: https://medium.com/@shivendraodean/software-architecture-the-monolithic-approach-b948ded8c333. [Accessed 30 05 2020].

[6] [Online]. Available: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63. [Accessed 31 05 2020].

[7] V. Singh and S. K. Peddoju. [Online]. Available: https://www. researchgate.net/publication/322001375_Container-based microservice architecture_for_cloud_applications. [Accessed 02 06 2020].

[8] T. Ueda, T. Nakaike and M. Ohara. [Online]. Available: https://dominoweb.draco.res.ibm.com/reports/RT0973.pdf. [Accessed 03 06 2020].

[9] T. Černý and M. J. Donahoo. [Online]. Available: https://www.researchgate.net/publication/320765439_Disambiguation_and_Comparison_of_SOA_Microservices_and_Self-Contained_Systems. [Accessed 05 06 2020].

[10] L. Chen, S. Li and Z. E. Li. [Online]. Available: https://www.researchgate.net/publication/323562483_From_Monolith_to_Microservices_A_Dataflow-Driven_Approach. [Accessed 07 06 2020].

[11] JetBrains, "Microservices," 2020. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2020/microservices/. [Accessed 29 January 2021].

[12] P. Mamczur, T. C. M. Mol and M. Nowak, "State of Microservices," THE SOFTWARE HOUSE, 2020.

[13] N. Mateus-Coelho, M. Cruz-Cunha and L. G. Ferreira, "Security in Microservices Architectures," in CENTRIS Conference, 2020.

[14] C. Rajasekharaiah, Cloud-Based Microservices: Techniques, Challenges, and Solutions, Suwanee: Apress, 2021.

[15] Microsoft Developer Division, .NET, and Visual Studio product teams, "Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure," in Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure, One Microsoft Way, 2020.

[16] Microsoft, "Github," Microsoft, 11 May 2020. [Online]. Available: https://github.com/dotnet-architecture/eShopOnWeb. [Accessed 11 June 2020].

[17] B. A. &. D. B. Revision, "IdentityServer4," [Online]. Available: https://identityserver4.readthedocs.io/en/latest/. [Accessed 1 June 2020].

[18] R. A. a. S. Smith, "Microsoft," Microsoft, 5 June 2020. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-3.1. [Accessed 5 June 2020].

[19] E. J. R. E. R. Fielding, "ietf," ietf, June 2014. [Online]. Available: https://tools.ietf.org/html/rfc7231#section-4. [Accessed 07 June 2020].

[20] Apache JMeter, "Apache," Apache, [Online]. Available: https://jmeter.apache.org/. [Accessed 11 June 2020].