e-Commerce Mobile Systems Development with Enhanced Microservice Architecture Design

Mohamed Saif Ali Khan Bin Mohamed Aowtab Khan
Computing Science
Singapore Institute of Technology (SIT)
Singapore, Singapore
2001559@sit.singaporetech.edu.sg

Xiaorong Li
Information and Communication Technology
Singapore Institute of Technology (SIT)
Singapore, Singapore
xiaorong.li@singaporetech.edu.sg

Abstract

This paper investigates the suitability of using microservice architecture over monolithic for e-Commerce mobile applications. It examines past studies on this topic and provides a more detailed comparison between the two architectures, especially with their development processes. It also tackles the common issue of complex testing environments found in microservice systems by proposing a new feature called the Core Logger. Comparative study has been conducted to compare two e-Commerce systems with the same set of features based on a monolithic backend and a microservice backend. The microservice system contains the proposed Core Logger feature, which prints out the messages from all services in a sequential manner. The results showed that while the microservice system was initially harder to develop, it offered better overall scalability in development and performance. The Core Logger can reduce the time taken significantly during testing for the microservice system. The paper concludes by recommending the initial usage of monolithic architecture before migrating to a microservice backend for its better scalability, along with the implementation of the Core Logger which helps in expediting the testing process and shortening the overall development period.

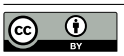
CCS Concepts

• **Information Systems**; • **Information Systems applications**; • **Mobile Information Processing Systems**;

Keywords

Microservice architecture, e-commerce, mobile application development

ACM Reference Format:
Mohamed Saif Ali Khan Bin Mohamed Aowtab Khan and Xiaorong Li. 2024. e-Commerce Mobile Systems Development with Enhanced Microservice Architecture Design. In *2024 the 12th International Conference on Information Technology (ICIT) (ICIT 2024), December 13–15, 2024, Kuala Lumpur, Malaysia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3718391.3718405



ICIT 2024, Kuala Lumpur, Malaysia

ACM ISBN 979-8-4007-1737-6/2024/12
https://doi.org/10.1145/3718391.3718405

1 Introduction

The e-commerce industry has rapidly expanded in the past years thanks to its convenience for customers to have anything they want delivered to their doorstep. The market has grown fiercely competitive as a result, and companies must ensure their system is quick to adapt and improve to ensure customer retention and satisfaction. With that key factor in mind, one must consider forgoing the use of a traditional monolithic design for their e-commerce application due to its numerous flaws that will hinder its growth. Instead, microservice architecture is much more suitable for such a design philosophy. However, companies must be wary that microservices come with their own set of problems that must be addressed.

Monolithic architectures are seen as the standard design for software systems, with one survey showing 75% of companies still utilizing it [1]. Monolithic systems have their features and functionalities tightly linked together, which allows for easier development and monitoring of the system [2]. However, there are numerous issues that can arise which can prove detrimental for an e-commerce system when using monolithic architecture.

One of the major drawbacks of monolithic architecture is the lack of scalability and flexibility that it offers. As applications grow in codebase over time, it becomes more difficult and cumbersome to make even small changes without the risk of an unintended ripple effect throughout the system [3]. Using such architecture also results in an inability to adapt and expand system features quickly. One study emphasized the need for e-commerce companies to possess various services and be highly scalable to maintain customer retention [4]. While attempts have been made to make monolithic architecture easier to scale using virtual servers and load balancers [5], the cost does not justify its continued use over microservices.

Due to the high cash flow that e-commerce systems handle, security and reliability are some of its most important factors. This can prove to be a major issue for monolithic systems when updating dependencies. When updating the libraries of a monolithic application, it requires the entire program to be rebuilt and deployed once more. One study mentions that a lack of updates to these dependencies will increase the system's security risk and vulnerability [6] and as such cannot be ignored. However, another study states that even one change in a script can affect the entire application on a large scale [7]. This results in the arduous task of checking the entire codebase when updating the system's libraries to ensure its reliability is not hindered. While a different study has attempted to improve reliability in monolithic systems [8, 9], they are still at a great disadvantage compared to microservices.

While some mobile applications now use microservice architecture, a common problem they face is its complex testing environment. Due to the scattered nature of microservices, it becomes more difficult to conduct testing for microservice systems compared to ones that are monolithic [10, 11]. If a system fails to prepare its testing beforehand, it can lead to long development cycles as the number of microservices grows, lowering its flexibility and hindering the growth for an e-commerce system. Microservice systems inherently face this issue due to the independent nature of its architecture, a problem which this paper aims to tackle.

The goal of this work is to evaluate the benefits and drawbacks of microservice architecture over a monolithic architecture while assessing its suitability for e-commerce mobile systems. We aim to improve the essential elements of an e-commerce application by suggesting a transition from traditional monolithic design to microservice architecture. We tackle the issue of the complex testing environments commonly found in many microservices systems by implementing its own Core Logger feature. The Core Logger provides a simple testing environment where information regarding each service is transmitted through a single component of the system, greatly expediting the testing process for microservices and reducing the development time.

2 Literature Review

There were numerous studies available that discussed the use of microservice architecture for e-Commerce systems over monolithic ones. However, as their focus was on discussing the benefits of microservices, studies directly compared between the two different architectures were also looked at to possibly get a less biassed view on the matter. There were also many studies that found that suggested improvements on the microservice architecture to patch its flaws, some of them intended to directly improve e-Commerce systems which used the architecture.

A direct comparison in suitability for e-commerce systems was made in a previously mentioned study which tested the different architectures in multiple different factors [4]. The microservice application had a varied performance for its loading times, loading faster in some aspects but slower in others compared to its monolithic counterpart, while also maintaining a high reliability throughout its stress test. The study concluded by stating that the changes in work environment required to migrate to a microservice architecture are considerable enough that a modular monolith system should be considered instead. It's important to note, however, that the study is addressing migrating an existing system architecture [4] and not the development of a new system.

One study evaluated the performances of both architectures to determine which is more suitable for an e-Commerce startup to use [11]. Two compact versions of e-Commerce systems were built with similar functionality for the study, one using monolithic and the other using microservices. After deployment, the systems were evaluated via load tests. Other factors such as difficulty in development between the systems were compared as well. Results from the study showed that the monolithic structure was more suitable for an e-Commerce startup due to the initial lower costs and easier development period. However, the paper acknowledges that microservices were much more suitable to use when expanding and recommended migrating to such an architecture when scaling up.

In another study, analysis was conducted to determine the appropriate architecture to use for e-commerce systems [13]. The study examined other studies regarding the development of high and low-load systems similar to e-commerce. The results suggested that a mixed architecture design is the most appropriate, where monolithic can be used for minor parts of the system while microservices can be used to manage the system sections that need to be scaled later. This is due to the microservices' ability to easily scale, with only the additional costs being its downside.

A separate study performed a comparative analysis of different software architectures used in e-Commerce to determine their characteristics and whether it matches the requirements needed for such systems [14]. The paper analyses monolithic, microservice, and serverless architecture using research obtained from other studies as well as its own observations. Using these findings, it set out to analyse the scalability, reliability, costs, and development speed of these architectures. The findings from this research concluded that e-Commerce systems stand to benefit the most from utilizing microservice architecture due to its ability to scale each service independently to handle service loads along with it being easy to deploy. It is important to note, however, that the study does not provide statistical proof for its findings and determines its results based on its own simple observations of the architectures.

One study aimed to identify the benefits of microservices and DevOps and why they are preferred over monolithic architecture for larger software systems [15]. By developing their own mini-application with both monolithic and microservice architectures, they deduced many comparisons between the two. The results showed that the monolithic app proved to be more difficult to develop when it grew in size, while the microservice app was easier to work with as it grew whilst boasting a higher performance.

Another study analysed and compared the difference in properties between both architectures [16]. The study developed a web system using WordPress before deploying it using both monolithic and microservice architecture. Numerous tests examining the application's speed, reliability, and costs were carried out with load balancers that measured the success rate and time to complete requests. Its findings concluded that microservices were more reliable and easier when expanding the system compared to monolithic architecture, which was found to be simpler to develop along with a better performance. The study concludes by recommending microservices for systems that require scaling and monolithic for smaller systems.

Many studies have proposed numerous improvements for microservice architecture. One such study proposed the use of machine learning for an e-commerce microservice system's framework [17]. This new addition to the framework will allow services to collect data and extract user patterns and behaviour that can then be used to cater and personalise to the user. An example used in the study is the product catalogue service, which can monitor and analyse user behaviour with items. Once the model is trained, it will provide the service with items that it should recommend to the user based on their preference. While an increase in response time was shown when testing a prototype of this feature, it was

noted that utilising asynchronous requests with the service and introducing preprocessing for the model can improve it.

As microservices systems consist of many services working together, it is important that deployment is handled carefully to allow for seamless interaction. One study proposes a framework which allows code to be shared easily across multiple microservice instances within the system [18]. This is done by having a primary microservice that handles copies as well as users. It allows for quick code-sharing between the services as well as requests sharing from the users, which the results from experimentation showed. The study concluded with the framework being proposed to help simplify the work between services with code-sharing and only needing to maintain a single codebase. However, it must be noted that this implementation of a single codebase for the microservice system [16] will introduce problems similar to monolithic architectures where the system can grow too complex and become difficult to scale, even more so with high reliability.

Another suggested improvement from a study is implementing a serverless aspect to the microservice system [19]. With the use of Amazon Web Services (AWS), the study proposes an entire e-commerce microservice platform built without using servers, including details such as order processing and payment handling. The final conclusion and results showed a prototype design of the serverless mobile system. While the study mentions many advantages in a serverless microservice system such as cost-effectiveness and higher performance, there is little numerical data behind these results.

The various improvements that have been suggested and made to the microservice architecture are important to note as well. However, there are aspects of microservices that these studies fail to cover fully. Most studies that compared the two pieces of architecture mention only the performance aspect of the microservices with not much acknowledgement to their development process. Meanwhile, the studies that discuss improvements for microservices do not propose ways to mitigate the drawback of having a complex testing environment.

Our work aims to cover the comparison between monolithic and microservice architectures in terms of their suitability for e-commerce mobile systems. In addition to scalability and flexibility, it also plans to analyse the difficulty when developing and scaling up the system as few studies cover such an aspect. We also propose further improvements for the microservice architecture by implementing a new feature called the core logger. This feature aims to tackle the issue of handling complex testing environments commonly found in microservice systems, as there are a limited number of studies that discuss the flaw let alone address it.

3 Proposed Architecture Design

3.1 Monolithic System

1) Framework: The monolithic system of the e-Commerce application is developed on Visual Studio 2022 using the ASP.NET Web Forms framework. ASP.NET has long been renowned to be a powerful tool for web development due to the fast performance and high security it can provide for web applications. The Web Forms framework allows for quick and easy development thanks to the state-driven model it employs, which makes it easy to manage user interactions and change web page elements accordingly.

2) Software Architecture: The architecture of the monolithic system consists mainly of two areas, the server section and the database section as well as a mobile section. The architecture diagram is shown in Figure 1.

The Mobile section holds the mobile interface which serves as the gateway for users to interact with and utilize the e-Commerce system. The mobile interface is then connected to the web server which holds the e-Commerce system features itself. Once the e-Commerce system is fully developed using the ASP.NET Web Forms framework, it is then deployed onto an Internet Information Services (IIS) server. The server is deployed as a single unit containing all the features and logic of the application which are tightly interconnected and dependent on one another.

3) Database Implementation: The database of the monolithic system is handled by the Microsoft SQL server system. This is due to the native support that ASP.NET provides for Microsoft SQL, making it easy to develop code that reads from and writes to the database server in a fast and secure manner. The database server stores all information required for the application to operate, such as users, items, and orders made. The data is split into their tables based on its type and is then accessed by the application to communicate with, such as the Login feature reading from the users table to verify identification.

While monolithic systems make development and deployment quick and easy, they are also highly vulnerable to failure should any of the features start raising issues. In addition, any changes made to the features present in the system required the whole application to be redeployed again.

3.2 Microservice System

1) Framework: The microservice system is developed on Visual Studio 2022 but with the ASP.NET Core Razor Pages framework instead. A different framework is used for the microservices system because the Web Forms framework does not allow Docker containerization without the purchased use of an Enterprise version of Windows. In concordance with the current state of the art where most microservices utilize containerization to deploy their systems, an alternate framework was required. The Razor Pages framework in that regard was chosen as it supports Docker containerization and is regarded as a modern alternative to Web Forms by providing similar functionality but with easier syntax implementation.

2) Software Architecture: Unlike the architecture of the monolithic system, the microservice system consists of many sections to account for each feature being fully separated from one another. The architecture diagram for the system is as shown in Figure 2.

Similar to its monolithic counterpart, the Mobile section consists solely of the mobile interface serving as the gateway for users to interact with the e-Commerce system. In this case, however, the mobile interface interacts with only one aspect of the developed e-Commerce system which is the Main Service.

As shown in Figure 2 , the Main service acts as the centre point of the microservice system. Its purpose in that regard is similar to that of an Application Programming Interface (API) gateway as it organizes and processes the requests sent between the user and all

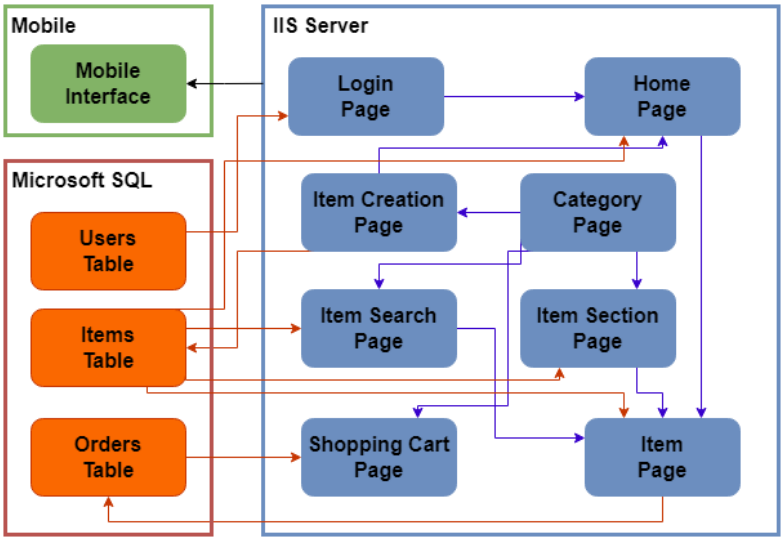


Figure 1: System architecture diagram for the monolithic e-Commerce application

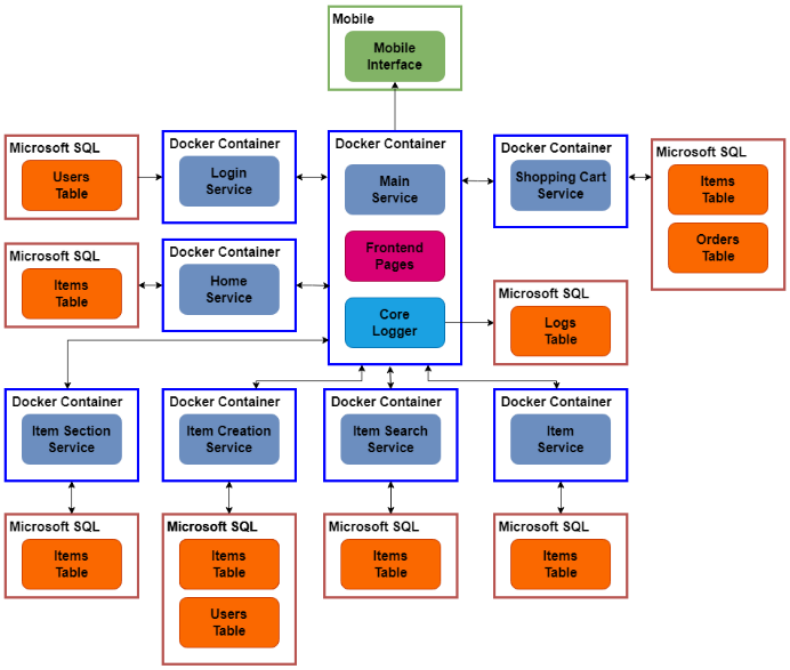


Figure 2: System architecture diagram for the microservice e-Commerce application

other feature services in the system. The Main service also holds the frontend pages as well, which is how the user interacts with the web system. This includes the Category page which simply acts as a redirect hub to other pages in the system.

3) Service Containerization: Following the principles of microservice architecture, each feature in the e-Commerce design is separated into their individual codebases and databases. Docker containers are used to deploy the individual services which simplifies the deployment process, allowing developers to quickly set the service servers without much configuration necessary. Containerisation allows for effective isolation of the services from one another. In addition, Docker was the main choice among the studies to containerise their system due to how particularly lightweight it is compared to other containerisation technologies.

4) Database Implementation: For the database aspect of the microservices, Microsoft SQL servers are also used to manage the service databases. Each service has its own database in accordance with microservice structure, with it strictly holding only whatever information is needed from its service and nothing more to maintain the lightweight nature of microservices.

5) Service Orchestration: An orchestration-based Saga pattern [20] is also implemented to address the common flaw of data consistency found in microservice systems. Whenever a database change is required across multiple services, the change is made locally by each service in a sequential order until all services have performed it. If any transaction fails along the sequence, a compensating transaction is then conducted to then undo the changes made, likewise in a sequential order as well. An orchestration-based Saga pattern is where a single orchestrator is in charge of managing the entire transaction process, communicating with each service affected to commit and undo any changes necessary. Maintaining consistent data across all services is of utmost importance for e-commerce systems that utilise microservice architecture, so it is highly beneficial to follow the Saga pattern for this system.

The Main service functions as the Saga orchestrator for the services, sending out orders to update the databases across all services when needed to. One example of this is when a user checks out an order they've made. The Main service receives the order confirmation and proceeds to contact each service that handles item data to update the stock amount, only continuing the chain once it has received confirmation from the service. If any service fails to update its database, the orchestrator sends out another chain of queries to undo the changes that were previously made.

6) Core Logger Implementation: The microservice system also consists of a new proposed feature called the Core Logger. The Core Logger functions as a universal message logger for all services active in the web server, with each service sending its records to the component. This additional feature is introduced to greatly reduce the testing time and difficulty of testing the microservices by making the errors generated easily identifiable by noting the sequence of events that occur. The Core Logger is attached to the Main service as it communicates with every service in the system, thus making it easy to retrieve the records from them.

The main purpose of the Core Logger is to tackle one of the most common issues found in microservices systems which is the complex testing environment it has. Studies that were previously mentioned [2, 3, 7, 8] noted the difficulty in conducting effective testing for microservice architecture due to its distributed nature. Integration testing was noted in this regard, where interactions involving multiple services are more difficult to monitor as tests are concerned solely with the microservice it is related to. Hence, by proposing the Core Logger, the testing process for microservices becomes more streamlined and easier to observe, especially for integrated testing.

The manner of microservice systems where services are independently developed and deployed from one another allows systems to individually work on its features without the risk of affecting other components. This greatly increases the scalability and flexibility of web projects, which is a major factor when working on e-Commerce systems. While there are some drawbacks in microservice architecture, solutions to overcome such challenges can be implemented such as the Saga pattern to maintain data consistency and the proposed Core Logger for a more simplified testing environment.

4 Implementation and Testing

Development Process: The development process of the e-Commerce microservice system was vastly different and more difficult than that of the monolithic system. This is due to the architecture having its own set of unique challenges that the development of a monolithic system wouldn't face. These issues are ones commonly faced by developers when working on microservice systems such as designing the services architecture, minimizing inter-service coupling, and maintaining data consistency. While such drawbacks can be negated with the use of modern techniques, their implementation can be difficult and complex.

An example of this is the Saga pattern that helps in the synchronization of data across microservices in the e-Commerce system. While the pattern is very effective in maintaining data consistency, it is also notably complex with the amount of interactions that occur between the services and the Saga orchestrator. An example of a Saga pattern-implemented transaction where the user checks out their order can be seen in Figure 3.

The Cart service first updates its own stock before creating the pending order and sending its response to the Saga orchestrator. The orchestrator then has to sequentially update the stock of the Home, Item, Item Searching, and Item Section services before then confirming the order. If any service fails to update its stock quantity, such as the Item service in the example figure above, the orchestrator then has to contact the previous services to rollback their item quantity as well and cancel the order. As this action needs to occur for every item in the user's order, the ability to rollback successfully ordered items too. For an example, assume a user is ordering items A and B at the same time and checks out their order. The Saga orchestrator successfully completes the order process for item A but fails it for item B. The orchestrator would then now have to not only rollback the stock quantity for item B but for item A as well as they are part of the same order.

The implementation process for such a pattern solely to ensure data consistency can be highly complex as well as resource intensive for the orchestrator. When compared with the monolithic system which doesn't have to deal with such issues, it's clear that the initial development process for microservice is more difficult. However, while the implementation process for the microservice system was harder, it provided for an easier process when making changes to the features in the system compared to its monolithic counterpart. This is due to the isolating and independent feature of the services that allows it to be worked on without the worry of inter-feature interaction that would have to be considered in a monolithic system.

Performance testing for both monolithic and microservice systems is done with the use of JMeter, an open-source software that allows developers to easily configure web tests of their own. A custom test scenario was designed, simulating a user flow of the application from start to end. The test scenario starts at the login page of the system and includes the following steps:

1. Login with a correct set of user credentials to access the Home page.

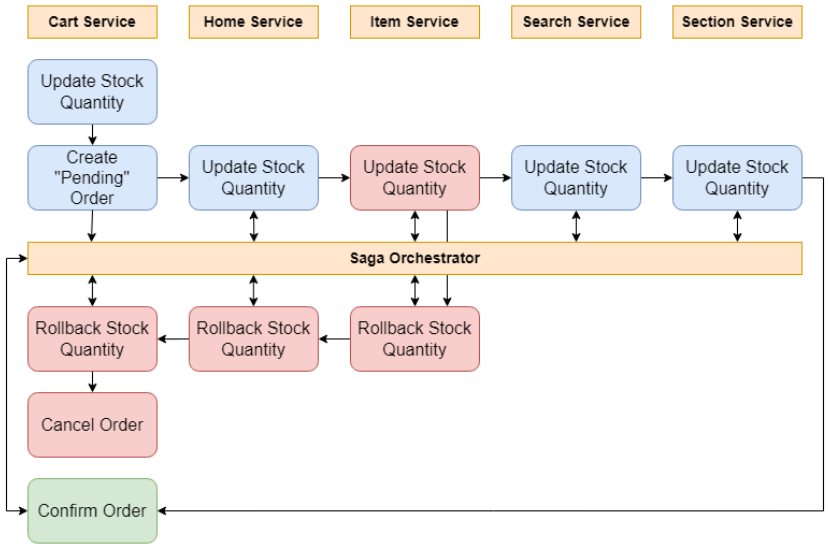


Figure 3: Flow chart for the orchestration-based Saga pattern of an item checkout transaction

	LOG_ID	SERVICE_NAME	STATUS	MESSAGE	LOGGED_AT
13...	312	HOME	OK	New item ID 1000007 has been successfully posted.	2024-07-11 01:19:15.107
13...	311	CREATE	OK	Item sucessfully created at PENDING stage.	2024-07-11 01:19:14.990
13...	310	CREATE	ERROR	Item posting with ID 1000006 failed, status has been set to FAILED.	2024-07-11 01:16:44.687
13...	309	HOME	ERROR	Error encountered when posting item to database. Must declare the scalar variable "@ItemN...	2024-07-11 01:16:44.667
13...	308	CREATE	OK	Item sucessfully created at PENDING stage.	2024-07-11 01:16:44.593
13...	307	HOME	OK	Item list successfully retrieved for Home.	2024-07-11 01:16:25.393
13...	306	LOGIN	OK	Login approved.	2024-07-11 01:16:25.257
13...	305	CREATE	ERROR	Item posting with ID 1000005 failed, status has been set to FAILED.	2024-07-11 01:13:59.750
13...	304	CREATE	OK	Item sucessfully created at PENDING stage.	2024-07-11 01:13:59.730
13...	303	HOME	OK	Item list successfully retrieved for Home.	2024-07-11 01:13:39.553
13...	302	LOGIN	OK	Login approved.	2024-07-11 01:13:39.047

Figure 4: Logs generated by Core Logger showcasing the sequence of processes by each service

2. Select one of the item listings displayed on the Home page to access the Item page.

3. Place down an order of two copies of the item and be redirected to the Cart page.

4. Press the Checkout button and wait until the system confirms the order.

Using JMeter's load testing capabilities, the test scenario is run with multiple concurrent users to test the performance of the systems under heavy load. The time taken for each simulated user to complete the testing scenario is then recorded to provide statistical evidence on the speed of each system and analyse the difference in performance between architecture types.

5 Performance Evaluation

Core Logger Evaluation: The Core Logger was implemented into the Main service component shortly after work on the e-Commerce microservice system began. After its implementation, the Core Logger was used constantly as a debugging tool for the microservices to identify the cause of an error or issue. The Core Logger greatly reduced the amount of time spent debugging the microservices and as such, sped up the overall development process of the system. An example of the logs generated by the Core Logger is as shown in Figure 4.

System Performance Evaluation: Performance testing was conducted for both systems simulating an end-to-end user flow and handling multiple simultaneous requests. The time taken for each user to finish the testing scenario was recorded and the average, median, minimum, and median completion time was noted down. A table showcasing the results of the test is as shown in Table 1.

The results of the performance test as shown in Figure 5 indicate that the monolithic system has a shorter overall completion time than the microservice system. However, when the number of simulated users increases, there is a larger increase in the completion time for the monolithic system than there is for the microservices. The performance of the monolithic system decreases in a more rapid manner than microservice when its usage is further scaled up. This is especially true between the increase from 10 to 25 users, with the monolithic system having a 20-minute completion time as compared to the microservice system having a 13-minute longer completion time.

This was especially noted during the development of the Item Creation feature. During development, the choice was made to use SkiaSharp, a 2D graphics library, to resize uploaded item images to

Table 1: Performance test results for both systems

	Completion Times (ms)							
	Monolithic				Microservice			
No. of Users	Avg	Med	Min	Max	Avg	Med	Min	Max
5	76	73	65	100	129	130	124	136
10	80	80	63	94	133	127	118	176
25	100	62	45	329	146	146	129	173

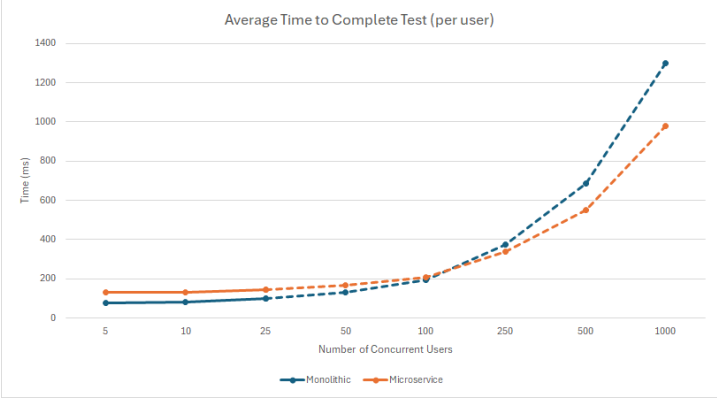


Figure 5: Scalability tests of monolithic and microservice based e-commerce mobile systems

a fixed dimension. When applying this library in the monolithic system, there were unforeseen effects as the library affected not only the resizing of the image in the Item Creation feature but also displaying the image in the Home feature. In comparison, no such issue was faced in the microservice system as the library was applied to and affected only the Item Creation service. This further demonstrates the benefits of microservices with their isolationism preventing unintended effects with the rest of the system.

The microservice system was developed carefully regarding covering as many possible test cases as possible. As information regarding the system must be sent to the services through the form of web APIs, it is important that the data can be properly processed when transferring between services. For example, all user inputs where the user is required to type in the field have a default value if it is submitted blank. This is because the service is unable to read a blank field when it is passed through the web API.

Core Logger Impact: One major and common issue that developers normally face when working microservice systems is the complex testing environments they have due to each service having its own separate codebase. The Core Logger was implemented to address this issue by funneling the messages sent by each service into an easily readable set of sequential logs. After its implementation in the microservice system, it was much easier to debug the system as the cause of error and the events leading up to it were easily identified by reading the logs.

The implementation of a Core Logger would prove to be highly beneficial to any e-Commerce mobile system that utilizes the microservice architecture. While there are some minor flaws with its implementation, the overall testing process is much improved thanks to its inclusion. By expediting the development process via quick identification of the root cause of an issue during debugging, e-Commerce companies can expand their application in a faster and more reactive manner.

6 Conclusion

In this paper, the use and viability of microservice architecture over monolithic architecture in e-Commerce mobile systems has been discussed. We designed and developed both monolithic and microservice e-Commerce prototypes and discovered that while the initial development process of the microservice system proved to be difficult compared to its monolithic counterpart, the process of adding newer features was found to be an easier process. The performance shows the monolithic system offers better performance for small scale systems but worse performance scaling than the microservice system.

To address the complexity of developing scalable e-commerce mobile systems, we propose the Core Logger, a new feature that expedites the testing process for microservice systems by logging all service messages into a sequential and easy-to-read set of records. The feature proved to be highly beneficial during the development of the microservice prototype as any errors caused by the services were quickly identified. The project thereby recommends its usage in future e-Commerce mobile systems as it speeds up overall development thanks to the easier and faster testing process it offers. The proposed Core Logger feature expedites the microservice programming process especially for its debugging and testing phases, thereby reducing the overall development time. Future work may

include a Core Logger with a message broker, eliminating the need for a service orchestrator.

References

[1] Tudoran, A. Breaking Down the Battle of the Architectures: Monolithic vs. Microservices. Kubeark. https://kubeark.com/resources/blog/monolithic-vs-microservices, 2023.

[2] Blinowski, G., Ojdowska, A., & Przybyłek, A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access, 10, 20357-20374, 2022.

[3] Kyryk, M., Tymchenko, O., & Pleskanka, N. Methods and process of service migration from monolithic architecture to microservices. 2022 IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 553-558, 2022.

[4] Asrowardi, I., Putra, S. D., Subyantoro, E. Designing microservice architectures for scalability and reliability in e-commerce. Journal of Physics: Conference Series, 1450, 2020.

[5] Velepucha, V., Flores, P. A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges. IEEE Access, 11, 88339-88358, 2023.

[6] Pashchenko, I., Plate, H., & Ponta, S. E. Vulnerable open source dependencies: counting those that matter. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18), 42, 1-10, 2018.

[7] ElGheriani, N. S. (2022). Microservices vs. Monolithic Architectures. Al-Mansour Journal. https://www.iasj.net/iasj/download/629f8ed5c54a1d0f, 2022.

[8] Milić, M., Makajić-Nikolić, D. Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures. Symmetry 2022, 14(9), 1824, 2022.

[9] R. Buyya, R. N. Calheiros and X. Li, "Autonomic Cloud computing: Open challenges and architectural elements," 2012 Third International Conference on Emerging Applications of Information Technology, Kolkata, India, 2012, pp. 3-10, doi: 10.1109/EAIT.2012.6407847.

[10] Söylemez, M., Tekinerdogan, B., Tarhan, A. K. Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. Appl. Sci. 2022, 12(11), 5507.

[11] Toomwong, Napawit & Viyanon, Waraporn. (2021). Performance Comparison between Monolith and Microservice Using Docker and Kubernetes. International Journal of Computer Theory and Engineering. 13. 91-95. 10.7763/IJCTE.2021.V13.1295.

[12] Nayim, N. N., Karmakar, A., Ahmed, M. D. Performance Evaluation of Monolithic and Microservice Architecture for an E-commerce Startup. 2023 26th International Conference on Computer and Information Technology (ICCIT), 2023.

[13] Yakovenko, V., Ulianovska, Y., Yakovenko, T. Analysing features of e-commerce systems architecture. Technology Audit and Production Reserves, 1(4(63), 27–31, 2022.

[14] Samoylenko, H. T., Selivanova, A. V. Features of microservices architecture in e-commerce systems. Mathematical machines and systems, 2023.

[15] Chouhan, U., Tiwari, V., Kumar, H. Comparing Microservices and Monolithic Applications in a DevOps Context. 2023 3rd Asian Conference on Innovation in Technology (ASIANCON), 2023.

[16] Selivorstova, T., Klishch, S., Kyrychenko, S. Analysis of monolithic and microservice architectures features and metrics. Computer systems and information technologies. https://doi.org/10.31891/CSIT-2021-5-8, 2021.

[17] Dobrița, G. Adaptive Microservices for Dynamic E-commerce: Enabling Personalized Experiences through Machine Learning and Real-time Adaptation. Economic Insights – Trends and Challenges, 12(1), 95-103, 2023.

[18] Chaplia, O., Klym, H. An Approach for Automated Code Deployment Between Multiple Node.js Microservices. 2023 IEEE 13th International Conference on Electronics and Information Technologies (ELIT), 2023.

[19] Athreya, S., Kurian, S., Dange, A. Implementation of Serverless E-Commerce Mobile Application. 2022 2nd International Conference on Intelligent Technologies (CONIT), 2022.

[20] Daraghmi, E., Zhang, C., Yuan, S. Enhancing Saga Pattern for Distributed Transactions within a Microservices Architecture. Appl. Sci. 2022, 12(12), 6242, 2022.