Quest Journals
Journal of Software Engineering and Simulation
Volume 9 ~ Issue 6 (2023) pp: 31-42
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Implementation of Microservices to Improve Performance and Security of Train Search System in Pegipegi Application

[1st] Nicko Sambrano Putra
Bina Nusantara University
Jakarta, Indonesia nicko.putra001@binus.ac.id

[2nd] Pieters Nicholas P.T.
Bina Nusantara University
Jakarta, Indonesia pieters.tambunan@binus.ac.id

[3rd] Fathanal Achsan
Bina Nusantara University
Jakarta, Indonesia fathanal.achsan@binus.ac.id

[4th] Dennis Rydarto Tambunan
Dehasen University
Bengkulu, Indonesia tambunandennis376@gmail.com

Abstract— *The purpose of this research is to know the results of the Train Search implementation system used in the Pegipegi application to search train schedules. The methodology of Train Search system development uses Spring Boot-based Microservices with Event-Driven and Reactive Programming paradigms. The test results from the Train Search system with microservices are more structured. So that makes easier maintenance and further development. The new Train Search system has improved search speed performance from 9 seconds to 2 seconds and improved security against vulnerabilities from 17 types to 12. Besides that, monitoring data is also added from 6 types of monitoring data to 12. Then it can be concluded from this research is the implementation of microservices in the Train Search system can improve performance and security in the Pegipegi application.*
Keywords— Web Services, Scrum, Spring Boot, Reactive Programming, Microservices, Event-Driven Architecture

Received 15 June, 2023; Revised 28 June, 2023; Accepted 30 June, 2023 © The author(s) 2023. Published with open access at ***www.questjournals.org***

I. INTRODUCTION

Advances in technology can not be divided from the the rapid development of technology. The reason of technology is growing because the help of internet service factor, which is getting better and more reliable for these technological needs. It cannot be denied that at this time many enterprises are competing to create solutions for problems or needs that exist in the community or commonly called startup companies.

Many of startup companies are trying to solve the problem in the community environment which aims to help the community to help more efficiently than before. One of the example is Pegipegi which is a startup based on growing online travel agencies in Indonesia. In the process of developing systems for the application of Pegipegi, there is a large system that runs to support all the needs of the application (monolith application), and the system is broken down into domains according to existing business requirements (service oriented architecture). Each of these domains is broken down into sub-domains according to existing business needs, one of which is the Train Search feature in the Train domain.

Service migration is a very common thing in developing systems that have implemented SOA or Microservices. There were several factors that made the team decide to migrate existing services. Some of these

reasons, such as outdated technology, massive service improvisation or system paradigm changes. In addition to changing the technology ecosystem for the better, according to the standards set by the company. Migration is also carried out to improve security for the better. It is known that a security company named **Snyk** collected vulnerability data on the technology used by previous legacy systems. This is also one of the reasons for doing this migration. In addition, this migration also aims to remove legacy code (code that is obsolete and difficult to maintain) that exists in the Train domain.

This research will explain the migration process of the Train Search feature in the legacy code to become a stand-alone service and implement the company's technology standards. So that with the migration of the Train Search feature from the legacy system, there will be microservices that support the Train Search feature, namely "Train Search" and "Train Search Worker".

1.1. Problem Statement

Based on the introduction above, broad problem formulation can be concluded as follows:

1. How can service migration reduce the load on legacy systems?
2. How can service migration improve performance in searching and managing train schedule data?
3. How can service migration improve security in the Train Search feature?
4. How can service migration improve monitoring of data search and availability by suppliers?

1.2. Scope and Limitation

When develope Train Search and Train Search Worker services, the author sets limitations on the discussion to be more focused, as follows:

1. Development of this service only consists API endpoints that will be accessed by Pegipegi applications in the Train domain.
2. Development of the Train Search and Train Search Worker services uses Spring Boot by implementing Reactive Programming paradigm.
3. Database design uses PostgreSQL.
4. Caching storage uses Redis.
5. Message broker implementation uses Apache Kafka.

1.3. Objective

The objectives of this development are:

1. Reducing features in the legacy Train system and developing the Train Search feature into a more independent service, also separating the responsibilities of the Train Search feature to be more clear, namely Train Search and Train Search Worker.
2. Improve the performance in schedule searching by maximizing the available routes from suppliers.
3. Improving security level in the Train Search service by reducing vulnerabilities that have Critical, High, and Medium level gaps, and avoiding vulnerabilities that have Low-level gaps.
4. Maximizing other indicators in monitoring data, requests, and available responses .

II. LITERATURE REVIEW

2.1. Scrum

Scrum is an incremental software development model that is a subset of agile software development. It is particularly useful in managing uncertain conditions and meeting tight deadlines. Scrum consists of three essential roles: product owner, development team, and scrum master. Its applications are widespread, ranging from developing educational mobile applications to integrating document management systems, among others. The effectiveness of Scrum depends on the expertise and proficiency of team members. Scrum entails certain risk factors that necessitate risk management approaches. Additionally, a comparison between Scrum and the traditional waterfall model is often discussed. Scrum's performance can be measured through various testing approaches, and it helps ensure timely delivery of all software components. Other benefits of Scrum include promoting a creative and productive workplace and enhancing business value.

2.2. Software Architecture

Software architecture is one of the points to describe the organization or structure of the system and describes its behavior. A system represents a collection of components that perform a particular function or set of functions. In short, software architecture provides a stable foundation upon which software can be built. Many architectural choices and trade-offs affect system quality, performance, maintainability, and overall success. Failure to consider common issues and long-term consequences can put your system at risk. There are some common architectural patterns and principles commonly used in modern systems. These are often called

architectural styles. Software system architectures are rarely confined to a single architectural style. Instead, the combination of styles often forms a complete system [12].

2.3. Monolith Architecture

Typically, enterprise applications follow the classic three-tier model, consisting of user interface code running on the user's machine (such as HTML pages and JavaScript), server-side business logic responsible for handling HTTP requests, executing domain logic, retrieving and updating data from the database, and selecting and populating HTML views to be sent to the browser, and a database backend. The server-side application is a monolith, which means it is a single logical executable that runs as a single process in the application server's environment. When a new version of the application is deployed, it replaces the previous version in a single step, requiring only the copying of a single executable file to a designated folder.

The monolithic architecture has the advantage of simplicity and ease of testing, deployment, debugging, and monitoring compared to other distributed applications. All data is retained in one database, and all internal communication is done via intra-process mechanisms, which is fast and does not suffer from problems typical to inter-process communication (IPC). It is a natural and first-choice approach for building an application, with all logic for handling requests running in a single process. The development team's preferred language can be used to structure the application into classes, functions, and namespaces.

However, as the application grows and complexity, problems arise. Modifying the application's source becomes more challenging, and changes in one module may lead to unexpected behavior in other modules and a cascade of errors. The size of the monolith results in longer start-up time, slowing down development and becoming an obstacle to continuous deployment. As the application grows, it becomes harder for the development team to maintain a modular structure and ensure that changes related to a specific module only affect that module. Additionally, as the number of developers increases, unequal workforce utilization can lead to productivity losses [5].

2.4. Microservices Architecture

The concept of microservices involves breaking down a business domain into small, self-contained, and independently deployable services, which are consistently bounded and autonomous. Netflix was one of the first companies to adopt microservices, as they began moving away from their monolithic architecture in 2009 before the term had even been coined. The term "microservice" was created by a group of software architects in 2011 and was officially announced a year later at the 33rd Degree Conference in Kraków [6]. However, it wasn't until 2014, when Lewis and Fowler wrote a blog post on the topic and Netflix shared their successful transition, that microservices started to gain popularity. The rise of container technologies such as Kubernetes and Docker have also helped to increase the momentum of this architectural style, especially in cloud-based environments. As a result, microservices have been widely adopted by global companies, including Amazon, eBay, Zalando, Spotify, Uber, Airbnb, LinkedIn, Twitter, Groupon, and Coca-Cola.

2.5. REST API

Representational State Transfer (REST) architectural style for distributed hypermedia systems, describing the software engineering principles guiding REST and the interaction constraints chosen to retain those principles, while contrasting them to the constraints of other architectural styles [7]. The constraints are defined in Roy Fielding's dissertation and include the following:

1. Client-server architecture: The system must be designed to separate the client and the server in order to improve scalability and flexibility.
2. Statelessness: The server must not store any client state between requests. Each request must contain all the information necessary for the server to understand it.
3. Cacheability: Responses must be labeled as cacheable or non-cacheable. Caching improves performance and scalability by allowing clients to reuse responses.
4. Layered system: The architecture must be designed to support a hierarchy of layers, where each layer provides a set of services to the layer above it. This improves scalability and flexibility.
5. Uniform interface: The interface between client and server must be uniform, meaning that it must follow a set of standardized rules for communication. This includes the use of standard HTTP methods (GET, POST, PUT, DELETE) and standard media types (JSON, XML).
6. Code on demand (optional): Servers can provide executable code to clients on demand, allowing for more dynamic and flexible applications.

2.6. Event – Driven Architecture

Event-Driven Architecture (EDA) is an architectural pattern that emphasizes the production, detection, consumption, and reaction to events that occur within a system or environment. In EDA, events are defined as significant occurrences that have an impact on the operation of a system, and the architecture is designed to enable

the system to respond to those events in a timely and efficient manner. The increasing number of connected devices and the rise of the Internet of Things (IoT) have been fueled by new business models and emerging applications. These aim to create systems that can manage large volumes of data generated daily, synchronize devices, and meet business requirements. To address this, EDA can be utilized as an event-driven backbone to process data from multiple applications in real-time. The proposed architecture is cost-effective and supports the Amazon Web Service (AWS) IoT core. Additionally, it can be implemented as free software [13].

2.7. Reactive Programming

Reactive programming is becoming increasingly popular as a paradigm suitable for developing applications that are event-driven and interactive. This programming paradigm provides abstractions for expressing time-varying values and manages dependencies between them automatically, which makes it easier to develop such applications. Several approaches to reactive programming have recently been proposed, including those embedded in different programming languages such as Haskell, Scheme, JavaScript, Java, and .NET. This survey categorizes existing approaches to reactive programming along six dimensions, namely the representation of time-varying values, evaluation models, lifting operations, multidirectional, glitch avoidance, and support for distribution. The survey reveals that there are still open challenges in the field of reactive programming, such as limited support for multidirectional in tracking dependencies between time-varying values in only a few languages. Similarly, glitch avoidance, which is essential in reactive programming, cannot be ensured in distributed reactive programs using current techniques.

2.8. Caching

Caching is the process of temporarily storing data or HTML and images of a website to reduce bandwidth and server loading [10]. In simple terms, a cache is a technology that can help display data faster. Unlike cookies that record user traces and activities when surfing the internet, cache or caching can copy data soon. In order to be used.

2.9. SQL

Structured Query Language (SQL) is a special programming language that is used in data management in the Relational Database Management System (RDBMS). SQL consists of simple syntax in the form of instructions for manipulating data, these instructions are often called queries. The query language is used as standard query language for most Database platform as well as PostgreSQL

2.10. Spring Boot

Spring Boot is very helpful in the development of systems due to its open-source nature, comprehensive documentation, and comprehensive modules such as JDBC, ORM, Servlet, etc. Spring Boot supports the creation of RESTful web service applications, allowing programmers to combine it with other programming languages. It is a very popular choice for building web applications and microservices, due to its ease of use, powerful features, and seamless integration with other Spring modules and third-party libraries [14].

2.11. PostgreSQL

PostgreSQL, which was developed by the Berkeley Computer Science Department, is a relational database management system (RDBMS) that is open-source and free to use. It is capable of processing data in tables that are interrelated with one another, making it suitable for applications that require more complex data processing. Over the last 30 years, PostgreSQL has become a powerful database with stable performance, high security, and a range of features.

Due to its versatility and compatibility with popular programming languages like .NET, C/C++, C#, Java, JavaScript, PHP, Python, etc., PostgreSQL is widely used in web apps, mobile applications, and analytics applications. Using PostgreSQL as the database management system can make developing a web app easier since it eliminates compatibility issues. This is evidenced by the fact that many leading companies, including Apple, Cisco, Instagram, Netflix, Spotify, Uber, and more, use PostgreSQL.

2.12. Apache Kafka

Apache Kafka is a scalable publish-subscribe messaging system with its core architecture as a distributed commit log. It was originally built at LinkedIn as its centralized event pipelining platform for online data integration tasks. Over the past years developing and operating Kafka we extend its log-structured architecture as a replicated logging backbone for much wider application scopes in the distributed environment. Implementation of Apache Kafka usually to design and engineering experience to replicate Kafka logs for various distributed data-driven systems at LinkedIn including source-of-truth data storage and stream processing.

2.13. Redis

Redis is an in-memory database. Memory caching is a technique for storing data that is required by an application in memory, which can be retrieved quickly. Caching data in memory is highly effective when an application repeatedly accesses the data. Redis has a fast, scalable, and flexible design that makes it ideal for use cases such as real-time analytics, messaging, job queue management, caching, session management, and leaderboard scoring. It provides high availability, fault tolerance, and automatic failover through its built-in replication and clustering features. Redis can also be accessed using various programming languages, protocols, and tools, including Python, Java, Node.js, Redis-cli, and Redisson.

2.14. Kubernetes

Kubernetes is a cluster platform for containers or container orchestrator. Kubernetes is expected to be a solution for more efficient computing processes and the creation of systems with high availability. It is a popular open-source container orchestration platform that is widely used for managing containerized applications in production environments.

III. METHODOLOGY

3.1 The Current Condition (As-is)

Pegipegi is one of the largest online travel agents in Indonesia that will help you manage your travel needs, whether it's business, work or just for sightseeing. Pegipegi offers a simple method of selecting, booking and paying for any hotel rooms, in Indonesia or abroad, and airline tickets. Currently Pegipegi is connected to more than 25,000 flight routes, more than 25,000 hotel options, as well as more than 2,800 train routes and airport trains that you can order from the Pegipegi website and mobile application.

The current condition of the Train Search feature in the Pegipegi system can still be used and is still running today (at the time of writing). But seeing the increasing needs and security which is increasingly vulnerable to attacks from irresponsible people. Therefore, the team responsible for the Train domain decided to migrate slowly by moving the features that exist in the current monolith service. One of the existing features in the monolith service is the Train Search feature. Even though there are actually a lot of features in other Train domains that have been migrated before.

Apart from that, like legacy services in general, the existing Train Search feature uses outdated technology and also the technology framework developer team used has not carried out maintenance on that version. If you upgrade the version, it will definitely take quite a long time if you want to keep using the same technology. With these considerations and also the standardization set by the company, it was decided to revoke the Train Search feature in the legacy system to become a new service that will use Spring Boot technology. Apart from that, on the low-level side, the technology currently used still uses a blocking system implementation and will be changed to a non-blocking system with a reactive programming paradigm when migrating to a new service. In addition, on the high-level flow side of the service migration, changes will be made in the process of data retrieval, data storage, data monitoring and other matters that will be improved to better meet the needs of the product development team.

3.2 The Identification of problems

Based on the current conditions (as-is), there are several reasons for the Train Search feature which are the reasons for migrating to the new system, namely:

1. The difficulty of making changes and development to meet the needs of product development on legacy systems.
2. Technology is no longer in accordance with company standards. Even though the legacy system in the train domain still exists, many features have been migrated first using new technology.
3. The current system technology is outdated, after considering several things, namely when upgrading versions with the same system costs more than migrating to a new system.
4. Vulnerable to cyber-attacks, because the technology is outdated, this is consistent with the security of this technology which is no longer being maintained by the developer who developed the technology. Although this can still be overcome by the team that manages the server.
5. The current feature still uses a blocking process and will be changed according to company standards that use a non-blocking process with a reactive programming paradigm.
6. The workflow of the system is not optimal, there are several paths and ways of processing data that are not quite right which results in features that are not suitable for meeting needs within the limits set in contracts with third parties.

3.3 The Expected Condition (To-be)

Based on the problems described above, the engineering team decided that the most ideal solution to do was Migrate Train Search for a new system. Of course, this migration is expected to solve the problems above, namely:

1. Making technology changes using the Spring Boot framework and also using the Spring WebFlux library. These changes make the service easier to manage and also develop in the future according to needs.
2. Better security than before because it was moved to a new system with a new architecture.
3. The service used uses a reactive-programming paradigm in which the use of CPU and RAM resources is smaller when dealing with continuous requests or continuous requests.
4. Optimizing and improving the flow of existing data processing to be presented to clients and also safeguarding the boundaries set by third parties.
5. The last one is the uniformity of other technologies that have been migrated earlier according to the company's current standards (at the time of writing).

3.4 Scope of Application

To solve the problems that exist in Train Search, there are several scopes of work as follows:

1. Configuration Features, this configuration will be used for many things that are useful in configuring services. But for now (at the time of this writing) there is only one feature that is needed according to the needs of the team, namely the feature for on/off service. However, this feature can only be accessed by internal teams, be it engineers, products, SRE or other teams who have the need to access this feature.
2. Get Stations, this will aim to provide a list of existing stations. This feature also provides filters based on departure stations and arrival stations.
3. Get Schedule, this will aim to provide the train schedule you want to search based on the route and also the selected filter.
4. Get Schedule, this detail Aims to provide a detailed schedule of the selected train.
5. Get Seatmap, this will Aims to provide a list of positions from the train seats that have been filled and those that are still empty from the selected train schedule.

Features developed in the Train Search Worker service:

1. Rate Limiter aims so that data search requests do not exceed the existing provisions. At the same time overcome so that users do not search data using bots.
2. Circuit Breaker aims to ensure that when a request to a supplier an error does not become a blocking for other requests and also so that when a supplier does not respond to a given request, other requests can be thwarted.

The difference between Train Search and Train Search Workers is the responsibility of the two services. Train Search is more responsible for communicating to the front-end and managing data. Train Search Worker focuses more on managing requests and responses to suppliers. These two services are linked by using a message broker.

3.5 The Proposed Roles

The author's position is as a Backend Engineer in the Engineering Department, especially in the Land Transportation team, here the author plays a role in developing and managing services and databases for the Pegipegi application, especially for Train and Bus services.

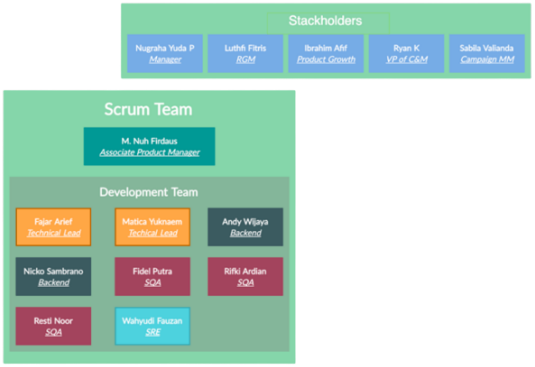


Fig 3.1. Structure in the Project Team

In addition, the author also has an overview of the author's position in the company in terms of the organizational structure below

Fig 3.2. Author's Role in the Company

IV. RESULT AND DISCUSSION

4.1. System Architecture Design

The image below is the result of the architecture implemented in this project. However, due to the limitations of information that can be provided to the public, several parts of the architecture are not depicted.

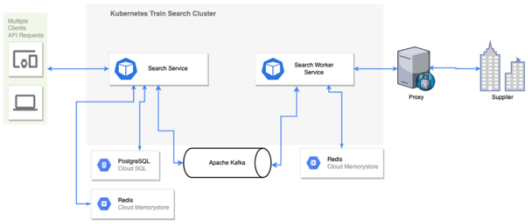


Fig 4.1. High-Level Architecture Design

4.2. Database Design

In the database structure owned by the Train Search service. There are 3 tables namely **stations**, **cities**, and **routes**. The stations table has a foreign-key relationship to the **cities** table through the *city_id* field. While the routes table is used to store a combination of **stations** and **cities** that will create a route between one station and another.

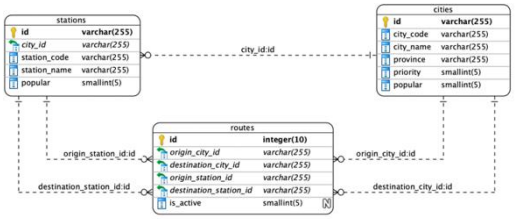


Fig 4.2. Database Design

4.3. Comparison Result

After the development of the Train Search and Train Search Worker services is complete, several comparisons exist in the **legacy** system and also the **new** service on the Train Search feature. Some of these comparisons include comparisons between the legacy system and also the new service, among others, as follows.

4.3.1. High-Level Architecture

The Search feature is no longer part of the **legacy** system where every incoming request to search for train schedules directly leads to the Train Search service.

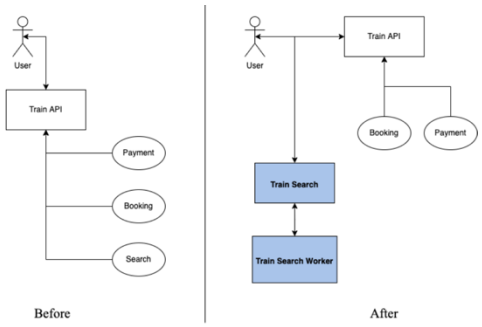


Fig 4.3. High-Level Architecture Comparison

Moving the responsibility of the Search feature from the legacy system, of course, solved another problem, which is the difficulty of the team in developing features according to their needs because the code in the legacy system was very unmanageable and difficult to manage properly. In the legacy system, some files that function as handlers of the Train Search feature have line-of-code containing thousands of lines. When compared to the new service on Train Search and also Train Search Worker. The files that are created are more structured and also have smaller line-code for each file. This makes the future development process easier to organize. If calculated from the line-of-code results, the average line-of-code in the Train Search and Train Search Worker services is 39 to 40 lines. Although it is known that the number of lines in each file is not a guarantee that it will be easier to develop in the future, it can make development easier for developers to read and understand the existing project structure.

4.3.2. Response Time

In migrating, of course, the main goal besides reducing the load on the legacy system is the performance of the response time on requests sent to the server.

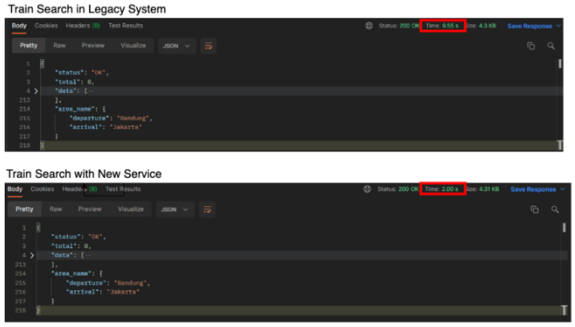
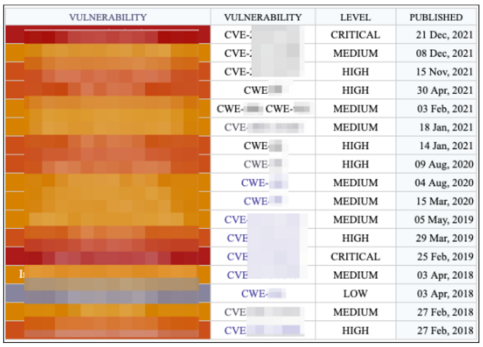


Fig 4.4. Response Time Comparison

Look at the image above, the request to perform the process is quite heavy. Both the legacy service and the Train Search service perform the same process without a caching mechanism. However, there is a considerable difference in response time, the legacy service returns a response in approximately **9** seconds while the new service performs the same process but only takes approximately **2** seconds.

4.3.3. Security

Basically, this vulnerability is found side by side with the version of the framework used on the legacy system. To make improvements, of course, the most correct way is to upgrade the version of the framework used. However, upgrading is very difficult because we must pay attention to the compatibility of the code in the libraries used in the framework.



VULNERABILITY	VULNERABILITY	LEVEL	PUBLISHED
	CVE-	CRITICAL	21 Dec, 2021
	CVE-	MEDIUM	08 Dec, 2021
	CVE-	HIGH	15 Nov, 2021
	CWE	HIGH	30 Apr, 2021
	CWE- CWE-	MEDIUM	03 Feb, 2021
	CVE	MEDIUM	18 Jan, 2021
	CWE	HIGH	14 Jan, 2021
	CWE	HIGH	09 Aug, 2020
	CWE-	MEDIUM	04 Aug, 2020
	CWE	MEDIUM	15 Mar, 2020
	CVE	MEDIUM	05 May, 2019
	CVE	HIGH	29 Mar, 2019
	CVE	CRITICAL	25 Feb, 2019
	CVE	MEDIUM	03 Apr, 2018
	CWE-	LOW	03 Apr, 2018
	CVE	MEDIUM	27 Feb, 2018
	CVE	HIGH	27 Feb, 2018

Fig 4.5. Legacy System Vulnerability

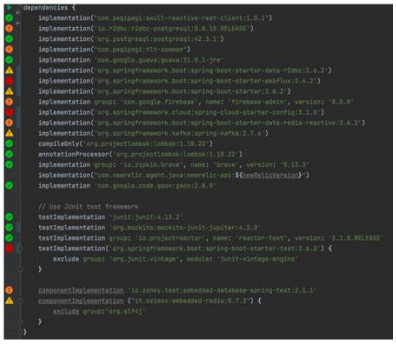


Fig 4.6. Train Search Vulnerability

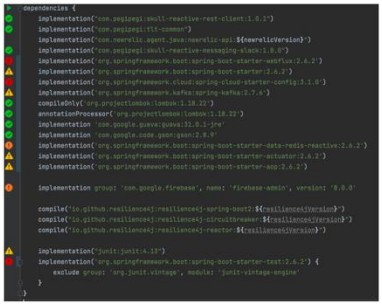


Fig 4.7. Train Search Worker Vulnerability

List of vulnerabilities obtained from the **JFog plugin** in the IntelliJ IDEA application. Seen in service Train Search and Train Search Worker have 12 and 10 vulnerabilities in the library used, respectively. But both have vulnerabilities in the same library. When viewed from the picture above, the vulnerabilities that exist in the Train Search and Train Search Worker services are most commonly sub-libraries that are used in the main library used. To deal with the vulnerability in the existing sub-library, we are waiting for the main library to provide an upgraded version. For vulnerabilities that exist in the Train Search System or the Train Search Worker, they can still be tolerated by the team agreement.

4.3.4 Monitoring

By migrating to the legacy system, it makes the team more flexible in making changes to the flow or workings of this Search feature. As well as making changes in managing monitoring data displayed on the monitoring dashboard.

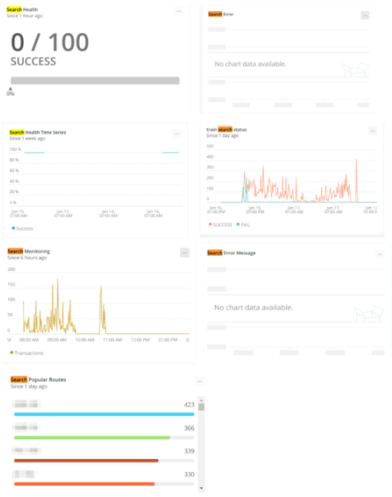


Fig 4.8. Legacy System Monitoring

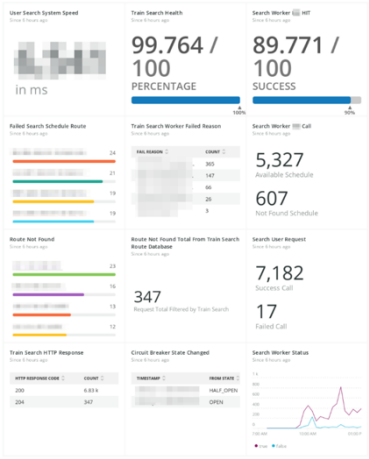


Fig 4.9. New Train Search System Monitoring

Compared to the existing data monitoring in the legacy system, the new Search service has more monitoring variations, which have 7 and 12 monitored data respectively. In the future, the data monitored in the new service will increase as the development progresses according to the needs of the product team.

V. CONCLUSION AND RECOMMENDATIONS

5.1. Conclusion

From the results of the design and development of the Train Search feature migration, several conclusions can be drawn such as:

1. With the development of the Train Search service and Train Search Worker, the burden on the legacy system has been reduced and minimized, as described in the previous chapter. It makes the responsibility of each feature become more independent without having code-coupling with other features. With this migration, the burden on the legacy system, which previously had three features, is reduced to two main features, namely booking and payment. There will be other features that will be migrated after the completion of the Train Search feature migration in the future.
2. This migration brings benefits in terms of easier and more structured development process for the Train Search feature in accordance with existing needs. The ease of development arises from the fact that the code structure becomes better and easier to read compared to the code in the previous legacy system. This is proven by comparing the number of lines of code in the legacy system, where the handler file for the Search feature has hundreds to thousands of lines, while the average number of lines of code in the Train Search service and Train Search Worker files is about 40 lines. Although the number of lines in each file does not guarantee easier development in the future, it helps developers in understanding the project's structure.
3. The implementation of the Event-Driven architecture reduces coupling between the Train Search service and the data supplier. To fulfill this requirement, a connecting service called Train Search Worker is introduced, which is responsible for bridging the Train Search service and the supplier in managing the requests sent to the

supplier. Another advantage of using the Event-Driven approach is that errors or increased latency will not affect the clients (mobile, web) making requests through the Train Search service. Besides serving as a bridge, the main function of the Train Search Worker service is to easily adapt to changes in constraints imposed by the supplier. In general, the Train Search service handles client (user) responsibilities, while the Train Search Worker service handles supplier responsibilities.

4. The development of the Train Search service and Train Search Worker optimizes the management of data obtained from the supplier, making it more suitable for the product and the needs of front-end teams. This also results in faster response times compared to the legacy system. A comparison was made by performing the same process in both the legacy system and the new service, and they both produced the same response, but with different times. The legacy system took 9 seconds to process the request, while the new service only required 2 seconds to complete the same request.
5. This development improves the security of the Train Search feature, which previously had vulnerabilities in the legacy system. With this migration, vulnerabilities with various levels of criticality, high, medium, or low, can be reduced significantly. Although the new service may still have vulnerabilities in the Train Search or Train Search Worker, they can be tolerated according to the agreed terms. It can be concluded that the security of the Search feature in the new service is better than that in the legacy system.
6. Furthermore, this migration allows for more extensive data monitoring, which helps the team identify deficiencies or errors that occur in the running service compared to the monitoring available in the legacy system. Compared to the monitoring data in the legacy system, the new Search service has a wider variety of monitoring, with each having 7 and 12 monitored data points, respectively.

5.2. Recommendations

Based on the conclusions, there are some recommendations to consider for future migration processes:

1. The migrated service can still implement some features from the legacy system to be added to the new service, both in the Train Search service and Train Search Worker.
2. Regularly document the developed service to ensure that the documentation always reflects what is implemented in the service.

Additionally, there are some inputs from one of the experts at Pegipegi. These suggestions are expected to help future migration processes as follows :

1. The migration process took a considerable amount of time because there were other tasks outside of this migration that had higher priorities. This caused other migration tasks to be postponed and other higher-priority tasks to be worked on, even though they were not part of this migration task.
2. Insufficient data collection and the need for additional requirements for the service being developed. This led to the addition of requirements in the middle of the development process.
3. Insufficient communication with the supplier related with new API contract. After the development, it was discovered that there were several new features in the API contract provided by the supplier. These new features can simplify previous development of the Train Search service.

REFERENCES

[1]. Nashrulloh, M. R., Setiawan, R., Heryanto, D., Sutedi, A., & Elsen, R. (2022). Designing Microservices Architecture for Software Product in Startup. Jurnal Teknologi Informasi (JUTIF), 3(1), 45-48.

[2]. Munawar, G., & Hodijah, A. (2018). Analisis Model Arsitektur Microservice Pada Sistem Informasi DPLK. Sinkron, 3.

[3]. Torvekar, N., & Game, P. (2019). Microservices and Its Applications: An Overview. International Journal of Computer Sciences and Engineering, 7(4), 803-809. https://doi.org/10.26438/ijcse/v7i4.803809

[4]. Wonohardjo, E. P., Sunaryo, R. F., Sudiyono, Y., Surantha, N., & Suharjito. (2019). A Systematic Review of SCRUM in Software Development. Journal of Information, Operation Management, and Information Technology, 2(2), 167-192. https://joiv.org/index.php/joiv/article/download/167/192

[5]. Blinowski, G., Ojdowska, A., & Przybyłek, A. (2022). Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access, 10, 20357-20374. https://doi.org/10.1109/ACCESS.2022.3152803.

[6]. Lewis, J., & Fowler, M. (2014). Microservices: A Definition of This New Architectural Term. https://www.Martinfowler.com/articles/microservices.html

[7]. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures [Doctoral Dissertation].

[8]. Bainomugisha, E., Lombide Carreton, A., Van Cutsem, T., Mostinckx, S., & De Meuter, W. (2012). A Survey on Reactive Programming. ACM Computing Surveys, 45, 1-34. DOI: 10.1145/2501654.2501666.

[9]. Khriji, S., Benbelgacem, Y., Cheour, R., El Houssaini, D., & Kanoun, O. (2022). Design and Implementation of a Cloud-based Event-driven Architecture for Real-time Data Processing in Wireless Sensor Networks. The Journal of Supercomputing, 78, 2843-2870. DOI: 10.1007/s11227-021-03955-6.

[10]. Syaefulloh, A., & Yusrizal, F. (2019). Implementasi Dan Analisa Performa DataBase Cache Redis [Implementation and Performance Analysis of Redis Database Cache].

[11]. Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., & Stein, J. (2015). Building a replicated logging system with Apache Kafka. Proceedings of the VLDB Endowment, 8, 1654-1655. https://doi.org/10.14778/2824032.2824063.

[12]. Synopsys. (n.d.). What Is Software Architecture & Software Security Design and How Does It Work? Retrieved April 2, 2023, from https://www.synopsys.com/glossary/what-is-software-architecture.html
[13]. TechTarget. (n.d.). What is event-driven architecture (EDA)? | Definition from TechTarget. App Architecture. Retrieved April 2, 2023, from https://www.techtarget.com/searchapparchitecture/definition/event-driven-architecture
[14]. Spring. (n.d.). EDA Getting Started | Building a RESTful Web Service. Retrieved April 2, 2023, from https://spring.io