International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 05 | May - 2023 SJIF 2023: 8.176 ISSN: 2582-3930

Microservices and Security in E-Commerce

Mr. Sammed Kage
Department of Electronics and Telecommunication SCTR's Pune Institute of Computer Technology Pune, India
sammedkage@gmail.com

Mr. Vedant Karale
Department of Electronics and Telecommunication SCTR's Pune Institute of Computer Technology Pune, India
vedantkarale271@gmail.com

Mr. Kaushal Kawade
Department of Electronics and Telecommunication SCTR's Pune Institute of Computer Technology Pune, India
kawadekaushal183@gmail.com

ABSTRACT

One of the objectives and goals of this project is to understand the various cloud computing paradigms. Other objectives and goals include contrasting traditional monolithic architecture, service-oriented architecture, and microservice architecture; learning Spring and Spring Boot for the implementation of an e-commerce system; implementing microservice architecture using Java and Spring; testing and deploying projects on the cloud using Spring; and communicating our findings.
The study attempts to thoroughly investigate and grasp the different cloud computing models. With an emphasis on understanding their strengths, shortcomings, and applicability for various application situations, it entails a detailed examination and comparison of traditional monolithic design, service-oriented architecture, and microservice architecture. The study goes into the tenets, ideas, and best practises connected to each style of architecture, offering useful information for software development decision-making.
The project also places a strong emphasis on learning the Spring and Spring Boot frameworks.The team will put a lot of work into learning the features and capabilities of Spring and Spring Boot to facilitate effective microservice development and maintenance in the context of an e-commerce system.
A crucial component of the project is the implementation of a microservices architecture utilising Java and Spring.The system will include crucial features like order processing, user account management, and shopping cart functionality, demonstrating the advantages and efficiency of microservices in creating complex applications. Last but not least, a major goal is to effectively communicate the project's results. The team will keep a record of its observations, encounters, and findings throughout the project's duration and share them with others. This will entail presenting the project results, going over the difficulties encountered, and making suggestions based on real-world experience utilising Spring and Spring Boot to develop microservice architecture.

1. INTRODUCTION

What exactly are microservices?
Using the contemporary software development method known as microservices, application code is distributed in manageable, separate, and independent of one another parts.

Why might you design microservices?
They could gain additional advantages from their small size and relative isolation, including simpler maintenance, increased productivity, increased fault tolerance, better business alignment, and more.

Using microservices and Spring Boot You can iterate quickly and scale up your microservices with Spring Boot. Java has become the de facto industry standard for microservices as a result. Use Spring Initializer to quickly launch your project, then package it as a JAR. The integrated server notion of Spring Boot allows you to start working right away.
With the help of its library of ready-to-use patterns, Spring Boot can help with service discovery, load balancing, circuit-breaking, distributed tracing, and monitoring. It also provides a straightforward example of how to install Spring, Spring Boot, and Spring Cloud to create a microservices system.

Microservices enable the construction of large systems from a large number of interconnected components. At the process level, it employs loosely linked processes as opposed to loosely coupled components, exactly as Spring has done at the component level.
Think about a website that offers separate microservices for user accounts, order processing from a product catalogue, and shopping carts.
You obviously need to set up and configure a lot of moving pieces in order to develop such a system. It is unclear how to get them to collaborate; you must be knowledgeable with Spring Boot because Spring Cloud and different Netflix or other open source

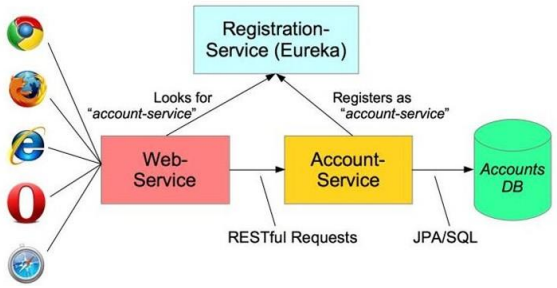

Fig. 1. Microservices Architecture

projects heavily rely on it. Of course, there is also some Spring configuration ”magic”!

A. RESTful webservices Microservices make it possible to build complex systems out of numerous interconnected parts. It uses loosely coupled components rather than loosely coupled processes at the process level, just like Spring did at the component level.Consider a website that provides distinct microservices for managing user accounts, processing orders from a product catalogue, and shopping carts. To create such a system, you clearly need to set up and configure a number of moving parts. It is not clear how to get them to cooperate; you need to be familiar with Spring Boot because it is used widely by Spring Cloud and other Netflix or other open source projects. Of course, there is also some ”magic” in the Spring configuration!

B. Spring cloud and Spring boot Developers can quickly implement some of the common patterns in distributed systems by utilising the tools provided by Spring Cloud (such as configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, one-time tokens, global locks, leadership election, distributed sessions, and cluster state). Using Spring Cloud, which is used to coordinate distributed systems, developers may easily set up services and applications that implement boilerplate patterns. They will function effectively in any distributed context, including managed platforms like Cloud Foundry, bare metal data centres, developer laptops, and others.

C. Java
A high-level, class-based, object-oriented programming language with the least amount of implementation dependencies feasible is called Java. Java is a general-purpose programming language, thus code that has been compiled once can run on any platform that supports Java without needing to be recompiled. The phrase for this is Writing Once, Running Anywhere (WORA). Regardless of the underlying computer architecture, Java programmes are commonly compiled to bytecode that can run on any Java virtual machine (JVM). Java features syntax that is similar to both C and C++, although having less low-level capabilities than those languages. Unlike the majority of traditional compiled languages, the Java runtime includes dynamic features (including reflection and runtime code change). 2019 will see one of the most widely used programming languages.
D. Cloud-based Netflix Eureka service
White Cloud For Spring Boot projects that use autoconfiguration,

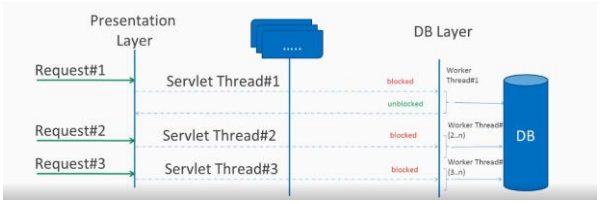

Fig. 2. Blocking-request-processing.png

binding to the Spring Environment, and other Spring programming model idioms, Netflix offers Netflix OSS connectors. You can easily activate and customise the common patterns inside your application with a few short annotations, and you can use Netflix’s tried-and-true components to create enormous distributed systems. Eureka’s Service Discovery pattern is one of the ones offered.

2. PROPOSED METHODOLOGY

1. Different blocks of Microservices –

A. Microservices
A microservices architecture is built on microservices. The phrase describes a technique for breaking down an application into typically small, independent services that can be written in any language and communicate via simple protocols. Software development teams can employ autonomous microservices to implement iterative development processes and dynamically construct and upgrade capabilities.
B. Containers
Software containers maintain a cohesive unit across development, test, and production by packaging services and their dependencies. Microservices and containers are not required for microservice deployment. Comparing containers to other deployment alternatives, such as virtual machines (VMs), however, may speed up deployment and increase programme efficiency.
C. Service mesh
The service mesh establishes a dynamic communications layer in a microservices architecture to streamline communication. Developers do not have to integrate inter-process communication when they construct the software because it abstracts the communication layer.

D. Service discovery
Due to fluctuating workloads, upgrades, or failure mitigation, the number of microservice instances active in a deployment fluctuates. It might be difficult to keep track of several services dispersed throughout the network locations of the application architecture.

E. API gateway
Another essential component of a microservices design is an API gateway. API gateways are crucial for communication in a distributed architecture as they may create the first layer of abstraction between microservices and the external clients. The API gateway will mostly handle the communication and administrative responsibilities that are often performed within a monolithic application, allowing the microservices to remain their lightweight nature.

2. RESTful webservices
REST often refers to a machine-to-machine interaction. REST en-

ables for dynamic content, also known as material that is rendered at the time of request, in web development. RESTful Dynamic content generates a website using server-side rendering and sends it to the user's web browser, which understands the server's code and produces the page. As a set of rules for developing stateless, dependable online APIs, REST has been widely used in the software industry. RESTful is an informal term used to denote a web API that adheres to the REST restrictions. RESTful web APIs often have a weak foundation in HTTP methods like GET and POST. Utilising URL-encoded parameters, HTTP requests are used to retrieve data or resources in the web application. In order to send the data, responses are often structured as either JSON or XML.

3. Load Balancing
Cloud load balancing is a technique for splitting workloads and computing resources in a cloud computing environment. Businesses can control workload needs or application demands by distributing resources among a large number of computers, networks, or servers. One aspect of cloud load balancing is controlling the flow of workload traffic and requests that come in over the Internet. The internet is expanding so quickly that traffic is growing by almost 100 percent annually. As a result of the steadily rising traffic on servers, servers are being overwhelmed, particularly popular web servers. There are two straightforward fixes for the overloaded server problem.An initial single-server strategy entails updating the server to a more performant model.

4. Distributed Tracing
Distributed tracing is the technique of following application requests as they pass from frontend devices to backend services and databases. Developers can utilise distributed tracing to examine requests that have a lot of delay or errors. In this post, we'll discuss distributed tracing's functionality, advantages, and initial tools.

5. Hysterix
Hystrix is a library that controls inter-microservice communication to provide fault tolerance and latency. A modification in the user interface (UI) might also make sense to let the user know that something might not have gone as expected or might take longer.

AES Algorithm

A popular symmetric encryption technique that offers high protection for sensitive data is called the Advanced Encryption Standard (AES). It uses fixed-size data blocks that are generally 128 bits in size and supports keys that are 128, 192, or 256 bits in size.

Here is a high-level explanation of the AES algorithm's operation:

1. Key Expansion: A series of round keys are created by expanding the original encryption key. Each round key is created by deriving it from the one before it and then going through several changes.

2. Initial Round: Blocks of the input data, sometimes referred to as plaintext, are created. Each block is merged with the first round key in the initial round using a bitwise XOR operation.

3.Rounds: The number of rounds in AES is fixed and dependent on the key size. For a 128-bit key, there are 10 rounds; for a 192-bit key, there are 12 rounds; and for a 256-bit key, there are 14 rounds. The following actions are taken in each round:

a. Subbytes: Each byte in the block is substituted using the S-box, a preset substitution box. The non-linear replacement makes the data more ambiguous.
b. ShiftRows: Each row of the block's bytes undergoes a cyclical leftward shift. Diffusion in the data is provided in this stage.
c. MixColumns: Using a mathematical process known as matrix multiplication over a finite field, each column of the block is changed. The data is further muddled by this procedure.
d. AddRoundKey: A bitwise XOR operation is used to combine the block with the current round key.

4. Final Round: There is no MixColumns step in the final round, but it is otherwise comparable to the rounds previously discussed.

5. Output: Following the last round, the created ciphertext serves as the plaintext's encrypted counterpart.

The reverse procedures are used with the same round keys in reverse order to decode the ciphertext. The following steps are involved in the decryption process:

1. Key Expansion: As previously, round keys are created.

2. Initial Round: A bitwise XOR technique is used to combine the ciphertext with the previous round key.

3. Rounds: The following actions are taken throughout the decryption rounds:

a. InvShiftRows: Each row of the block's bytes undergoes a cyclical right shift.
b. InvSubBytes: The inverse S-box is used to substitute each byte in the block.
c. AddRoundKey: Using a bitwise XOR operation, the block is joined with the current round key.
d. InvMixColumns: Similar to the MixColumns step in encryption, each column of the block is altered using a matrix multiplication over a finite field.

4. Final Round: The InvMixColumns step is not included in the final round.

5. Output: After the last round, the resultant plaintext—which is the ciphertext that has been decrypted—is received.

AES is frequently used in many applications that demand data security since it is universally considered as a safe and effective encryption technique.

3. SECURITY CONSIDERATIONS

When it comes to the deployment and use of microservices in the e-commerce space, security concerns are of the utmost significance. It is essential to preserve data, user privacy, and the system's general integrity in today's linked world where sensitive user information and financial transactions are conducted online.
Another crucial component of security in a microservices context for e-commerce is data encryption. It entails protecting data from unauthorised access and modification by encrypting it both at rest and while in transit. Sensitive information including client payment information, personal information, and authentication credentials can be protected using encryption methods like the AES (Advanced Encryption Standard).

For assuring the security and integrity of data shared between microservices and external systems, secure communication protocols, such as HTTPS, are crucial. The danger of eavesdropping, manipulation, and man-in-the-middle attacks may be considerably reduced by using industry-standard encryption techniques.
To prevent unauthorised access to microservices and the resources that go with them, access control measures are essential. To impose fine-grained permissions and guarantee that only authorised entities may carry out particular actions, access control models like as role-based access control (RBAC), attribute-based access control (ABAC), and others can be put into place.
For the system to be secure, it is essential to conduct regular security audits, vulnerability assessments, and penetration tests. Organisations can keep ahead of new threats and put the right security measures in place to thwart assaults by undertaking proactive assessments.
In conclusion, it is crucial to address security issues while using microservices in the e-commerce space. Organisations can improve the overall security posture of their e-commerce systems and inspire trust in their users by establishing strong authentication and authorization mechanisms, implementing data encryption, utilising secure communication protocols, enforcing access control, and performing regular security assessments.

4. CHALLENGES

Challenges while creating micro services

For developers, building microservices architecture might provide a variety of difficulties. The following are some of the major difficulties that programmers may encounter when building microservices:

Complexity:
The classic monolithic design is simpler than the microservices approach. Multiple services with various features must be designed and developed by developers, which can make maintenance and management difficult.

Communication:
It might be difficult for microservices to communicate with one another, especially in a dispersed context. Developers must make sure that one service can interact with others in a smooth manner while maintaining the integrity and consistency of the data.

Testing:
It might be more difficult to test a microservices architecture than a standard monolithic programme. Developers are required to thoroughly test every service and guarantee that they all function in unison.

Deployment:
Microservices architecture deployment can be challenging, especially in a distributed setting. Multiple services must be deployed and managed separately by developers, which raises the possibility of mistakes and inconsistencies.

Data management:
Data management across many providers might be difficult. When managing data storage and retrieval, developers must make sure that data is consistent across all services and that data integrity is upheld.

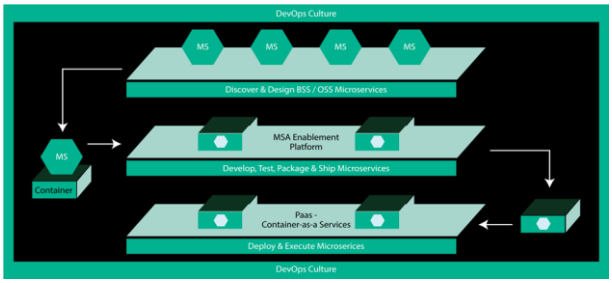

Fig. 3. DevOps Culture

Security:

A major issue with microservices design is security. Developers are responsible for ensuring that data is safeguarded, that communication between services is secure, and that each service is secure.

5. RELEVANCE

The gradual and iterative process of implementing microservices architecture for telephony must be approached with a long-term strategy. To properly use the MSA paradigm, CSPs must first establish the framework, then locate, design, and develop relevant microservices use cases.

Changing the culture
The main problem that DevOps, automated delivery, and MSA are expected to collectively address is welcome change requests. To accept change rather than to fight it requires a cultural shift. The DevOps culture is supportive of ongoing change, including continuous development, integration, and delivery. The emphasis on ownership and cooperation in DevOps benefits operators, designers, developers, and testers alike.

Prefer the PaaS
The second important problem that a microservices architecture design aims to solve is volatile volume. Dynamic scalability need an elastic hosting infrastructure and platform, which may be provided through this. In comparison to hardware virtualized infrastructure, OS virtualized container technologies provide more flexibility with substantially reduced virtualization overheads.

Discover the microservices As soon as they have successfully institutionalised the DevOps culture and come to an agreement to shift capabilities from static infrastructure to dynamic PaaS, the CSPs should be ready to scan their application capabilities landscape and find prospects for microservices applications. The most well-known domain-driven design method, which employs noun (thing) based and verb (action) based deconstruction, may be used to find and create microservices.

6. LITERATURE SURVEY

Microservices, a recent architectural trend that draws influence from service-oriented computing, are a result of this. Before describing the present state of the art in the field, this chapter studies the development of software architecture, including the circumstances that led to the first distribution of objects and services and the later expansion of microservices. The remaining problems are then discussed, along with impending challenges.

This survey primarily addresses newcomers to the discipline while offering an academic viewpoint on the matter. We also investigate a few real-world issues and provide some workable solutions.

A large, monolithic programme makes development more difficult and time-consuming; individual components cannot be scaled; any module issue has the potential to affect the availability of the entire application; a change in the framework or language will have an impact on the entire programme, making updates expensive and time-consuming; and finally, any module issue has the potential to affect the availability of the entire application.

New instances of a microservice may be immediately deployed to the related cluster to assist alleviate pressure if it meets its load capacity. Exploring and adopting microservices designs enable small teams that deploy often to embrace agile methods of working. Now that we are multi-tenant and stateless, our clients are spread out among several instances. Now that we can handle much larger instance sizes, our release cycles are quicker and more frequent. The frequency of our updates has increased from once per week to as much as two or three times daily. Teams are now free to experiment with new features and return to older ones if they don't work.This expedites the selling of new features and makes changing code less difficult. Additionally, because microservices are independent components, they make it straightforward and rapid to independently deploy certain functionality. They also make it simple to identify and correct flaws and issues in particular services.

7. APPLICATIONS

A series of tiny, independent services that work together to deliver an application's overall functionality is known as a "microservice" in terms of architecture. This strategy has become more well-liked in the e-commerce sector as a result of its capacity to improve the application's agility, scalability, and resilience.
Here are some examples of how security and microservices are used in e-commerce:

1. Scalability:
E-commerce apps can scale independently thanks to microservices design, which enables various services to be scaled up or down depending on demand. For instance, the services managing order processing, payment processing, and delivery may be scaled up to manage the additional load during the Christmas season, when there is a rise in online shopping.

2. Agility:
E-commerce apps may be designed and deployed more quickly thanks to microservices architecture. Since each service is created and tested separately, it may be deployed as soon as it is prepared without needing to wait for the full application to be finished.

3. Resilience:
E-commerce apps can be more fault-tolerant thanks to the microservices design. The failure of one service does not impact the operation of the entire application since each service is autonomous. The remaining services can carry running, ensuring that the entire application continues to run.

4. Security:
A more secure e-commerce application is possible using microservices architecture. Due to the independence of each service, security flaws are simpler to identify and contain. Additionally, if necessary, each service may be secured using a separate security protocol independently.

5. Faster Updates:
E-commerce apps may be updated more quickly thanks to the microservices design. It is possible to update a service as soon as it is ready without having to wait for the entire application to be updated since each service is built and tested individually.

Overall, the use of microservices and security in e-commerce may benefit firms by enhancing customer service, boosting agility, and lowering the chance of security breaches.

8. PROGRESS OF PROJECT AND DISCUSSION

Since its beginnings, the project on microservices and security in e-commerce has advanced considerably. Our team has been actively working on putting together a microservice architecture using Java and Spring, with a focus on making sure that the e-commerce system has strong security safeguards. We have successfully planned and created a number of important microservices that make up the e-commerce platform's framework in terms of microservice development. User management, inventory management, order processing, payment handling, and product catalogue are some of these microservices. Microservices provide modular development, scalability, and independent deployment because each is in charge of a particular functionality. We gathered the essential data about conventional approaches to application development architecture, including advantages and disadvantages. To address each of these issues, we did an online study of the literature utilising many publications from IEEE Conferences. We looked at different designs that may be utilised to construct numerous services under microservices utilising Spring Boot after evaluating the algorithms employed in the current systems.

In order to protect sensitive data, including user passwords, payment information, and personal information, we have also deployed strong data encryption techniques. In order to guard against unauthorised access and data breaches, we make use of encryption technologies and secure communication protocols. Along with the technical implementations, we tested the system thoroughly for security during the whole development period. For the purpose of identifying and addressing potential security flaws, this includes vulnerability assessments, penetration testing, and code reviews. To create a safe and dependable e-commerce system, we aggressively address security risks at every level of development. We are constantly evaluating and improving the security of the microservices architecture as the project moves forward. This include keeping up with the most recent security best practises, patching any vulnerabilities found, and doing routine security audits. We are dedicated to providing our consumers' sensitive data with the utmost security for our e-commerce system.
(i) Review and analysis of the problem statement
(ii) Gathering of research papers
(iii) Survey of the literature
(iv) Choosing the appropriate implementation strategies for the project
(v) Regular communication with the project's internal guide
(vi) Estimating new methods and techniques

9. FUTURE PLAN

Real benefits of using microservices include improved scalability, adaptability, and agility. Using microservices, each component can be scaled independently. The entire procedure is consequently quicker and less expensive than monoliths, where the entire programme must be scaled even when it is not necessary. Each monolith has scalability limitations as well, therefore the more users you add, the more problems your monolith encounters. Many companies are forced to rebuild their monolithic structures as a result.

First off, because each service may be independently installed and improved, there is more flexibility. Second, rather of affecting the entire programme, a bug in one microservice simply impacts that particular microservice. Additionally, microservice applications allow adding new functionality much easier than monolithic ones do. In conclusion, this project may be revised to take into account a variety of other E-commerce-related services or services as needed over time or in response to customer input.

For a more successful gradual migration approach away from large, monolithic systems. Most modern object-relational mapping and web application frameworks give the ability to use databases concurrently by several services without interfering with business operations. As a result, employing many databases to extend the implementation of a project like this might lead to more productive behaviour for the project.

The easiest method to avoid the problems of a database-per-service model is to provide a single database from which various services may pull the appropriate resources. For instance, using a common database makes it simpler to integrate several set data structures. As long as all required tables are present in a single database, distributed transactions may be reliably executed through the use of atomic guarantees and database primitives.

10. CONCLUSION

The project on e-commerce microservices and security has made substantial progress towards developing a solid microservice architecture while giving strong security measures a priority. We have successfully designed, implemented, and integrated important microservices that are in charge of a number of e-commerce platform operations, such as user management, inventory management, order processing, payment handling, and product catalogue, during the course of the project. To find and fix any security flaws, extensive security testing has been carried out, including vulnerability assessments, penetration testing, and code reviews. We have made an effort to create a safe and robust e-commerce system by proactively addressing security risks at each level of development.

By allowing for modular development, scalability, and independent service deployment, the project has shown the advantages of a microservice architecture for e-commerce. This design encourages flexibility and enables effective system development and maintenance. Additionally, by including robust security measures, we sought to give users a safe and secure buying experience, fostering confidence in the e-commerce platform.

The integrity of the microservices architecture and the protection of user data will require ongoing monitoring, improvement, and adherence to security best practises. To address new threats and provide continued protection against potential vulnerabilities, regular security assessments and upgrades will be carried out.

In conclusion, the project on microservices and security in e-commerce has advanced significantly in terms of putting a microservice architecture into place and making sure that there are reliable security precautions. A scalable, versatile, and secure e-commerce platform is built on the effective integration of microservices and sound security procedures. This project makes a significant contribution to the field of microservices and e-commerce security by solving the issues related to the development and security of e-commerce systems.

11. REFERENCES

Books: Microservices Patterns: With Examples in Java

Hai Dinh-Tuan, Maria Mora-Martinez, Felix Beierle, Sandro Rodriguez Garzon “Development Frameworks for Microservice-based Applications: Evaluation and Comparison”

[2] Kapil Bakshi, “Microservices-Based Software Architecture and Approaches”, Cisco Systems, Inc., 13635 Dulles Technology Drive Herndon, VA 20171 703 484 2057 kabakshi@cisco.com

[3] Hatma Suryotrisongko*, Dedy Puji Jayanto, Aris Tjahyanto Institut Teknologi Sepuluh Nopember, Kampus ITS Sukolilo, 4th Information Systems International Conference 2017, ISICO 2017, 6-8 November 2017, Bali, Indonesia “Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot” -Surabaya 60111, Indonesia

[4] Michael Simons, jax LONDON presents “SPRING BOOT”

[5] Arne Koschel Irina Astrova Jeremias Dötterl, International Conference on Information Society (i-Society 2017), “Making the Move to Microservice Architecture”, Faculty IV Department of Computer Science Department of Software Science School of IT Faculty IV Department of Computer Science University of Applied Sciences and Arts Tallinn University of Technology University of Applied Sciences and Arts Hannover, Germany Tallinn, Estonia Hannover, Germany akoschel@acm.org irina@cs.ioc.ee

[6] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, Stefan Tilkov, “Microservices: The Journey So Far and Challenges Ahead”, IEEE Software (Volume: 35, Issue: 3, May/June 2018)

[7] Baiqiang Gan, Chi Zhan, “Research on Application of Distributed Server Architecture for Virtual Reality Scenarios in Big Data Environment”, Date of Conference: 17-19 April 2020, Date Added to IEEE Xplore: 27 July 2020, INSPEC Accession Number: 19854151, DOI: 10.1109/CIBDA50819.2020.00020, Publisher: IEEE, Conference Location: Guiyang, China

Website: Spring Boot Reference Documentation : https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/