International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 07 | July 2022 www.irjet.net p-ISSN: 2395-0072

Developing microservices with Java and applying Spring security framework and OAuth2

Sneha Suresh Vanjire[1], Dr. Rajashekara Murthy S[2]

Student, Dept. of Information Science and Engineering, RV College of Engineering, Bangalore, Karnataka, India
Associate Professor, Dept. of Information Science and Engineering, RV College of Engineering, Bangalore, Karnataka, India

---

Abstract - *Today's development is mostly done using a microservice architecture. As part of the microservices architecture, applications are built from a number of small modular services. In 2014, Google, Netflix and Twitter implemented Microservice Architecture (MSA), which runs as separate processes and communicates with each other in different ways. Since then, these companies have widely decoupled and implemented Microservice Architecture. With this design, an application's services are broken down, distributed independently, and then run. In this study, we look at how Java microservices can be developed, as well as how OAuth2 and the Spring Security Framework can be used to secure microservice APIs that are built on top of the Spring Framework.*

Key Words: **SpringBoot, Microservice Architecture (MSA), Software Architecture, OAuth2, Proof of Concept (POC), and Spring framework.**

1. INTRODUCTION

Although RESTful endpoints with a single functionality are how microservices are frequently characterized, there are numerous different ways for developers to build these services. The microservice architectural style was created as a result of the architectural design of Service-Oriented Architecture and Domain-Driven Design, with a significant focus on DevOps techniques. It has attracted a lot of interest from academics as a trustworthy and scalable solution to construct cloud services, and it has also been widely adopted by companies with millions of clients, like Netflix or Amazon. In a nutshell, the microservice architecture encourages the division of the programme into services as an unique software architecture. According to the conventional monolithic method of software architecture, each deployment requires the bundle of the full application stack. This idea has numerous disadvantages for the application, including rigid scalability, significant resource costs and refactoring effort, and challenges with DevOps amongst scattered teams. By creating a Proof of Concept (POC) of an MSA application utilizing the Spring Framework, Spring Security, and OAuth2, and performing security testing over the POC, this research aims to close the knowledge gap on MSA and API security.

Different Ways of Creating Microservices

- RESTful endpoint-based applications operating independently and acting as services for a certain system characteristic.
- Creation of headless services like AWS Lambda, also referred to as Function as a Service (FaaS).
- Services that use messaging or events to communicate, such as clustered Vert.x vertices (the Java Reactive framework), are known as messaging or event-based services.

1.1 Various Microservices Libraries in Java

It can be difficult to think about the technology roadmap while switching from such a monolithic to something like a microservices architecture because there are so many concepts, problems, and technological options. People could therefore overthink the issue and overdevelop the answer when developing a new application under a microservice architecture. The most complete coverage for microservices libraries is provided by the Spring framework, which is part of the Java ecosystem.

A variety of libraries are listed with use cases in Table - 1.

Table -1: Libraries with Use Cases

Library	Use cases	Tools Illustration
Discovering and Registering Services	Manage configurations in a distributed and safe manner	Spring Cloud Configuration, Consultation, and Vault
Implementation Management	Using registered service names to locate service nodes	Spring Cloud - Netflix Eureka, Consul

Monitoring for Systems, JVMs, and APIs	Performance degradation, risk and crash analysis	New Relic, Spring Boot Admin and Datadog
API Gateway	serving as a point of entry and managing issues with security, URL name, etc.	Zuul, Nginx, cloud-provided API gateways
Security	Secure socket layer authenticity using a password and a digest, OAUTH, and JWT, and social login integrations	Spring Security for Spring

1.2 Microservices with Java Frameworks

A new breed of Java frameworks allows you to swiftly package an entire web service with the embedded container of your choice in an auto running JAR file. It was revolutionary to switch from a big, hefty J2EE container to a smaller, lighter version. There will be a few frameworks accessible, such as Spark, Dropwizard, and Spring Boot. We will investigate a SpringBoot sample. The Spring framework has been used to build Java applications for more than ten years and is the de facto industry standard.

2. SPRINGBOOT

One important factor in Java's success in enterprise development is the Spring Framework. Almost all of the widely used libraries and frameworks for Java web development are supported by this framework. We can swiftly develop self-running microservices with Spring Boot in a matter of minutes. In the past, developing boilerplate code for an application's sole purpose of wiring Spring infrastructure components required a significant amount of development effort. The majority of the routine stuff has already been set up for us.

Listed here are some of Spring Boot's highlights:

- Things may be rapidly set up and going.
- Overriding configuration, libraries, and frameworks is incredibly flexible.
- There is a very robust community that can offer answers to many potential issues.

Examine the base class for our usage cases and the Hello World application example. This class will serve as the foundation for our sample web application. If you want to change or modify anything, you can use a property file, a Metadata file, or Java-based configuration.

Define the Application class first. The Spring Boot application's starting point and RESTful endpoint are both provided by this short single class.

Class of Application shown in Fig -1 below.

```
package com.example.hello;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@SpringBootApplication
@RestController
public class HelloWorldApplication implements CommandLineRunner {
   public static void main(String[] args) {
   SpringApplication.run(HelloWorldApplication.class, args);
   }
   @Override
   public void run (String... args) throws Exception {
   log.info("Started the App which is running now");
   }
   @GetMapping ("/HelloWorld")
   public String HelloWorld() {
   return "Hello World!!!!";
   }
}
```

Fig -1: Application class

An order to eliminate the need for manual project setup in the beginning, the Spring Framework community offers an additional facility. You can get a rapid, ready-to-launch initial configuration with a list of all the Spring dependencies on the http://www.start.spring.io website.

By default, Spring Boot loads the settings from class path files called application. properties or application. yml. In this manner, Java will finally be used to develop the microservice.

3. SPRING SECURITY FRAMEWORK AND OAUTH2

So we have created the microservices necessary to apply security inorder to make secure applications.

Therefore, in order to build safe apps, we have designed the microservices that are required to apply security. Applications exchange information with one another across the Internet and network communication protocols, hence this architectural design strongly relies on APIs (API). A microservice application's Api must be appropriately secured as a result in order to protect the application as well

as its resources against risks associated with API invocations.

3.1 Microservice API and Microservice Architecture:

Microservice architecture (MSA) is the term for the application architecture that divides an application into a number of narrowly focused, single responsibilities, portable, and independently evolving services. The typical monolithic software architecture, in contrast, deploys the complete application with all of its services inside just one application server. Because REST is a lightweight protocol, this research employs it for the API development and experiment. MSA simply divides the application logic into numerous smaller components; it does not make a programme any simpler. Scalability and high availability are two benefits of decomposition, but it also results in a significantly highly complex network linkage between components, especially whenever the application is made up of an excess amount of services. A comparison of the monolithic and microservice architectures is shown in Fig -2 and Fig -3.

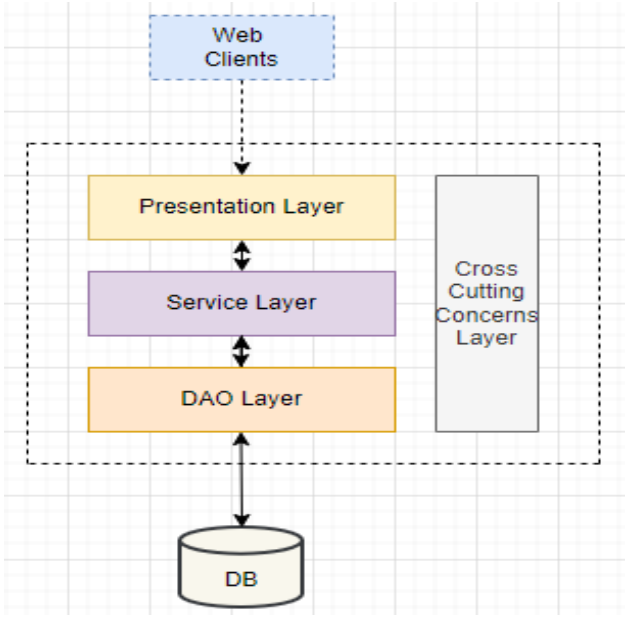

Fig –2: Monolithic Architecture

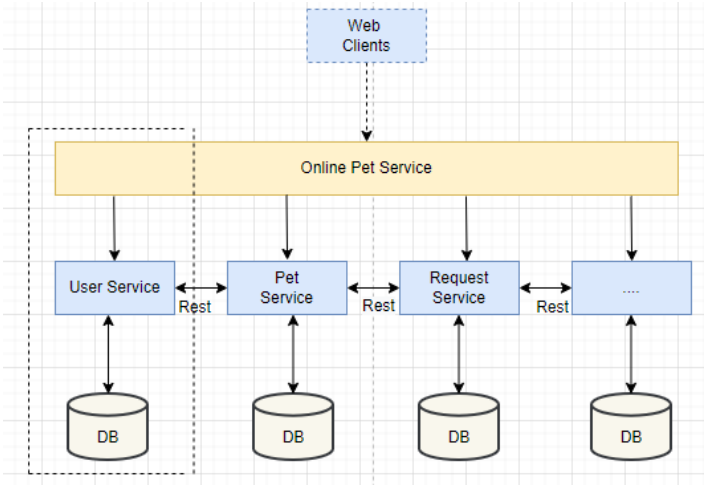

Fig -3: Microservice Architecture

4. PROOF-OF-CONCEPT CREATION

The POC is created in order to respond to the inquiry. Additionally, it must to demonstrate how MSA can be used in actual business situations. In order to reply to the enquiry, the POC is established. Additionally, it must show how MSA can be applied in real-world business scenarios. This makes a POC for such just an inventory management software an appropriate experimental application. OAuth2 is used for backend services that don't need a web application browser and user interaction techniques in additional to the online application, as seen in Fig -4. The POC must be constructed in a specific fashion in order to perform security checks for both of these technology types. Additionally, OAuth 2.0 should be investigated in the experiment for both authentication and authorization needs. Key characteristics and actors are suggested in the following subsections in accordance with the experimental needs.

4.1 Use cases

Use scenarios for an authorization server include:

The Identity Provider in the OAuth2 process is the Authorization Server. The design provides the fundamental use scenarios for the Authorization Server, as shown in Figure 4. The following two actors communicate with the Authorization Server:

- Owner of the resource in the OAuth2 workflow.
- The client app is the client application which has identified with the authorisation server.

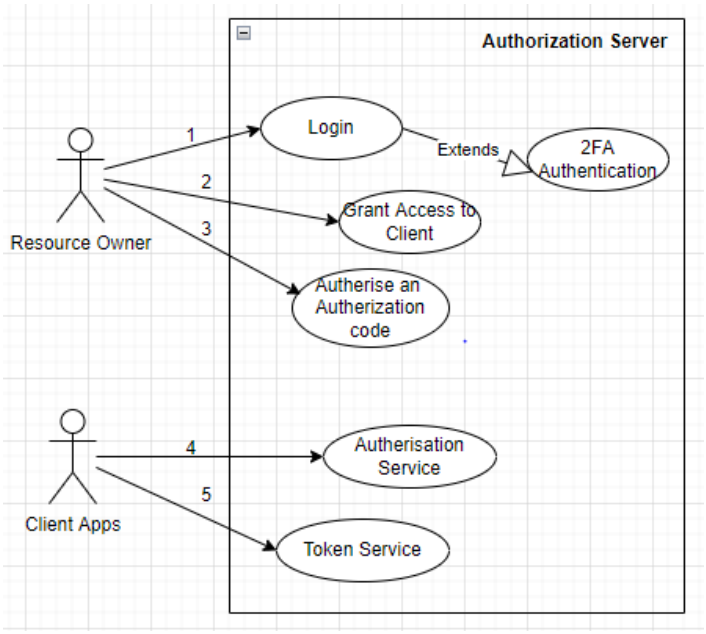

Fig -4: Authorization Server

Use scenarios for an resource server include:

There are two microservices put into place by the POC to manage the watch and mobile phone, respectively. The microservices replace the RPs in the OAuth pipeline. Figure 5 displays the use cases for the resource server.

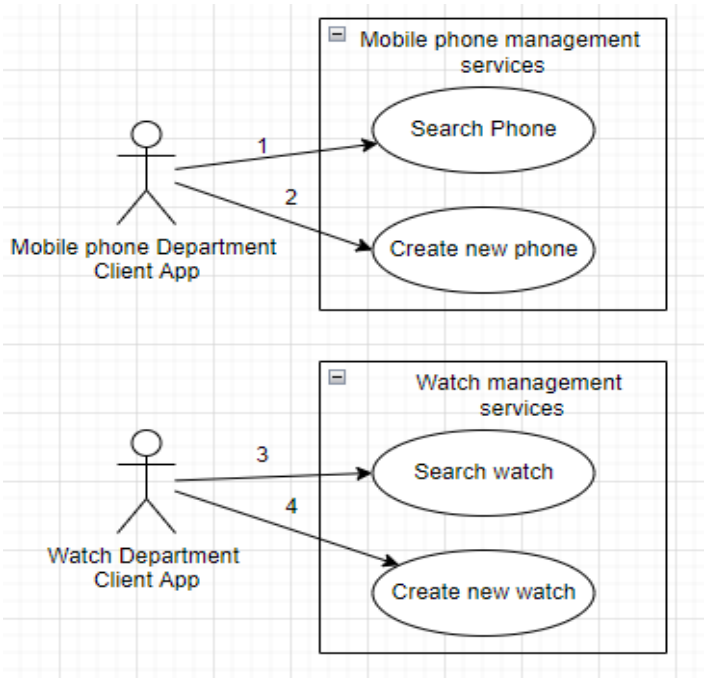

Fig -5: Authorization Server

4.2 OAuth2 Authorisation Configuration

The built-in Authorization Server Configure Adapter of the Spring Security Framework is improved by OAuth configuration and now includes an implementation for OAuth2 authorisation support. It offers the following key features:

- **OAuthDataSource**: searches the database for client information during the authorization procedure.
- **TokenStore**: The Java Database Connectivity (JDBC) approach is used to access the access tokens kept in the database as well.
- **ApprovalStore**: Using JDBC technology, users may access approval data that is kept in a database.
- **AuthorizationCodeServices**: Authorization codes are saved in a database, just like in the ApprovalStore.
- Configure the ClientDetailsService by defining each unique client and their properties with configure(ClientDetailsServiceConfigurer clients).
- void Use the configure(AuthorizationServerEndpointsConfigurer endpoints) command to set up the Authorization Server access points, including a token storage, authenticating code service, token customizations, user approvals, and grant types.

5. CONCLUSIONS

Just now, we looked at the standard starting point. Due to the fact that it offers the most comprehensive solutions for the diverse needs of any corporate system, the most of the examples we examine from here on out will utilise the Spring Framework. Using Spring Boot, we may split a larger microservice into smaller ones. Microservices interact within the MSA application with one another using service API endpoints. An API endpoint is a location in which the services can connect and receive the resources they need to perform their duties. The API endpoint, which serves as the interface through which data is transferred between services, is crucial in ensuring the proper operation of the systems and services that interact with it. As a result, API endpoint security is among the most crucial security components in an MSA applications. The researcher hopes to expand on this work in the future to include the security of all API implementations, as well as the security of additional application layers like the business layer and the data access layer. As a result, suggest an API security solution that is more complete for the Java-based microservice application.

REFERENCES

[1] Y. Gong, F. Gu, K. Chen and F. Wang, "The Architecture of Micro-services and the Separation of Frond-end and Back-end Applied in a Campus Information System," 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA), 2020, pp. 321-324

[2] R. Pereira, P. Simão, J. Cunha and J. Saraiva, "jStanley: Placing a Green Thumb on Java Collections," 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2018, pp. 856-859

[3] Y. Gan and C. Delimitrou, "The Architectural Implications of Cloud Microservices," IEEE Computer Architecture Letters, vol. 17, no. 2, July-Dec. 2018, pp. 155-158

[4] Hatma Suryotrisongko, Dedy Puji Jayanto, Aris Tjahyanto, "Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot", Procedia Computer Science, vol. 124, 2017, pp. 736-743

[5] Y. Jayawardana, R. Fernando, G. Jayawardena, D. Weerasooriya and I. Perera, "A Full Stack Microservices Framework with Business Modelling," 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer), 2018, pp.

[6] Hatma Suryotrisongko, Dedy Puji Jayanto, Aris Tjahyanto, "Design and Development of Backend Application for Public Complaint Systems Using Microservice Spring Boot", 4th Information Systems International Conference 2017, ISICO 2017, 6-8 November 2017, Bali, Indonesia