Developing SOAP Web Services with Spring Boot: A Comprehensive Guide

Extio Technology · Follow
6 min read · Jul 7, 2023

Listen Share

Extio Java Series — SOAP Web Services

Introduction

In today's interconnected world, web services play a crucial role in facilitating communication between different applications. SOAP (Simple Object Access Protocol) is a widely adopted protocol for creating web services. When it comes to building SOAP web services in Java, Spring Boot provides an efficient and developer-friendly framework. In this blog post, we will explore the fundamentals of

developing SOAP web services using Spring Boot and discuss the steps involved in building a robust SOAP-based application.

Table of Contents:

9. **Understanding SOAP Web Services:**
   SOAP is a protocol that defines a standardized format for exchanging structured information between networked applications. It relies on XML for message formatting and is based on a client-server model. SOAP web services allow different systems to communicate with each other over a network, regardless of the underlying technology stack.

10. **Setting up a Spring Boot Project:**
    To get started with developing SOAP web services, we need to set up a Spring Boot project. This can be achieved by using either Spring Initializr or your preferred integrated development environment (IDE) that supports Spring Boot project creation. We'll also need to add the necessary dependencies for SOAP web service development, such as Spring Web Services.

11. **Defining the Service Contract:**
    In SOAP web services, the service contract is defined using Web Service Definition Language (WSDL). The WSDL describes the operations, data types, and message formats used by the web service. Spring Web Services provides various ways to define the service contract, including WSDL-first and code-first approaches. We'll explore the code-first approach in this blog post.

12. **Implementing the Endpoint:**
    Once the service contract is defined, we can start implementing the SOAP endpoint. The endpoint acts as the entry point for incoming SOAP requests and maps them to the appropriate business logic. Spring Web Services provides annotations, such as `@Endpoint` and `@PayloadRoot`, to simplify the implementation of the SOAP endpoint.

13. **Testing the SOAP Web Service:**
    To ensure the correctness of our SOAP web service, thorough testing is essential. Spring Boot provides excellent support for testing SOAP web services using frameworks like JUnit and Spring Web Services Test. We can write integration tests to verify the behavior of our SOAP endpoint and ensure that it handles requests and responses correctly.

14. **Handling Errors and Faults:**
    In SOAP web services, errors and faults need to be handled appropriately. Spring Web Services offers several mechanisms for handling exceptions and generating SOAP faults. By using exception handling techniques, we can customize the fault messages returned to clients and provide meaningful error information.

15. **Securing the SOAP Web Service:**
    Security is a crucial aspect of any web service, and SOAP web services are no exception. Spring Boot provides various options for securing SOAP web services, such as SSL/TLS encryption, message-level security, and authentication mechanisms like Basic Authentication or WS-Security. We can choose the appropriate security measures based on the specific requirements of our application.

16. **Deploying the SOAP Web Service:**
    Once our SOAP web service is developed and tested, the next step is deployment. Spring Boot simplifies the deployment process by providing several deployment options. We can package our application as an executable JAR file or deploy it to a servlet container or application server. Dockerizing the application is another option for containerization and easy deployment.

Example

Here's an example of a simple Spring Boot SOAP web service implementation:

1. **Define the Service Contract (WSDL):**

   Create a file named `hello.wsdl` with the following content:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             targetNamespace="http://example.com/soap-web-service"
             xmlns:tns="http://example.com/soap-web-service">
 <message name="GetHelloRequest">
        <part name="name" type="xsd:string"/>
    </message>
    <message name="GetHelloResponse">
        <part name="greeting" type="xsd:string"/>
    </message>
    <portType name="HelloPortType">
        <operation name="getHello">
            <input message="tns:GetHelloRequest"/>
            <output message="tns:GetHelloResponse"/>
        </operation>
    </portType>
    <binding name="HelloBinding" type="tns:HelloPortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/sc
        <operation name="getHello">
            <soap:operation soapAction="http://example.com/soap-web-service/get
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="HelloService">
        <port name="HelloPort" binding="tns:HelloBinding">
            <soap:address location="http://localhost:8080/soap-api"/>
        </port>
    </service>
</definitions>
```

2. **Create the XML Schema Definition (XSD):**

   Create a file named `hello.xsd` with the following content:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://example.com/soap-web-service"
```

```
            targetNamespace="http://example.com/soap-web-service"
            elementFormDefault="qualified">
        <element name="GetHelloRequest">
            <complexType>
                <sequence>
                    <element name="name" type="string"/>
                </sequence>
            </complexType>
        </element>
        <element name="GetHelloResponse">
            <complexType>
                <sequence>
                    <element name="greeting" type="string"/>
                </sequence>
            </complexType>
        </element>
    </schema>
```

3. Create the Web Service Endpoint:

Create a class named `HelloEndpoint`:

```
import org.example.soap_web_service.GetHelloRequest;
import org.example.soap_web_service.GetHelloResponse;
import org.springframework.stereotype.Component;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;

@Endpoint
@Component
public class HelloEndpoint {
    private static final String NAMESPACE_URI = "http://example.com/soap-web-se
    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "GetHelloRequest")
    @ResponsePayload
    public GetHelloResponse getHello(@RequestPayload GetHelloRequest request) {
        GetHelloResponse response = new GetHelloResponse();
        String name = request.getName();
        String greeting = "Hello, " + name + "!";
        response.setGreeting(greeting);
        return response;
    }
}
```

4. **Configure the Web Service:**

Create a configuration class named `WebServiceConfig`:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.ws.config.annotation.EnableWs;
import org.springframework.ws.config.annotation.WsConfigurerAdapter;
import org.springframework.ws.server.endpoint.adapter.DefaultMethodEndpointAdap
import org.springframework.ws.transport.http.MessageDispatcherServlet;
import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;
import org.springframework.ws.wsdl.wsdl11.Wsdl11Definition;

@EnableWs
@Configuration
public class WebServiceConfig extends WsConfigurerAdapter {
    @Bean
    public DefaultMethodEndpointAdapter defaultMethodEndpointAdapter() {
        return new DefaultMethodEndpointAdapter();
    }
    @Bean
    public MessageDispatcherServlet messageDispatcherServlet() {
        return new MessageDispatcherServlet();
    }
    @Bean(name = "hello")
    public Wsdl11Definition helloWsdl11Definition() {
        DefaultWsdl11Definition wsdl11Definition = new DefaultWsdl11Definition(
        wsdl11Definition.setPortTypeName("HelloPortType");
```

Open in app ↗ Sign up Sign in

Medium Search

```
    @Bean
    public SimpleXsdSchema helloSchema() {
        return new SimpleXsdSchema(new ClassPathResource("hello.xsd"));
    }
}
```

5. **Create the Application Entry Point:**

Create the main class `SoapWebServiceApplication`:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class SoapWebServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(SoapWebServiceApplication.class, args);
    }
}
```

6. **Run the Application:**
Run the `SoapWebServiceApplication` class as a Java application.

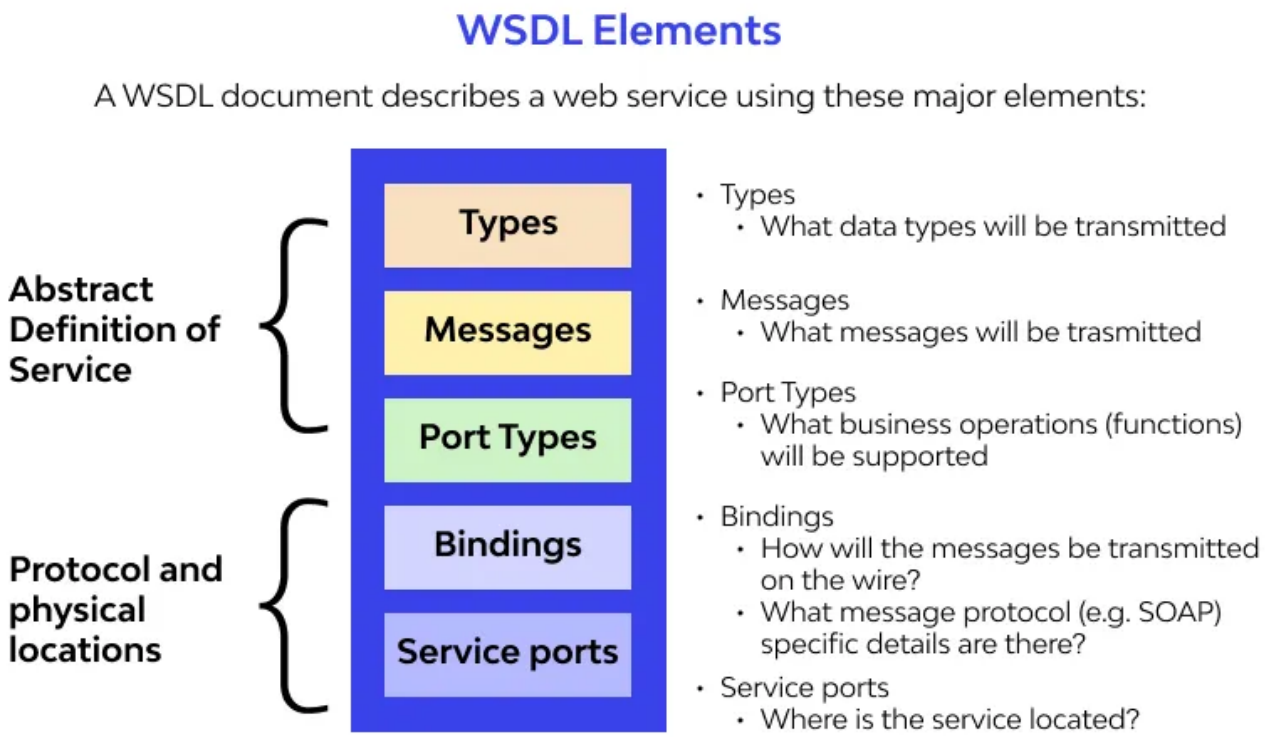
Congratulations! You have successfully created a simple SOAP web service using Spring Boot. The web service will be available at `http://localhost:8080/soap-api` and can be accessed using the defined WSDL (`hello.wsdl`). You can test the web service using SOAP clients or tools like SoapUI.

> ***Note**: Make sure to include the necessary dependencies such as `spring-boot-starter-web-services` and `org.springframework.boot:spring-boot-starter-web` in your `pom.xml` or `build.gradle` file.*
>
> *When you run the application, Spring Web Services uses the provided WSDL and XSD files to generate the necessary Java classes for the request and response objects. The generated classes will be located in the appropriate package based on the target namespace specified in the WSDL.*
>
> *In this case, the `GetHelloResponse` class will be automatically generated based on the `GetHelloResponse` message defined in the WSDL (`hello.wsdl`) and the associated XML schema definition (`hello.xsd`). You can use it in the `HelloEndpoint` class as shown in the example.*
>
> *Since the generation of these classes is handled by Spring Web Services at runtime, there's no explicit `GetHelloResponse.java` file in the example code. The class is generated dynamically based on the WSDL and XSD files provided.*

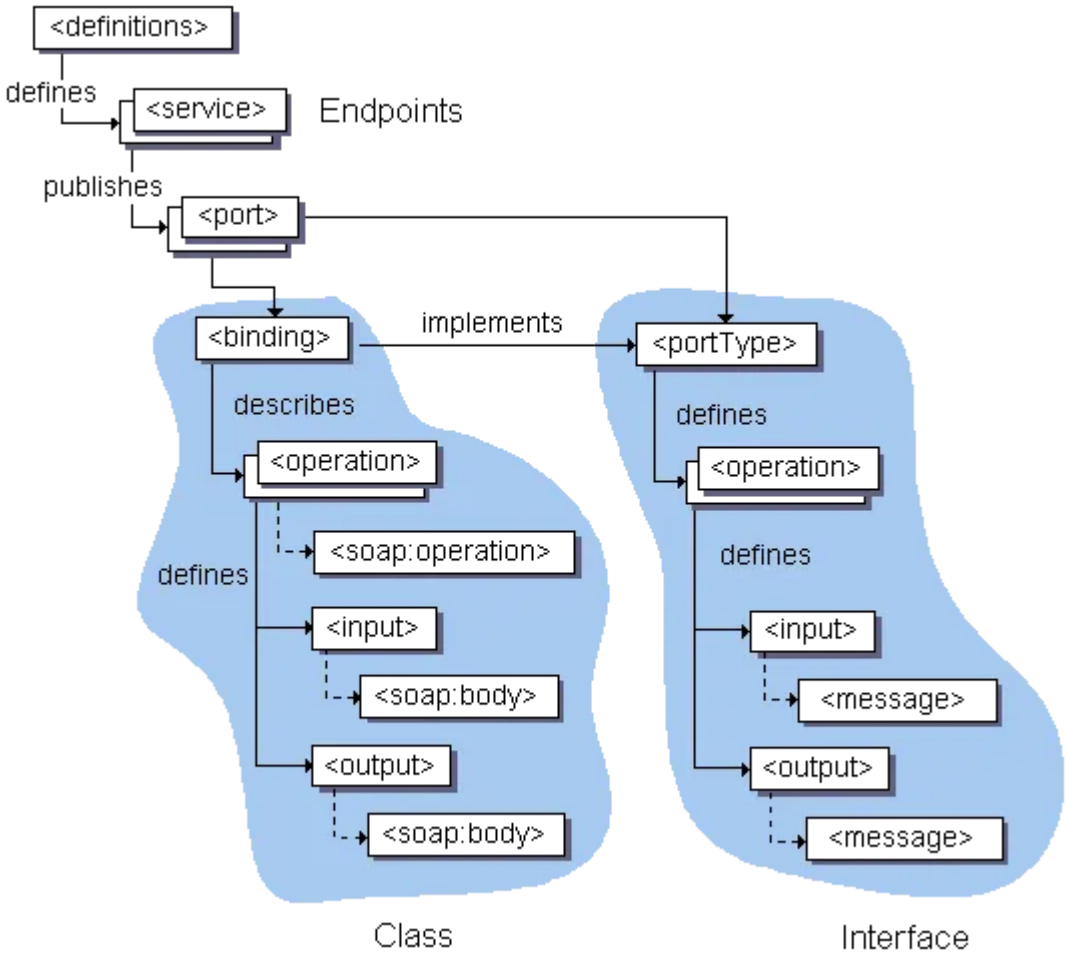
Conclusion

In this blog post, we explored the essentials of developing SOAP web services with Spring Boot. We discussed the fundamental concepts, walked through the steps involved in building a SOAP-based application, and covered important aspects such as testing, error handling, security, and deployment. By leveraging the power of

Spring Boot and its ecosystem, developers can create robust and scalable SOAP web services with ease.



WSDL Elements



WSDL Structure

Web Services API Spring Boot Java

Follow

Written by Extio Technology

594 Followers

Building the next generation virtualization layer for the cloud, virtual Kubernetes clusters.

More from Extio Technology

Extio Technology

Mastering Kubernetes Pod-to-Pod Communication: A Comprehensive Guide

Introduction

6 min read · Jun 16, 2023

127

Extio Technology

Understanding JSON Web Tokens (JWT): A Secure Approach to Web Authentication

Introduction

5 min read · Jul 28, 2023

17

Extio Technology

A Comprehensive Guide to Linux File System Types

Introduction

5 min read · Aug 2, 2023

12

Extio Technology

Kubernetes Authentication with OIDC: Simplifying Identity Management

Introduction

5 min read · Jun 22, 2023

41 1

See all from Extio Technology

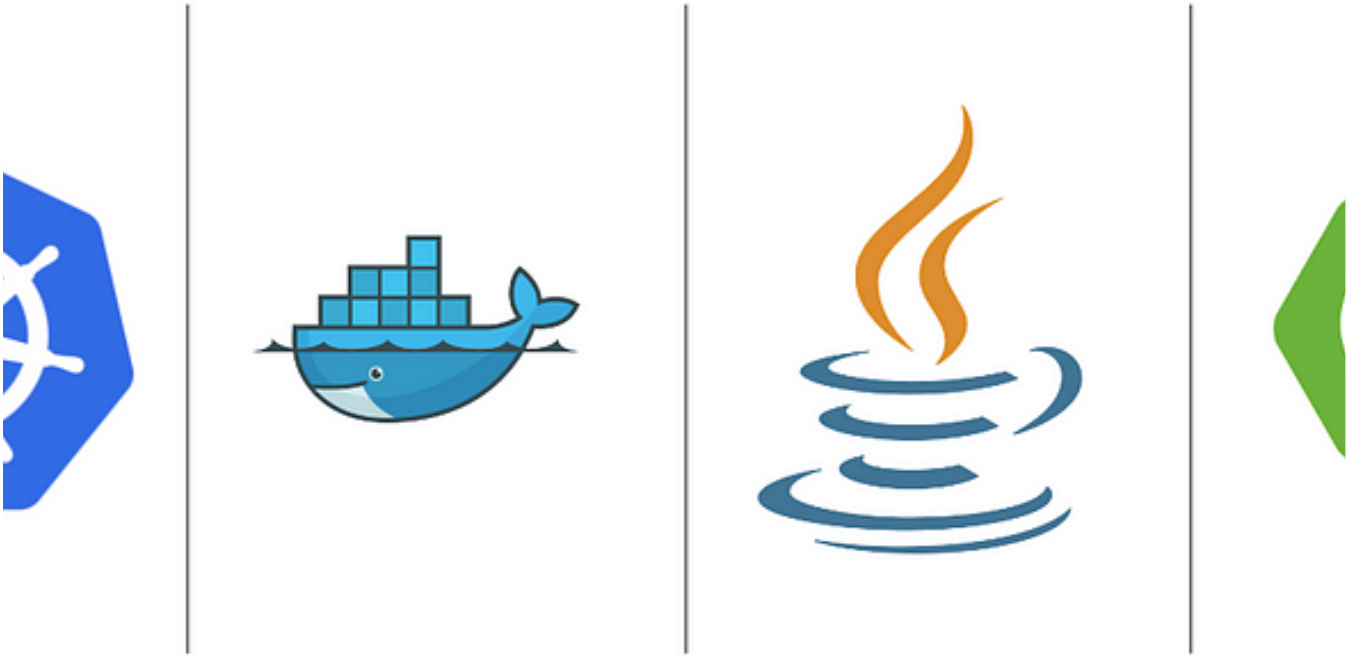
Recommended from Medium

Dwiki Witman

Spring Boot 3 Template (Part 7)—Configuring Swagger

Configuring Swagger in Spring Boot generates API docs and provides an interactive UI for easy exploration and testing.

3 min read · Dec 21, 2023



Truong Bui

Exploring Kubernetes: A Beginner’s Approach to Spring Boot Application Deployment

Recently, in my self-directed learning journey, Kubernetes caught my interest, prompting me to plan to write an article about it. Learning...

13 min read · Oct 4, 2023

14

Lists

Coding & Development

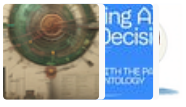
11 stories · 449 saves

Company Offsite Reading List

8 stories · 90 saves

General Coding Knowledge

20 stories · 924 saves



data science and AI

40 stories · 78 saves

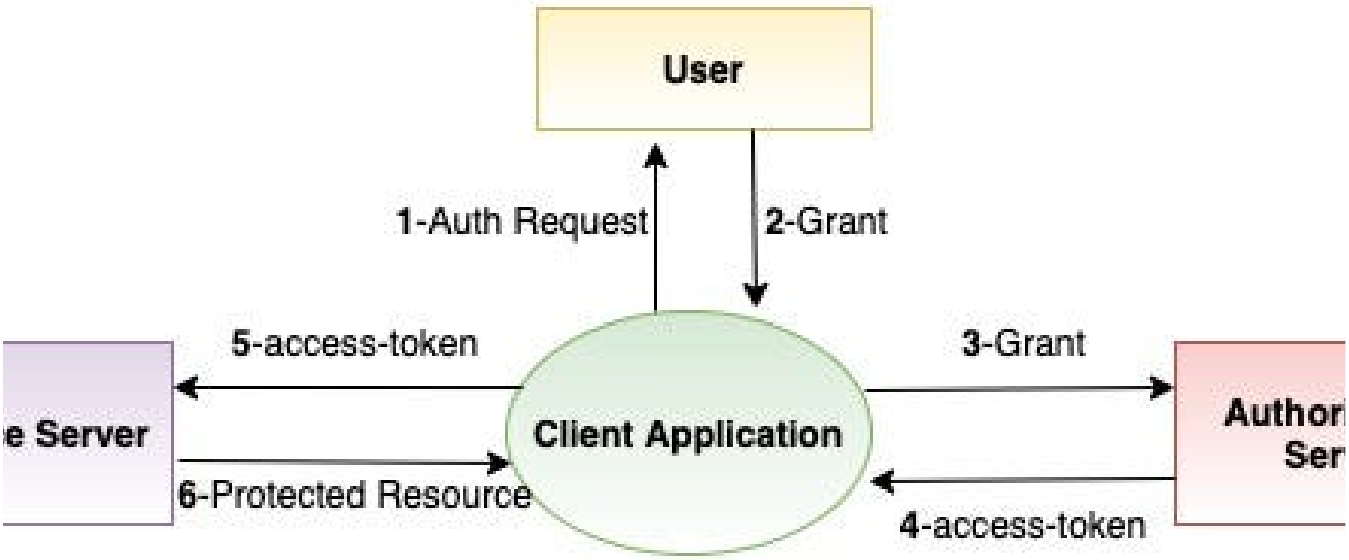
The Code Bean

Implementing JWT Authentication in a Spring Boot Application

JSON Web Tokens (JWT) have become a popular method for securing modern web applications. JWTs allow you to transmit information securely...

3 min read · Sep 19, 2023

108 1



Suvesh Agnihotri

Oauth2 with Spring Boot

What is oauth2?

10 min read · Sep 20, 2023

78

Varsha Das in Javarevisited

You don’t know Java Streams in-practice , Do You?

Why You Should Learn Java Streams Today

5 min read · Jan 9, 2024

1K 8

Ozzie Feliciano

Gateways To Glory: Forge Fortresses With Spring Cloud Gateway Mastery

n the ever-evolving landscape of microservices architecture, building resilient, scalable, and secure APIs is paramount. Enter the world of...

40 min read · Aug 30, 2023

2

See more recommendations