# Laporan Konversi Program Layanan Perpustakaan dari JSON-RPC ke SOAP

## Pendahuluan

Laporan ini menjelaskan proses konversi program layanan perpustakaan yang sebelumnya menggunakan JSON-RPC ke layanan berbasis SOAP. Layanan perpustakaan ini memiliki fitur untuk menambah koleksi buku, mendapatkan semua koleksi buku, dan mencari buku berdasarkan kata kunci. Konversi ini dilakukan untuk memenuhi standar layanan web yang lebih formal dan untuk meningkatkan interoperabilitas dengan sistem lain.

## Tujuan

Tujuan dari konversi ini adalah untuk: 1. Meningkatkan kompatibilitas layanan dengan sistem lain yang menggunakan protokol SOAP. 2. Menyediakan struktur yang lebih formal untuk komunikasi antara klien dan server. 3. Mempertahankan semua fitur yang ada dalam layanan perpustakaan sebelumnya.

## Deskripsi Sistem

Sistem ini adalah aplikasi layanan perpustakaan yang dibangun dengan menggunakan Spring Boot. Sebelumnya, aplikasi ini menggunakan JSON-RPC sebagai protokol komunikasi. Dengan konversi ini, aplikasi akan menggunakan SOAP sebagai protokol komunikasi. Fitur utama dari aplikasi ini meliputi: - Menambahkan koleksi buku. - Mendapatkan semua koleksi buku. - Mencari buku berdasarkan kata kunci.

## Langkah-langkah Konversi

### 1. Menyiapkan Proyek

* Menggunakan proyek yang sudah ada.
* Menambahkan dependensi yang diperlukan untuk SOAP dalam file pom.xml:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web-services</artifactId>  
</dependency>  
<dependency>  
 <groupId>jakarta.xml.bind</groupId>  
 <artifactId>jakarta.xml.bind-api</artifactId>  
 <version>4.0.0</version>  
</dependency>  
<dependency>  
 <groupId>org.glassfish.jaxb</groupId>  
 <artifactId>jaxb-runtime</artifactId>  
 <version>3.0.2</version>  
</dependency>  
<dependency>  
 <groupId>wsdl4j</groupId>  
 <artifactId>wsdl4j</artifactId>  
</dependency>

### 2. Mengatur Koneksi Database

Mengatur koneksi ke database MySQL dalam file application.properties, di sini saya membuat databse baru bernama perpustakaansoap\_db:

spring.application.name=perpustakaan  
spring.datasource.url=jdbc:mysql://localhost:3306/perpustakaansoap\_db  
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver  
spring.datasource.username=root  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

### 3. Membuat Kelas Entity

kelas entity Book yang merepresentasikan tabel books di database, masih sama seperti yang saya gunakan pada proyek JSON-RPC sebelumnya:

package com.polstat.perpustakaan.entity;  
  
import jakarta.persistence.Column;  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import jakarta.persistence.Table;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
@Setter  
@Getter  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Entity  
@Table(name = "books")  
public class Book {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 @Column(nullable = false)  
 private String title;  
 @Column(nullable = false)  
 private String author;  
 @Column(nullable = true)  
 private String description;  
}

### 4. Membuat Web Service Configuration

Membuat konfigurasi untuk SOAP Web Service:

package com.polstat.perpustakaan.config;  
  
import org.springframework.boot.web.servlet.ServletRegistrationBean;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.ws.config.annotation.EnableWs;  
import org.springframework.ws.transport.http.MessageDispatcherServlet;  
import org.springframework.ws.wsdl.wsdl11.DefaultWsdl11Definition;  
import org.springframework.xml.xsd.SimpleXsdSchema;  
import org.springframework.xml.xsd.XsdSchema;  
  
@EnableWs  
@Configuration  
public class WebServiceConfig {  
 @Bean  
 public ServletRegistrationBean<MessageDispatcherServlet> messageDispatcherServlet(ApplicationContext context) {  
 MessageDispatcherServlet servlet = new MessageDispatcherServlet();  
 servlet.setApplicationContext(context);  
 servlet.setTransformWsdlLocations(true);  
 return new ServletRegistrationBean<>(servlet, "/ws/\*");  
 }  
  
 @Bean(name = "library")  
 public DefaultWsdl11Definition defaultWsdl11Definition(XsdSchema librarySchema) {  
 DefaultWsdl11Definition definition = new DefaultWsdl11Definition();  
 definition.setPortTypeName("LibraryPort");  
 definition.setLocationUri("/ws");  
 definition.setTargetNamespace("http://www.polstat.com/perpustakaan");  
 definition.setSchema(librarySchema);  
 return definition;  
 }  
  
 @Bean  
 public XsdSchema librarySchema() {  
 return new SimpleXsdSchema(new ClassPathResource("library.xsd"));  
 }  
}

### 5. Membuat Schema (XSD)

Membuat file library.xsd untuk mendefinisikan struktur pesan SOAP:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.polstat.com/perpustakaan" targetNamespace="http://www.polstat.com/perpustakaan" elementFormDefault="qualified">  
  
 <xs:element name="getBooksRequest">  
 </xs:element>  
  
 <xs:element name="getBooksResponse">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="books" type="tns:book" maxOccurs="unbounded" />  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
  
 <xs:element name="addBookRequest">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="title" type="xs:string" />  
 <xs:element name="author" type="xs:string" />  
 <xs:element name="description" type="xs:string" minOccurs="0" />  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
  
 <xs:element name="addBookResponse">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="status" type="xs:string" />  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
  
 <xs:element name="searchBooksRequest">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="keyword" type="xs:string" />  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
  
 <xs:element name="searchBooksResponse">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="books" type="tns:book" maxOccurs="unbounded" />  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
  
 <xs:complexType name="book">  
 <xs:sequence>  
 <xs:element name="id" type="xs:long" />  
 <xs:element name="title" type="xs:string" />  
 <xs:element name="author" type="xs:string" />  
 <xs:element name="description" type="xs:string" minOccurs="0" />  
 </xs:sequence>  
 </xs:complexType>  
  
</xs:schema>

### 6. Membuat Service dan Controller

Kelas interface service dan serviceImpl saya sesuaikan dengan variabel baru yang saya gunakan di library.xsd, yaitu GetBookById:

package com.polstat.perpustakaan.service;  
import com.polstat.perpustakaan.dto.BookDto;  
import com.polstat.perpustakaan.entity.Book;  
import com.polstat.perpustakaan.mapper.BookMapper;  
import com.polstat.perpustakaan.repository.BookRepository;  
import java.util.List;  
import java.util.stream.Collectors;  
  
import jakarta.persistence.EntityNotFoundException;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
@Service  
public class BookServiceImpl implements BookService{  
 private static final Logger logger = LoggerFactory.getLogger(BookServiceImpl.class);  
 @Autowired  
 private BookRepository bookRepository;  
  
 @Override  
 public void createBook(BookDto bookDto) {  
 bookRepository.save(BookMapper.mapToBook(bookDto));  
 }  
  
 @Override  
 public List<BookDto> getBooks() {  
 List<Book> books = bookRepository.findAll();  
 List<BookDto> bookDtos = books.stream()  
 .map((product) -> (BookMapper.mapToBookDto(product)))  
 .collect(Collectors.toList());  
 return bookDtos;  
 }  
  
 @Override  
 public List<BookDto> searchBooks(String keyword) {  
 List<Book> books = bookRepository.searchBooks(keyword);  
 return books.stream()  
 .map(BookMapper::mapToBookDto)  
 .collect(Collectors.toList());  
 }  
}

### 7. Membuat Endpoint SOAP

Membuat endpoint untuk menangani permintaan SOAP:

package com.polstat.perpustakaan.endpoint;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import com.polstat.perpustakaan.dto.BookDto;  
import com.polstat.perpustakaan.service.BookService;  
import generated.\*;  
import org.springframework.beans.factory.annotation.Autowired;  
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;  
  
  
  
import java.util.List;  
  
@Endpoint  
public class BookEndpoint {  
 private static final Logger logger = LoggerFactory.getLogger(BookEndpoint.class);  
  
 private static final String NAMESPACE\_URI = "http://www.polstat.com/perpustakaan";  
  
 @Autowired  
 private BookService bookService;  
  
 @PayloadRoot(namespace = NAMESPACE\_URI, localPart = "getBooksRequest")  
 @ResponsePayload  
 public GetBooksResponse getBooks() {  
 GetBooksResponse response = new GetBooksResponse();  
  
 List<BookDto> books = bookService.getBooks();  
 for (BookDto book : books) {  
 Book it = new Book();  
 it.setTitle(book.getTitle());  
 it.setAuthor(book.getAuthor());  
 it.setDescription(book.getDescription());  
  
 response.getBooks().add(it);  
 }  
  
 return response;  
 }  
  
 @PayloadRoot(namespace = NAMESPACE\_URI, localPart = "addBookRequest")  
 @ResponsePayload  
 public AddBookResponse addBook(@RequestPayload AddBookRequest request) {  
 AddBookResponse response = new AddBookResponse();  
 BookDto bookDto = new BookDto(null, request.getTitle(), request.getAuthor(), request.getDescription());  
 bookService.createBook(bookDto);  
 response.setStatus("Book added successfully");  
 return response;  
 }  
  
 @PayloadRoot(namespace = NAMESPACE\_URI, localPart = "searchBooksRequest")  
 @ResponsePayload  
 public SearchBooksResponse searchBooks(@RequestPayload SearchBooksRequest request) {  
 SearchBooksResponse response = new SearchBooksResponse();  
 List<BookDto> books = bookService.searchBooks(request.getKeyword());  
 for (BookDto bookDto : books) {  
 response.getBooks().add(mapToBook(bookDto));  
 }  
 return response;  
 }  
  
 private Book mapToBook(BookDto bookDto) {  
 Book book = new Book();  
 book.setId(bookDto.getId());  
 book.setTitle(bookDto.getTitle());  
 book.setAuthor(bookDto.getAuthor());  
 book.setDescription(bookDto.getDescription());  
 return book;  
 }  
}

## Contoh Pengujian

Setelah semua langkah konversi selesai, kita dapat menguji layanan SOAP menggunakan Postman dengan mengikuti instruksi yang sudah dibahas sebelumnya. Berikut adalah contoh pengujian untuk menambah buku, mendapatkan daftar buku, dan mencari buku.

### 1. Pengujian Menambah Buku

* **Request**:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:lib="http://www.polstat.com/perpustakaan">  
 <soapenv:Header/>  
 <soapenv:Body>  
 <lib:addBookRequest>  
 <lib:title>Sehari Bersama Bogeem</lib:title>  
 <lib:author>Bogeem Silaholo</lib:author>  
 <lib:description>Mengenal lebih dekat sang orang baik</lib:description>  
 </lib:addBookRequest>  
 </soapenv:Body>  
</soapenv:Envelope>

* **Response**:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  
 <soapenv:Body>  
 <addBookResponse xmlns="http://www.polstat.com/perpustakaan">  
 <message>Book added successfully</message>  
 </addBookResponse>  
 </soapenv:Body>  
</soapenv:Envelope>

### 2. Pengujian Mendapatkan Daftar Buku

* **Request**:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:per="http://www.polstat.com/perpustakaan">  
 <soapenv:Header/>  
 <soapenv:Body>  
 <per:getBooksRequest/>  
 </soapenv:Body>  
</soapenv:Envelope>

* **Response**:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  
 <soapenv:Body>  
 <getBooksResponse xmlns="http://www.polstat.com/perpustakaan">  
 <books>  
 <book>  
 <id>1</id>  
 <title>Learning Spring</title>  
 <author>John Doe</author>  
 <description>An introductory book to Spring Framework</description>  
 </book>  
 </books>  
 </getBooksResponse>  
 </soapenv:Body>  
</soapenv:Envelope>

### 3. Pengujian Mencari Buku

* **Request**:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:lib="http://www.polstat.com/perpustakaan">  
 <soapenv:Header/>  
 <soapenv:Body>  
 <lib:searchBooksRequest>  
 <lib:keyword>Spring</lib:keyword>  
 </lib:searchBooksRequest>  
 </soapenv:Body>  
</soapenv:Envelope>

* **Response**:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  
 <soapenv:Body>  
 <searchBooksResponse xmlns="http://www.polstat.com/perpustakaan">  
 <books>  
 <book>  
 <id>1</id>  
 <title>Learning Spring</title>  
 <author>John Doe</author>  
 <description>An introductory book to Spring Framework</description>  
 </book>  
 </books>  
 </searchBooksResponse>  
 </soapenv:Body>  
</soapenv:Envelope>

## Kesimpulan

Proses konversi layanan perpustakaan dari JSON-RPC ke SOAP telah berhasil dilakukan dengan mematuhi semua langkah yang diperlukan. Semua fitur yang ada sebelumnya tetap dipertahankan, dan aplikasi kini dapat berfungsi dengan baik sebagai layanan SOAP.