

2.1 Deskripsi Singkat

Web service adalah teknologi yang memungkinkan aplikasi untuk berkomunikasi dan berinteraksi melalui jaringan, terlepas dari bahasa pemrograman atau platform yang digunakan oleh masing-masing aplikasi. Web service memungkinkan integrasi yang mulus antara sistem yang berbeda, termasuk di antara organisasi yang berbeda, dengan menggunakan protokol standar berbasis web. Dua pendekatan utama dalam web service adalah XML-RPC dan JSON-RPC, yang merupakan protokol Remote Procedure Call (RPC) untuk pertukaran data melalui jaringan.

JSON-RPC adalah protokol yang memungkinkan pemanggilan fungsi atau prosedur di server melalui jaringan menggunakan format JSON sebagai bentuk pertukaran data. Dalam JSON-RPC, permintaan dikirim dalam format JSON yang berisi informasi tentang metode yang dipanggil dan parameter yang diperlukan. Server merespons dengan hasil pemanggilan atau pesan kesalahan dalam format JSON. JSON-RPC cocok untuk komunikasi yang cepat dan ringan, sering digunakan dalam aplikasi web dan mobile, serta arsitektur mikroservis.

XML-RPC adalah protokol yang memungkinkan pemanggilan fungsi atau prosedur di server melalui jaringan menggunakan format XML sebagai bentuk pertukaran data. Dalam XML-RPC, permintaan dikirim dalam format XML yang berisi informasi tentang metode yang dipanggil dan parameter yang diperlukan. Server merespons dengan hasil pemanggilan atau pesan kesalahan dalam format XML. XML-RPC lebih sederhana daripada beberapa alternatif seperti SOAP, dan cocok untuk situasi di mana kesederhanaan lebih diutamakan daripada efisiensi.

Baik JSON-RPC maupun XML-RPC bertujuan untuk memudahkan interaksi antara aplikasi yang berjalan di lingkungan yang berbeda. Keduanya menggunakan pendekatan Remote Procedure Call (RPC) untuk memungkinkan pemanggilan fungsi jarak jauh. Mereka juga memiliki komponen seperti metode yang dipanggil, parameter, dan ID transaksi

dalam permintaan dan respons. Perbedaan utama terletak pada format pertukaran data, di mana JSON-RPC menggunakan format JSON yang lebih ringan dan terbaca oleh manusia, sementara XML-RPC menggunakan format XML yang lebih lengkap dan serbaguna.

Web service dengan JSON-RPC dan XML-RPC merupakan solusi yang kuat untuk mengintegrasikan aplikasi yang berbeda di era digital yang semakin terhubung. Dengan menyediakan standar komunikasi yang sederhana melalui jaringan, baik JSON-RPC maupun XML-RPC memungkinkan pengembang untuk membangun aplikasi yang efisien, berdaya guna, dan interoperabel dalam berbagai lingkungan dan platform. Pemilihan protokol yang sesuai tergantung pada kebutuhan aplikasi, preferensi pengembang, serta kondisi jaringan dan sumber daya yang tersedia.

2.2 Tujuan Praktikum

Tujuan praktikum ini adalah membuat web service sederhana menggunakan JSON-RPC dan XML-RPC dengan Spring Framework.

Setelah praktikum ini kompetensi yang akan dicapai adalah mahasiswa mampu memahami konsep teknologi *web service* dan memahami implementasi *web service* dengan JSON-RPC dan XML-RPC.

2.3 Alokasi Waktu

Alokasi waktu pada praktikum ini adalah sebagai berikut :

1. Tatap muka di kelas : 50 menit
2. Kerja mandiri (mengerjakan penugasan) : 120 menit

2.4 Material Praktikum

Praktikum ini memerlukan beberapa material sebagai berikut:

- 1) Java Development Kit (JDK) versi 17 atau di atasnya,
- 2) Java IDE: Netbeans, IntelliJ IDEA, atau Spring Tool Suite (STS),

- 3) Maven 3.5+,
- 4) MySQL
- 5) Postman <https://www.postman.com/downloads/>

2.5 Kegiatan Praktikum

Pada kegiatan praktikum kali ini kita akan membuat layanan kalkulator dan layanan perpustakaan. Layanan kalkulator menyediakan fungsi penjumlahan, pengurangan, perkalian dan pembagian. Layanan ini akan diimplementasikan dengan JSON-RPC dan XML-RPC. Proyek JSON-RPC akan dibuat menggunakan Spring Web MVC sedangkan XML-RPC dibuat menggunakan library Apache XML RPC (tanpa Spring Framework).

Sedangkan layanan perpustakaan menyediakan fungsi menambahkan koleksi buku dan mendapatkan semua koleksi buku. Layanan ini akan diimplementasikan dengan JSON-RPC. Proyek layanan perpustakaan ini akan dibuat menggunakan Spring Web MVC dengan database MySQL untuk penyimpanan data buku.

A. Layanan Kalkulator dengan JSON-RPC

Berikut langkah-langkah untuk membuat layanan kalkulator menggunakan JSON-RPC.

Langkah 1: Persiapan Proyek

- 1) Buat proyek Spring Boot melalui Spring Initializr <https://start.spring.io/> dengan pengaturan seperti gambar berikut.

Meet the Spring team this August at SpringOne.

spring initializr

Project

- ☐ Gradle - Groovy
- ☐ Gradle - Kotlin
- ☒ Maven

Language

- ☒ Java
- ☐ Kotlin
- ☐ Groovy

Spring Boot

- ☐ 3.2.0 (SNAPSHOT)
- ☐ 3.2.0 (M1)
- ☐ 3.1.3 (SNAPSHOT)
- ☒ 3.1.2
- ☐ 3.0.10 (SNAPSHOT)
- ☐ 3.0.9
- ☐ 2.7.15 (SNAPSHOT)
- ☐ 2.7.14

Project Metadata

Group:

Artifact:

Name:

Description:

Package name:

Packaging: ☒ Jar ☐ War

Java: ☐ 20 ☒ 17 ☐ 11 ☐ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC.
Uses Apache Tomcat as the default embedded container.

Buttons: GENERATE CTRL + G, EXPLORE CTRL + SPACE, SHARE...

- 2) Buka proyek Spring Boot menggunakan NetBeans IDE atau IDE lainnya.
- 3) Setelah proyek dibuka, klik menu *Build with Dependencies* agar *dependency library* yang dibutuhkan dapat diunduh. Tunggu sampai prosesnya selesai.

Langkah 2: Menulis Kode Program

1) Membuat *Interface Service Calculator*

Interface Service Calculator mendefinisikan fungsi yang disediakan oleh layanan kalkulator yaitu penambahan, pengurangan, perkalian, dan pembagian.

Buatlah file interface *CalculatorService.java* pada package *com.example.calculator.service*. Berikut kode program selengkapnya.

```
package com.example.calculator.service;

public interface CalculatorService {
    public double add(double a, double b);
    public double subtract(double a, double b);
    public double multiply(double a, double b);
    public double divide(double a, double b);
}
```

2) Membuat implementasi Service Calculator

Interface Service Calculator yang telah didefinisikan harus dibuat implementasinya.

Buatlah file *CalculatorServiceImpl.java* yang meng-*implements interface CalculatorService.java*. Berikut kode program selengkapnya.

```
package com.example.calculator.service;

import org.springframework.stereotype.Service;

@Service
public class CalculatorServiceImpl implements CalculatorService{

    public double add(double a, double b) {
        return a + b;
    }

    public double subtract(double a, double b) {
        return a - b;
    }

    public double multiply(double a, double b) {
        return a * b;
    }

    public double divide(double a, double b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

Kode program di atas ditandai dengan anotasi *@Service*. Tujuannya agar dikenali oleh Spring sebagai objek *service* yang nantinya akan digunakan untuk menginisiasi objek *CalculatorService*.

3) Membuat kelas yang merepresentasi request dan response JSON-RPC.

Buatlah file *JsonRpcRequest.java* dan *JsonRpcResponse.java* pada package *com.example.calculator.rpc*. Berikut kode program selengkapnya.

```

package com.example.calculator.rpc;

import com.fasterxml.jackson.databind.JsonNode;

public class JsonRequest {
    private String jsonrpc;
    private String method;
    private JsonNode params;
    private String id;

    //tambahkan method getter dan setter
}

```

```

package com.example.calculator.rpc;

public class JsonResponse {
    private String jsonrpc;
    private Object result;
    private Object error;
    private String id;

    //tambahkan method getter dan setter
}

```

4) Membuat kelas JsonRequestController

Kelas JsonRequestController dibuat untuk menangani request dan response. Di dalam kelas ini ada metode yang digunakan untuk membaca request dalam format JSON. Selanjutnya memetakan service apa yang akan dieksekusi.

Buatlah *file JsonRequestController.java* pada package *com.example.calculator.controller*. Berikut kode program selengkapnya.

```

package com.example.calculator.controller;

import com.example.calculator.rpc.JsonRpcRequest;
import com.example.calculator.rpc.JsonRpcResponse;
import com.example.calculator.service.CalculatorService;
import com.fasterxml.jackson.databind.JsonNode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class JsonRequestController {

```

```

@Autowired
private CalculatorService calculatorService;

@PostMapping("/jsonrpc")
public ResponseEntity<Object> handleJsonRpcRequest(@RequestBody
JsonRpcRequest request) {
    try {
        String method = request.getMethod();
        JsonNode params = request.getParams();

        double a = params.get("a").asDouble();
        double b = params.get("b").asDouble();
        double result;

        switch (method) {
            case "add":
                result = calculatorService.add(a, b);
                return ResponseEntity.ok(new JsonRpcResponse(result, request.getId()));
            case "subtract":
                result = calculatorService.subtract(a, b);
                return ResponseEntity.ok(new JsonRpcResponse(result, request.getId()));
            case "multiply":
                result = calculatorService.multiply(a, b);
                return ResponseEntity.ok(new JsonRpcResponse(result, request.getId()));
            case "divide":
                result = calculatorService.divide(a, b);
                return ResponseEntity.ok(new JsonRpcResponse(result, request.getId()));
            default:
                return ResponseEntity.badRequest().build();
        }
    } catch (Exception e) {
        return ResponseEntity.badRequest().build();
    }
}

```

Langkah 3: Menjalankan Aplikasi

Jalankan *web service Calculator*. Klik kanan pada file *CalculatorApplication.java* dan pilih menu *Run File*. Tunggu sampai proses persiapan eksekusi program selesai. Layanan kalkulator akan tersedia pada url <http://localhost:8080/jsonrpc>

Selanjutnya uji JSON-RPC melalui aplikasi Postman. Berikut contoh request untuk menghitung penjumlahan 5+10.

1) Buat POST request dengan url <http://localhost:8080/jsonrpc>

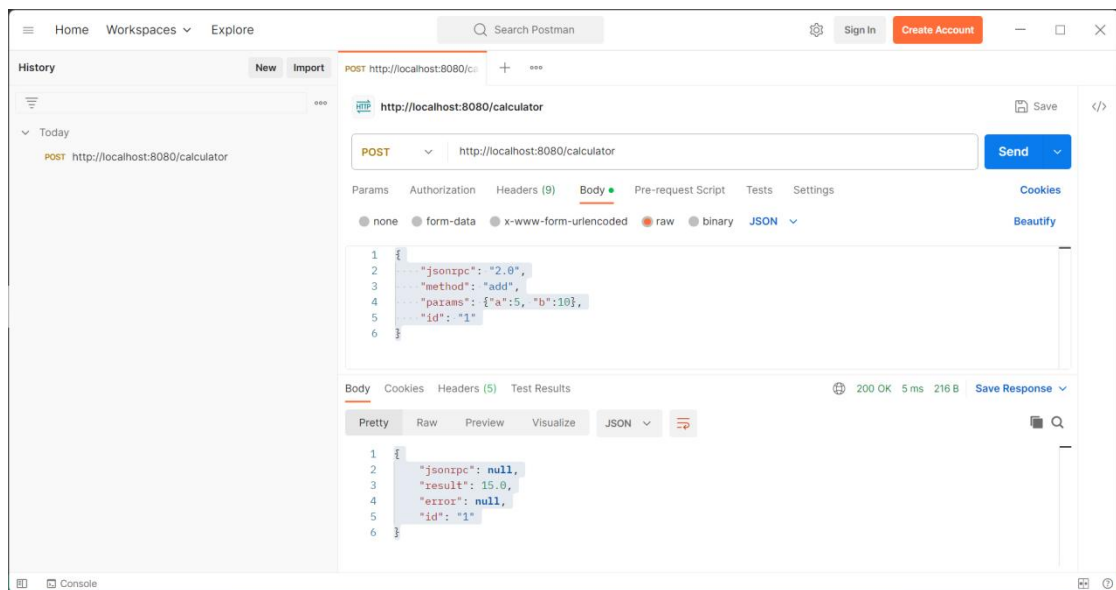
2) Tambahkan json berikut pada Body Request.

```
{
  "jsonrpc": "2.0",
  "method": "add",
  "params": {"a":5, "b":10},
  "id": "1"
}
```

3) Setelah dieksekusi akan menampilkan response sebagai berikut.

```
{
  "jsonrpc": null,
  "result": 15.0,
  "error": null,
  "id": "1"
}
```

Berikut tampilan pada Postman.



B. Layanan Kalkulator dengan XML-RPC

Berikut langkah-langkah membuat layanan kalkulator menggunakan XML-RPC dengan memanfaatkan library Apache XML RPC Server.

Langkah 1: Persiapan Proyek

1) Buat proyek Maven menggunakan Netbeans. Klik tombol New Project atau menu File>New Project...

New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: xmlrpc

Project Location: C:\Users\user\Documents\NetBeansProjects Brgwse...

Project Folder: C:\Users\user\Documents\NetBeansProjects\xmlrpc

Artifact Id: xmlrpc

Group Id: com.example

Version: 1.0-SNAPSHOT

Package: com.example.xmlrpc (Optional)

< Back Next > **Finish** Cancel Help

- 2) Isikan Project Name: xmlrpc dan Group Id: com.example. Klik Tombol Finish. Proyek xmlrpc akan dibuat dan ditampilkan dalam NetBeans.
- 3) Tambahkan dependency Apache XML RPC Server. Buka file pom.xml dan modifikasi seperti berikut.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>xmlrpc</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.apache.xmlrpc</groupId>
      <artifactId>xmlrpc-server</artifactId>
      <version>3.1.3</version>
    </dependency>
  </dependencies>
</project>
```

- 4) Setelah dependency ditambahkan, klik menu *Build with Dependencies* agar *dependency library* yang dibutuhkan dapat diunduh. Tunggu sampai prosesnya selesai.

Langkah 2: Menulis Kode Program

1) Membuat Kelas CalculatorService

Kelas ini memuat fungsi dasar kalkulator seperti penjumlahan, pengurangan, perkalian, dan pembagian. Fungsi-fungsi tersebutlah yang akan dibuatkan xml-rpc sehingga nantinya bisa diakses oleh aplikasi lainnya.

Buat *file* *CalculatorService.java* pada *package* *com.example.xmlrpc.service*. Berikut kode program selengkapnya.

```
package com.example.xmlrpc.service;

public class CalculatorService{

    public double add(double a, double b) {
        return a + b;
    }

    public double subtract(double a, double b) {
        return a - b;
    }

    public double multiply(double a, double b) {
        return a * b;
    }

    public double divide(double a, double b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

2) Membuat Kelas CalculatorApplication

Kelas ini memuat metode main sebagai titik awal aplikasi atau server xml rpc dijalankan. Di dalam metode main dibuat objek server yang akan menangani *request xml-rpc* yang masuk. Selanjutnya server memetakan service yang akan dieksekusi sesuai dengan request yang masuk.

Buatlah file *CalculatorApplication.java* dalam package *com.example.xmlrpc*. Berikut kode program selengkapnya.

```
package com.example.xmlrpc;

import com.example.xmlrpc.service.CalculatorService;
import org.apache.xmlrpc.server.PropertyHandlerMapping;
import org.apache.xmlrpc.server.XmlRpcServer;
import org.apache.xmlrpc.server.XmlRpcServerConfigImpl;
import org.apache.xmlrpc.webserver.WebServer;

public class CalculatorApplication {
    public static void main(String[] args) {
        try {
            System.out.println("Starting XML-RPC Calculator Service...");

            // Buat server XML-RPC pada port 8080
            WebServer webServer = new WebServer(8080);
            XmlRpcServer xmlRpcServer = webServer.getXmlRpcServer();

            // Konfigurasi handler mapping
            PropertyHandlerMapping phm = new PropertyHandlerMapping();
            phm.addHandler("calculator", CalculatorService.class);
            xmlRpcServer.setHandlerMapping(phm);

            // Konfigurasi server
            XmlRpcServerConfigImpl serverConfig = (XmlRpcServerConfigImpl)
xmlRpcServer.getConfig();
            serverConfig.setEnabledForExtensions(true);
            serverConfig.setContentLengthOptional(false);

            // Mulai server
            webServer.start();

            System.out.println("XML-RPC Calculator Service is running.");
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

Penjelasan:

- Kode pertama kali membuat objek *WebServer* yang akan mendengarkan permintaan pada port 8080. Objek *XmlRpcServer* digunakan untuk menangani permintaan xml-rpc dari klien.
- *PropertyHandlerMapping* digunakan untuk menghubungkan nama metode yang akan dipanggil oleh klien dengan kelas yang akan menjalankan metode tersebut. Ini seperti peta yang memberi tahu server apa yang harus dilakukan ketika permintaan untuk "calculator" datang. Dalam contoh ini, kita menghubungkan nama "calculator" dengan kelas *CalculatorService*.

Langkah 3: Menjalankan Aplikasi

Jalankan *web service Calculator*. Klik kanan pada file *CalculatorApplication.java* dan pilih menu *Run File*. Tunggu sampai proses persiapan eksekusi program selesai. XML-RPC akan siap menangani permintaan melalui url <http://localhost:8080>.

Selanjutnya uji JSON-RPC melalui aplikasi Postman. Berikut contoh request untuk menghitung penjumlahan 5+10.

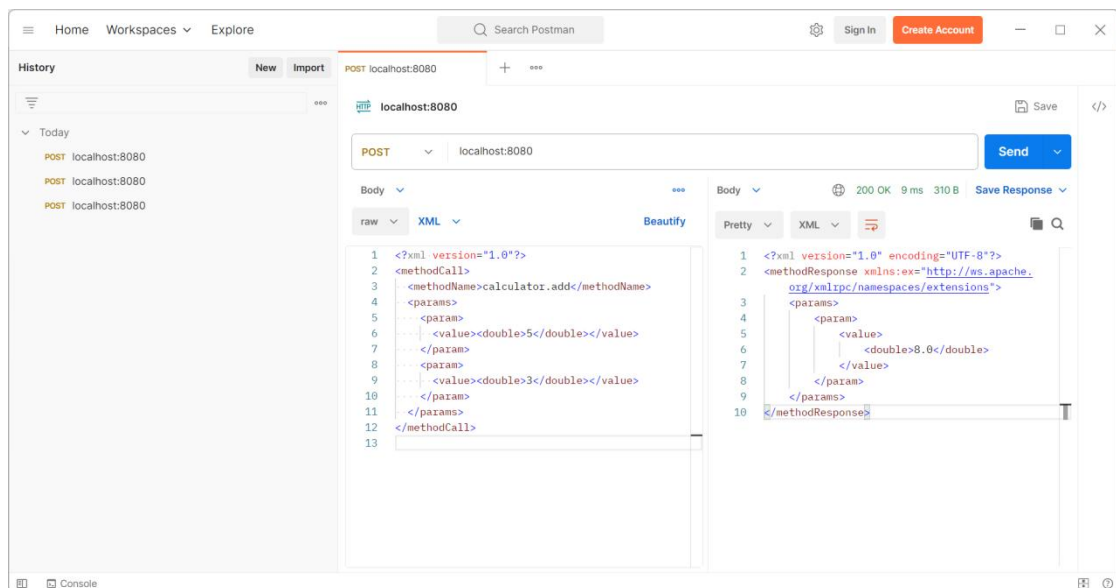
- 1) Buat POST request dengan url <http://localhost:8080>.
- 2) Tambahkan xml berikut pada Body Request.

```
<?xml version="1.0"?>
<methodCall>
  <methodName>calculator.add</methodName>
  <params>
    <param>
      <value><double>5</double></value>
    </param>
    <param>
      <value><double>3</double></value>
    </param>
  </params>
</methodCall>
```

- 3) Setelah dieksekusi akan menampilkan response berupa xml sebagai berikut.

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse xmlns:ex="http://ws.apache.org/xmlrpc/namespaces/extensions">
  <params>
    <param>
      <value>
        <i4>8</i4>
      </value>
    </param>
  </params>
</methodResponse>
```

Berikut tampilan pada Postman.



C. Layanan Perpustakaan dengan JSON-RPC

Layanan perpustakaan yang menyediakan fungsi menambah koleksi buku dan mendapatkan semua koleksi buku. Berikut langkah-langkahnya.

Langkah 1: Menyiapkan Proyek

- 1) Buat proyek Spring Boot menggunakan Spring Initializr dengan pengaturan seperti gambar berikut.

Meet the Spring team this August at SpringOne.

spring initializr

Project
☐ Gradle - Groovy
☐ Gradle - Kotlin
☒ **Maven**

Language
☒ **Java** ☐ Kotlin
☐ Groovy

Spring Boot
☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (M2) ☐ 3.1.4 (SNAPSHOT)
☒ **3.1.3** ☐ 3.0.11 (SNAPSHOT) ☐ 3.0.10
☐ 2.7.16 (SNAPSHOT) ☐ 2.7.15

Project Metadata
 Group
 Artifact
 Name
 Description
 Package name
 Packaging ☒ **Jar** ☐ War
 Java ☐ 20 ☒ **17** ☐ 11 ☐ 8

Dependencies ADD DEPENDENCIES... CTRL + B
Spring Web WEB
 Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Spring Data JPA SQL
 Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
MySQL Driver SQL
 MySQL JDBC driver.
Lombok DEVELOPER TOOLS
 Java annotation library which helps to reduce boilerplate code.

GENERATE CTRL + G
EXPLORE CTRL + SPACE
SHARE...

Library yang dibutuhkan antara lain:

- a) Spring Web
 - b) Spring Data JPA
 - c) MySQL Driver
 - d) Lombok
 - e) Jakarta Validation API
- 2) Buka proyek Spring Boot menggunakan NetBeans IDE atau IDE lainnya.
 - 3) Tambahkan library Jakarta Validation API pada file pom.xml.

```
<dependency>
  <groupId>jakarta.validation</groupId>
  <artifactId>jakarta.validation-api</artifactId>
  <version>3.0.2</version>
  <type>jar</type>
</dependency>
```

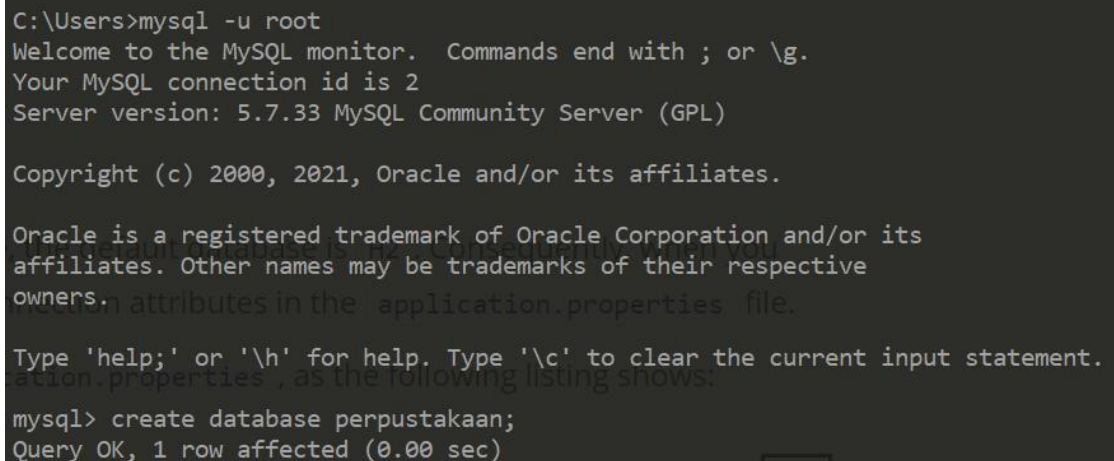
- 4) Setelah proyek dibuka, klik menu *Build with Dependencies* agar *dependency library* yang dibutuhkan dapat diunduh. Tunggu sampai prosesnya selesai.

Langkah 2: Membuat Kode Program

1) Membuat Database

Buat database MySQL dengan nama **perpustakaan**. Pembuatan database MySQL dapat menggunakan aplikasi phpMyAdmin, HeidiSQL, Navicat, atau aplikasi semisal. Berikut ini contoh membuat database MySQL melalui *command prompt*.

- a) Pastikan di komputer kita sudah terpasang database MySQL.
- b) Buka *command prompt* (Window) atau *terminal* (MacOS/Linux)
- c) Jalankan perintah ini: ***mysql -u root***
- d) Jalankan perintah membuat database: ***create database perpustakaan;***



```
C:\Users>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.33 MySQL Community Server (GPL)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> create database perpustakaan;
Query OK, 1 row affected (0.00 sec)
```

2) Mengatur Koneksi Database

Atur koneksi ke database pada proyek Spring Boot. Modifikasi file `src/main/resources/application.properties` seperti berikut.

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

3) Membuat Kelas Entity

Kelas entity adalah kelas yang merepresentasikan tabel dalam database. Kelas entity yang dibuat pada proyek ini adalah kelas entity Book dengan atribut id, title, author, description.

Buatlah *file Book.java* pada *package com.polstat.perpustakaan.entity*. Berikut kode program selengkapnya.

src/main/java/com/polstat/perpustakaan/entity/Book.java:

```
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;
}
```

Kode program Book.java di atas menggunakan anotasi yang disediakan oleh library Lombok antara lain:

- @Setter dan @Getter: untuk men-generate metode setter dan getter secara otomatis saat program di-*compile/build*.
- @AllArgsConstructor dan @NoArgsConstructor: untuk men-generate *constructor* secara otomatis saat program di-*compile/build*.

4) Membuat Repository

Repository adalah kelas yang menangani operasi database seperti insert, update, delete, dan query. Spring menyediakan kemudahan dalam mengimplementasikan operasi tersebut. Library yang menanganinya adalah Spring Data JPA. Dalam proyek ini, kita cukup membuat satu interface yang men-extends kelas JpaRepository.

Buatlah *interface BookRepository.java* pada *package com.polstat.perpustakaan.repository*. Berikut kode program selengkapnya.

```
package com.polstat.perpustakaan.repository;

import com.polstat.perpustakaan.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long>{

}
```

5) Membuat Kelas Data Transfer Object (DTO)

Kelas DTO adalah kelas yang digunakan untuk mengemas data yang akan dikirimkan antara lapisan dalam aplikasi, seperti antara controller dan service, atau antara service dan database. Kelas DTO ini membantu memisahkan antara struktur data yang dikirimkan melalui antarmuka publik dari entitas atau objek domain yang mendasarinya, sehingga meminimalkan ketergantungan dan memungkinkan manipulasi data yang lebih terkontrol.

Kelas DTO yang dibuat dalam proyek ini adalah BookDto dengan atribut id, title, author, dan description.

Buatlah *file BookDto.java* dalam *package com.polstat.perpustakaan.dto*.
Berikut kode program selengkapnya.

```
package com.polstat.perpustakaan.dto;
import jakarta.validation.constraints.NotEmpty;
import jakarta.validation.constraints.NotNull;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class BookDto {
    private Long id;

    @NotEmpty(message = "Judul buku wajib diisi.")
    private String title;

    @NotNull(message = "Penulis buku wajib diisi.")
    private String author;

    private String description;
}
```

6) Membuat Mapper Kelas Entity dan Kelas DTO

Tujuannya adalah untuk mengkonversi kelas Entity Book ke Kelas DTO BookDto atau sebaliknya. Berikut kode program selengkapnya.

```
package com.polstat.perpustakaan.mapper;

import com.polstat.perpustakaan.dto.BookDto;
import com.polstat.perpustakaan.entity.Book;

public class BookMapper {
    public static Book mapToBook(BookDto bookDto){
        return Book.builder()
            .id(bookDto.getId())
            .title(bookDto.getTitle())
            .description(bookDto.getDescription())
            .author(bookDto.getAuthor())
            .build();
    }
}
```

```

    }
    public static BookDto mapToBookDto(Book book){
        return BookDto.builder()
            .id(book.getId())
            .title(book.getTitle())
            .description(book.getDescription())
            .author(book.getAuthor())
            .build();
    }
}

```

7) Membuat Service

Service berisi bisnis logic, validasi, dan pemrosesan data. Komponen-komponen ini menerapkan aturan bisnis tertentu dan dapat berinteraksi dengan lapisan akses data.

Service yang dibuat dalam proyek ini adalah BookService yang menyediakan fungsi menambah koleksi buku yang disimpan ke database dan mendapatkan semua koleksi buku dari database. Service didefinisikan dalam interface dengan dua method yaitu createBook() dan getBooks().

Buatlah *interface BookService* pada *package com.polstat.perpustakaan.service*. Berikut kode program selengkapnya.

```

package com.polstat.perpustakaan.service;

import com.polstat.perpustakaan.dto.BookDto;
import java.util.List;

public interface BookService {
    public void createBook(BookDto bookDto);
    public List<BookDto> getBooks();
}

```

Selanjutnya interface BookService diimplementasikan dalam kelas BookServiceImpl. Berikut kode programnya.

```

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 org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class BookServiceImpl implements BookService{

    @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;
    }
}

```

8) Membuat kelas yang merepresentasi request dan response JSON-RPC.

Buatlah file *JsonRpcRequest.java* dan *JsonRpcResponse.java* pada package *com.polstat.perpustakaan.rpc*. Berikut kode program selengkapnya.

```

package com.polstat.perpustakaan.rpc;

import com.fasterxml.jackson.databind.JsonNode;

public class JsonRpcRequest {
    private String jsonrpc;
    private String method;
}

```

```

private JsonNode params;
private String id;

//tambahkan method getter dan setter
}

package com.polstat.perpustakaan.rpc;

public class JsonResponse {
    private String jsonrpc;
    private Object result;
    private Object error;
    private String id;

    //tambahkan method getter dan setter
}

```

9) Membuat Controller

Kelas `JsonRpcController` dibuat untuk menangani request dan response. Di dalam kelas ini ada metode yang digunakan untuk membaca request dalam format JSON. Selanjutnya memetakan service apa yang akan dieksekusi.

Buatlah file `JsonRpcController.java` pada package `com.example.calculator.controller`. Berikut kode program selengkapnya.

```

package com.polstat.perpustakaan.controller;

import com.fasterxml.jackson.databind.JsonNode;
import com.polstat.perpustakaan.dto.BookDto;
import com.polstat.perpustakaan.rpc.JsonRpcRequest;
import com.polstat.perpustakaan.rpc.JsonRpcResponse;
import com.polstat.perpustakaan.service.BookService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class JsonRpcController {

    @Autowired
    private BookService bookService;
}

```

```

@PostMapping("/jsonrpc")
public ResponseEntity<Object> handleJsonRpcRequest(@RequestBody
JsonRpcRequest request) {
    try {
        String method = request.getMethod();
        JsonNode params = request.getParams();
        System.out.println("Method: "+ method);
        switch (method) {
            case "createBook":
                String title = params.get("title").asText();
                String author = params.get("author").asText();
                String description = params.get("description").asText();
                BookDto book = BookDto.builder()
                    .title(title)
                    .description(description)
                    .author(author)
                    .build();

                bookService.createBook(book);
                return ResponseEntity.ok(new JsonRpcResponse("created",
request.getId()));
            case "getBooks":
                List<BookDto> books = bookService.getBooks();
                return ResponseEntity.ok(new JsonRpcResponse(books, request.getId()));

            default:
                return ResponseEntity.badRequest().build();
        }
    } catch (Exception e) {
        return ResponseEntity.badRequest().build();
    }
}
}

```

Langkah 3: Menjalankan Aplikasi

Jalankan proyek perpustakaan. Klik kanan pada file *PerpustakaanApplication.java* dan pilih menu *Run File*. Tunggu sampai proses persiapan eksekusi program selesai. Layanan perpustakaan akan tersedia pada url <http://localhost:8080/jsonrpc>

Selanjutnya uji JSON-RPC melalui aplikasi Postman. Berikut contoh request untuk menambahkan koleksi buku dan mendapatkan koleksi buku.

- 1) Buat POST request dengan url <http://localhost:8080/jsonrpc>

- 2) Untuk menambah koleksi buku, tambahkan json berikut pada Body Request.

```
{
  "jsonrpc": "2.0",
  "method": "createBook",
  "params": {"title": "Belajar Web Service", "author": "Budi", "description": "Ini adalah buku web service terbaik"},
  "id": "1"
}
```

- 3) Setelah dieksekusi akan menampilkan response sebagai berikut.

```
{
  "jsonrpc": null,
  "result": "created",
  "error": null,
  "id": "1"
}
```

- 4) Untuk mendapatkan koleksi buku, tambahkan json berikut pada Body Request.

```
{
  "jsonrpc": "2.0",
  "method": "getBooks",
  "params": {},
  "id": "2"
}
```

- 5) Setelah dieksekusi akan menampilkan response sebagai berikut.

```
{
  "jsonrpc": null,
  "result": [
    {
      "id": 3,
      "title": "Belajar Web Service",
      "author": "Budi",
      "description": "Ini adalah buku web service terbaik"
    }
  ],
  "error": null,
  "id": "2"
}
```

2.6 Penugasan

Lengkapi layanan perpustakaan di atas dengan menambahkan layanan pencarian buku berdasarkan kata kunci pencarian.

Buat laporan pekerjaan Anda dalam format pdf yang memuat penjelasan kode program yang Anda buat dan contoh pengujian yang menunjukkan layanan pencarian buku dapat berjalan dengan baik atau menghasilkan output yang sesuai.