Modul 10 Web Service

Tugas Pendahuluan :

1. Apa yang dimaksud dengan web service?
2. Apa fungsi web service dalam bahasa pemograman jaringan?
3. Apa perbedaan utama antara protokol SOAP dan REST?
4. Kapan kita sebaiknya menggunakan SOAP daripada REST, dan sebaliknya?

Materi :

Web Services adalah suatu teknologi yang memungkinkan aplikasi untuk berkomunikasi melalui jaringan internet. Secara sederhana, Web Services adalah aplikasi yang dirancang untuk berbagi data antar sistem dan aplikasi melalui protokol internet standar seperti HTTP, XML, dan SOAP.

Web Services biasanya dibuat dengan menggunakan bahasa pemrograman seperti Java, .NET, atau PHP, dan terdiri dari tiga komponen utama:

1. Deskripsi Layanan (Service Description): Merupakan deskripsi teknis tentang layanan yang diberikan, termasuk operasi apa saja yang bisa diakses oleh pengguna, format data yang didukung, dan protokol yang digunakan.
2. Protokol Pertukaran Pesan (Message Exchange Protocol): Merupakan protokol yang digunakan oleh aplikasi untuk mengirim dan menerima pesan antar sistem.
3. Protokol Pemanggilan Layanan (Service Invocation Protocol): Merupakan protokol yang digunakan oleh aplikasi untuk memanggil layanan yang disediakan oleh sistem lain.

Web Services dapat digunakan untuk berbagai keperluan, seperti integrasi antar aplikasi, berbagi data antar organisasi, dan penyediaan layanan publik. Web Services juga sangat fleksibel karena dapat digunakan oleh berbagai platform, bahasa pemrograman, dan sistem operasi yang berbeda.

Web Services di Java dapat dibuat dengan menggunakan Java API untuk XML Web Services (JAX-WS) atau Java API untuk RESTful Web Services (JAX-RS).

JAX-WS adalah teknologi Web Services berbasis SOAP di Java yang memungkinkan aplikasi untuk berkomunikasi melalui protokol standar seperti HTTP, XML, dan SOAP. JAX-WS menggunakan anotasi untuk mengidentifikasi operasi yang tersedia dan data yang digunakan dalam aplikasi Web Services. Selain itu, JAX-WS juga menyediakan tools untuk membuat WSDL (Web Services Description Language) yang memungkinkan pengguna untuk memahami dan menggunakan layanan Web Services yang disediakan.

Sementara itu, JAX-RS adalah teknologi Web Services berbasis RESTful di Java yang memungkinkan aplikasi untuk berkomunikasi melalui protokol HTTP dan JSON. JAX-RS menggunakan anotasi untuk mengidentifikasi operasi yang tersedia dan data yang digunakan dalam aplikasi Web Services. Selain itu, JAX-RS juga menyediakan tools untuk menghasilkan format data XML atau JSON, dan memungkinkan pengguna untuk membuat URL yang mudah dipahami oleh manusia.

Dalam pengembangan aplikasi Web Services di Java, terdapat beberapa tools yang sering digunakan, seperti Apache Axis2, Apache CXF, dan Spring Web Services. Tools ini memudahkan pengembang untuk membuat, memodifikasi, dan menguji Web Services dengan mudah.

Untuk membuat aplikasi Web Services di Java, terdapat beberapa kelas yang dapat digunakan, antara lain:

1. javax.jws.WebService: Kelas ini digunakan untuk mendefinisikan kelas Java sebagai Web Service. Kelas ini menyediakan anotasi seperti @WebService, @WebMethod, dan @WebParam yang digunakan untuk mengidentifikasi operasi yang tersedia dalam Web Service.
2. javax.xml.ws.Endpoint: Kelas ini digunakan untuk menerbitkan layanan Web Service. Kelas ini memungkinkan pengguna untuk membuat dan menerbitkan endpoint Web Service pada alamat tertentu.
3. javax.jws.WebMethod: Kelas ini digunakan untuk menandai metode Java sebagai operasi Web Service. Kelas ini menyediakan anotasi seperti @WebMethod dan @WebResult untuk mengidentifikasi parameter masukan dan keluaran yang digunakan dalam operasi.
4. javax.jws.WebParam: Kelas ini digunakan untuk mengidentifikasi parameter masukan pada operasi Web Service. Kelas ini menyediakan anotasi seperti @WebParam untuk menentukan nama dan jenis data parameter masukan.
5. javax.jws.WebResult: Kelas ini digunakan untuk mengidentifikasi keluaran dari operasi Web Service. Kelas ini menyediakan anotasi seperti @WebResult untuk menentukan nama dan jenis data keluaran.
6. javax.xml.bind.annotation.XmlRootElement: Kelas ini digunakan untuk menandai kelas Java sebagai elemen root dalam format XML. Kelas ini menyediakan anotasi seperti @XmlRootElement untuk menentukan nama elemen root dan namespace XML yang digunakan.
7. javax.xml.bind.annotation.XmlAccessorType: Kelas ini digunakan untuk menentukan jenis akses untuk mengakses elemen XML. Kelas ini menyediakan anotasi seperti @XmlAccessorType untuk menentukan apakah akses dilakukan melalui field atau melalui metode get/set.
8. javax.xml.bind.annotation.XmlType: Kelas ini digunakan untuk menentukan jenis tipe data XML. Kelas ini menyediakan anotasi seperti @XmlType untuk menentukan nama dan namespace XML yang digunakan.

Berikut adalah contoh sederhana pembuatan server dan client untuk Web Services menggunakan JAX-WS di Java.

Server:

```
1. import javax.jws.WebMethod;
2. import javax.jws.WebService;
3. import javax.xml.ws.Endpoint;
4.
5. @WebService
6. public class HelloWorld {
7.
8.     @WebMethod
9.     public String sayHello(String name) {
```

```
10.         return "Hello, " + name + "!";
11.     }
12.
13.     public static void main(String[] args) {
14.         Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
15.         System.out.println("Web Service Started");
16.     }
17. }
18.
```

CLient

```
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;

public class HelloWorldClient {

    public static void main(String[] args) throws Exception {
        URL url = new URL("http://localhost:8080/hello?wsdl");
        QName qname = new QName("http://webservice.example.com/", "HelloWorldService");
        Service service = Service.create(url, qname);
        HelloWorld hello = service.getPort(HelloWorld.class);
        System.out.println(hello.sayHello("John"));
    }
}
```

Pada contoh di atas, server akan melayani permintaan pada alamat http://localhost:8080/hello, dan akan memanggil operasi sayHello saat client melakukan request ke Web Service tersebut. Sedangkan pada client, akan melakukan request ke alamat http://localhost:8080/hello?wsdl untuk mendapatkan deskripsi Web Service, dan kemudian melakukan panggilan terhadap operasi sayHello dengan parameter "John". Output yang dihasilkan oleh client adalah "Hello, John!".

Untuk menjalankan contoh sederhana Web Services menggunakan JAX-WS di Java seperti yang telah saya jelaskan sebelumnya, Anda dapat mengikuti langkah-langkah berikut:

1. Buatlah file Java untuk server dengan nama HelloWorld.java, kemudian salin kode yang telah dijelaskan sebelumnya.
2. Buatlah file Java untuk client dengan nama HelloWorldClient.java, kemudian salin kode yang telah dijelaskan sebelumnya.
3. Kompilasi kedua file tersebut dengan menjalankan perintah berikut di terminal atau command prompt:

```
javac HelloWorld.java
javac HelloWorldClient.java
```

4. Setelah berhasil dikompilasi, jalankan server terlebih dahulu dengan menjalankan perintah berikut di terminal atau command prompt:

```
java HelloWorld
```

5. Setelah server berjalan, jalankan client dengan menjalankan perintah berikut di terminal atau command prompt:

```
java HelloWorldClient
```

Jika semuanya berjalan dengan benar, maka output yang dihasilkan oleh client adalah "Hello, John!".

Catatan: Pastikan bahwa Anda telah menginstal Java Development Kit (JDK) dan telah memasukkan direktori bin JDK ke dalam PATH system Anda. Selain itu, pastikan juga bahwa firewall pada komputer Anda tidak menghalangi koneksi ke port 8080 (atau port lain yang Anda tentukan pada server).

Web Service yang telah dibuat bisa dipanggil melalui browser dengan menggunakan metode HTTP GET atau POST. Namun, untuk memanggil Web Service melalui browser, kita perlu mengetahui URL dari Web Service tersebut dan juga format input/output yang diterima/dikembalikan oleh Web Service. Sebagai contoh, jika kita menggunakan Web Service "HelloWorld" yang telah dijelaskan sebelumnya, URL untuk memanggil operasi sayHello adalah sebagai berikut:

```
http://localhost:8080/hello/sayHello?name=John
```

Di mana "localhost:8080" adalah alamat server Web Service, "hello" adalah nama endpoint yang telah kita tentukan, "sayHello" adalah nama operasi yang akan dipanggil, dan "name" adalah parameter yang diberikan pada operasi tersebut. Jika kita membuka URL tersebut di browser, maka hasil yang akan ditampilkan adalah "Hello, John!". Namun, untuk memanggil Web Service dengan cara ini, kita harus memastikan bahwa Web Service yang kita buat mendukung metode HTTP GET atau POST dan input/output yang dikembalikan telah diformat agar dapat dibaca oleh browser.

Untuk mengembalikan output Web Service dalam format JSON di Java, kita dapat menggunakan library JSON seperti Jackson atau Gson. Berikut adalah contoh sederhana mengubah output Web Service menjadi format JSON menggunakan library Jackson di Java.

1. Tambahkan dependency Jackson di file pom.xml (jika menggunakan Maven), atau unduh library Jackson dan tambahkan ke classpath project Anda.
2. Ubah kode operasi sayHello pada kelas HelloWorld seperti berikut:

```
 1. import com.fasterxml.jackson.databind.ObjectMapper;
 2. import javax.ws.rs.*;
 3. import javax.ws.rs.core.MediaType;
 4.
 5. @WebService
 6. @Path("/hello")
 7. public class HelloWorld {
 8.
 9.     @GET
10.     @Path("/sayHello")
11.     @Produces(MediaType.APPLICATION_JSON)
12.     public String sayHello(@QueryParam("name") String name) throws
JsonProcessingException {
13.         ObjectMapper mapper = new ObjectMapper();
14.         String json = mapper.writeValueAsString("Hello, " + name + "!");
15.         return json;
16.     }
17.
```

```
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
        System.out.println("Web Service Started");
    }
}
```

Pada kode di atas, kita menambahkan anotasi @Path dan @GET untuk menentukan path dan method HTTP yang digunakan. Selain itu, kita juga menambahkan anotasi @Produces(MediaType.APPLICATION_JSON) untuk menandakan bahwa output yang dihasilkan adalah dalam format JSON.

Jalankan server seperti yang telah dijelaskan sebelumnya, kemudian coba panggil operasi sayHello dengan cara yang sama seperti sebelumnya. Jika semuanya berjalan dengan benar, maka output yang dihasilkan adalah sebagai berikut:

```
"Hello, John!"
```

Perlu diperhatikan bahwa keluaran tersebut dalam format JSON, dan kita dapat membaca dan memproses keluaran tersebut dengan mudah menggunakan library JSON seperti Jackson atau Gson. Untuk mengembalikan output Web Service dalam format JSON di Java, kita dapat menggunakan library Jackson. Berikut adalah contoh sederhana mengubah output Web Service menjadi format JSON menggunakan Jackson di Java.

1. Tambahkan dependency Jackson pada file pom.xml:

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.13.0</version>
</dependency>
```

2. Ubah kode operasi sayHello pada kelas HelloWorld seperti berikut:

```
import javax.jws.WebService;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import com.fasterxml.jackson.databind.ObjectMapper;

@WebService
@Path("/hello")
public class HelloWorld {

    @GET
    @Path("/sayHello")
    @Produces(MediaType.APPLICATION_JSON)
    public String sayHello(@QueryParam("name") String name) throws Exception {
        Greeting greeting = new Greeting("Hello, " + name + "!");
        ObjectMapper objectMapper = new ObjectMapper();
        String json = objectMapper.writeValueAsString(greeting);
        return json;
    }

    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
        System.out.println("Web Service Started");
```

```
    }
}

```

Pada kode di atas, kita menambahkan anotasi @Path dan @GET untuk menentukan path dan method HTTP yang digunakan. Selain itu, kita juga menambahkan anotasi @Produces(MediaType.APPLICATION_JSON) untuk menandakan bahwa output yang dihasilkan adalah dalam format JSON.

Jalankan server seperti yang telah dijelaskan sebelumnya, kemudian coba panggil operasi sayHello dengan cara yang sama seperti sebelumnya. Jika semuanya berjalan dengan benar, maka output yang dihasilkan adalah sebagai berikut:

```
{"message":"Hello, John!"}
```

Perlu diperhatikan bahwa keluaran tersebut dalam format JSON, dan kita dapat membaca dan memproses keluaran tersebut dengan mudah menggunakan library Jackson. Untuk mengembalikan output Web Service dalam format XML di Java, kita dapat menggunakan library JAXB (Java Architecture for XML Binding). Berikut adalah contoh sederhana mengubah output Web Service menjadi format XML menggunakan JAXB di Java.

1. Buatlah file XSD (XML Schema Definition) untuk mendefinisikan format output XML. Misalnya, kita akan membuat file XSD dengan nama hello.xsd dengan isi sebagai berikut:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="greeting" type="xs:string"/>
</xs:schema>
```

2. Buatlah kelas Java yang merepresentasikan objek yang akan dikonversi ke format XML. Misalnya, kita akan membuat kelas Greeting seperti berikut:

```
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Greeting {
    private String message;

    public Greeting() {}

    public Greeting(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

```
21.
```

3. Pada kelas di atas, kita menambahkan anotasi @XmlRootElement untuk menandakan bahwa kelas tersebut dapat dikonversi ke format XML. Ubah kode operasi sayHello pada kelas HelloWorld seperti berikut:

```
 1. import javax.jws.WebService;
 2. import javax.xml.bind.JAXBContext;
 3. import javax.xml.bind.JAXBException;
 4. import javax.xml.bind.Marshaller;
 5. import javax.ws.rs.*;
 6. import javax.ws.rs.core.MediaType;
 7. import java.io.StringWriter;
 8.
 9. @WebService
10. @Path("/hello")
11. public class HelloWorld {
12.
13.     @GET
14.     @Path("/sayHello")
15.     @Produces(MediaType.APPLICATION_XML)
16.     public String sayHello(@QueryParam("name") String name) throws JAXBException {
17.         Greeting greeting = new Greeting("Hello, " + name + "!");
18.         JAXBContext jaxbContext = JAXBContext.newInstance(Greeting.class);
19.         Marshaller marshaller = jaxbContext.createMarshaller();
20.         marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
21.         StringWriter stringWriter = new StringWriter();
22.         marshaller.marshal(greeting, stringWriter);
23.         return stringWriter.toString();
24.     }
25.
26.     public static void main(String[] args) {
27.         Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
28.         System.out.println("Web Service Started");
29.     }
30. }
31.
```

Pada kode di atas, kita menambahkan anotasi @Path dan @GET untuk menentukan path dan method HTTP yang digunakan. Selain itu, kita juga menambahkan anotasi @Produces(MediaType.APPLICATION_XML) untuk menandakan bahwa output yang dihasilkan adalah dalam format XML.

Jalankan server seperti yang telah dijelaskan sebelumnya, kemudian coba panggil operasi sayHello dengan cara yang sama seperti sebelumnya. Jika semuanya berjalan dengan benar, maka output yang dihasilkan adalah sebagai berikut:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<greeting>Hello, John!</greeting>
```

Perlu diperhatikan bahwa keluaran tersebut dalam format XML, dan kita dapat membaca dan memproses keluaran tersebut dengan mudah menggunakan library JAXB. Untuk mengembalikan

output Web Service dalam format XML di Java, kita dapat menggunakan library JAXB. Berikut adalah contoh sederhana mengubah output Web Service menjadi format XML menggunakan JAXB di Java.

Tambahkan dependency JAXB pada file pom.xml:

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>
```

Buat kelas Greeting seperti berikut:

```
 1. import javax.xml.bind.annotation.XmlRootElement;
 2.
 3. @XmlRootElement
 4. public class Greeting {
 5.
 6.     private String message;
 7.
 8.     public Greeting() {}
 9.
10.     public Greeting(String message) {
11.         this.message = message;
12.     }
13.
14.     public String getMessage() {
15.         return message;
16.     }
17.
18.     public void setMessage(String message) {
19.         this.message = message;
20.     }
21. }
22.
23. Kelas Greeting di atas diberi anotasi @XmlRootElement untuk menandakan bahwa kelas tersebut
dapat dikonversi ke format XML.
24. Ubah kode operasi sayHello pada kelas HelloWorld seperti berikut:
25. import javax.jws.WebService;
26. import javax.ws.rs.*;
27. import javax.ws.rs.core.MediaType;
28. import javax.xml.bind.JAXBContext;
29. import javax.xml.bind.Marshaller;
30.
31. @WebService
32. @Path("/hello")
33. public class HelloWorld {
34.
35.     @GET
36.     @Path("/sayHello")
37.     @Produces(MediaType.APPLICATION_XML)
38.     public Greeting sayHello(@QueryParam("name") String name) throws Exception {
39.         Greeting greeting = new Greeting("Hello, " + name + "!");
40.         return greeting;
41.     }
42.
43.     public static void main(String[] args) {
44.         Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
```

```
45.         System.out.println("Web Service Started");
46.     }
47. }
48.
```

Pada kode di atas, kita menambahkan anotasi @Path dan @GET untuk menentukan path dan method HTTP yang digunakan. Selain itu, kita juga menambahkan anotasi @Produces(MediaType.APPLICATION_XML) untuk menandakan bahwa output yang dihasilkan adalah dalam format XML.

Jalankan server seperti yang telah dijelaskan sebelumnya, kemudian coba panggil operasi sayHello dengan cara yang sama seperti sebelumnya. Jika semuanya berjalan dengan benar, maka output yang dihasilkan adalah sebagai berikut:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<greeting>
    <message>Hello, John!</message>
</greeting>
```

Perlu diperhatikan bahwa keluaran tersebut dalam format XML, dan kita dapat membaca dan memproses keluaran tersebut dengan mudah menggunakan library JAXB.

Web Service dapat menggunakan parameter untuk menerima input dari client. Ada beberapa cara yang dapat digunakan untuk mengirim parameter ke Web Service, seperti query parameter, path parameter, dan request body. Berikut adalah contoh penggunaan parameter pada Web Service di Java:

Contoh penggunaan query parameter:

```
 1. import javax.jws.WebService;
 2. import javax.ws.rs.*;
 3. import javax.ws.rs.core.MediaType;
 4.
 5. @WebService
 6. @Path("/hello")
 7. public class HelloWorld {
 8.
 9.     @GET
10.     @Path("/sayHello")
11.     @Produces(MediaType.TEXT_PLAIN)
12.     public String sayHello(@QueryParam("name") String name) {
13.         return "Hello, " + name + "!";
14.     }
15.
16.     public static void main(String[] args) {
17.         Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
18.         System.out.println("Web Service Started");
19.     }
20. }
21.
```

Pada contoh di atas, kita menggunakan query parameter dengan menggunakan anotasi @QueryParam("name"). Kita dapat mengakses parameter tersebut pada operasi sayHello menggunakan variabel name.

Jika kita menjalankan Web Service dan memanggil URL http://localhost:8080/hello/sayHello?name=John, maka output yang dihasilkan adalah Hello, John!.

Contoh penggunaan path parameter:

```
1. import javax.jws.WebService;
2. import javax.ws.rs.*;
3. import javax.ws.rs.core.MediaType;
4.
5. @WebService
6. @Path("/hello")
7. public class HelloWorld {
8.
9.     @GET
10.    @Path("/sayHello/{name}")
11.    @Produces(MediaType.TEXT_PLAIN)
12.    public String sayHello(@PathParam("name") String name) {
13.        return "Hello, " + name + "!";
14.    }
15.
16.    public static void main(String[] args) {
17.        Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
18.        System.out.println("Web Service Started");
19.    }
20. }
21.
```

Pada contoh di atas, kita menggunakan path parameter dengan menggunakan anotasi @PathParam("name"). Kita dapat mengakses parameter tersebut pada operasi sayHello menggunakan variabel name. Jika kita menjalankan Web Service dan memanggil URL http://localhost:8080/hello/sayHello/John, maka output yang dihasilkan adalah Hello, John!.

Contoh penggunaan request body:

```
1. import javax.jws.WebService;
2. import javax.ws.rs.*;
3. import javax.ws.rs.core.MediaType;
4.
5. @WebService
6. @Path("/hello")
7. public class HelloWorld {
8.
9.     @POST
10.    @Path("/sayHello")
11.    @Consumes(MediaType.APPLICATION_JSON)
12.    @Produces(MediaType.TEXT_PLAIN)
13.    public String sayHello(Person person) {
14.        return "Hello, " + person.getName() + "!";
15.    }
16.
17.    public static void main(String[] args) {
18.        Endpoint.publish("http://localhost:8080/hello", new HelloWorld());
```

```
        System.out.println("Web Service Started");
    }
}

class Person {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Pada contoh di atas, kita menggunakan request body dengan menggunakan anotasi @Consumes(MediaType.APPLICATION_JSON) untuk menandakan bahwa input yang diterima adalah dalam format JSON, dan anotasi @Produces(MediaType.TEXT_PLAIN) untuk menandakan bahwa output yang dihasilkan adalah dalam format teks. Kita juga membuat kelas Person untuk merepresentasikan input yang diterima pada operasi sayHello.

Jika kita menggunakan aplikasi seperti Postman atau cURL untuk mengirimkan, Java dapat memanggil Web Service yang dibuat dengan PHP menggunakan protokol standar seperti SOAP atau REST. Namun, untuk dapat memanggil Web Service PHP, kita harus mengetahui spesifikasi dari Web Service tersebut seperti URL, tipe data yang digunakan, dan parameter yang diperlukan.

Untuk memanggil Web Service PHP dengan protokol SOAP, kita dapat menggunakan kelas javax.xml.soap.* pada Java. Berikut adalah contoh kode untuk memanggil Web Service PHP dengan protokol SOAP pada Java:

```
import javax.xml.soap.*;

public class SoapClient {

    public static void main(String[] args) {

        try {
            // Membuat koneksi ke Web Service PHP
            SOAPConnectionFactory soapConnectionFactory =
SOAPConnectionFactory.newInstance();
            SOAPConnection soapConnection = soapConnectionFactory.createConnection();
            String url = "http://localhost:8000/soapserver.php";
            SOAPMessage soapResponse = soapConnection.call(createSOAPRequest(), url);

            // Memproses response dari Web Service PHP
            SOAPBody soapBody = soapResponse.getSOAPBody();
            NodeList nodeList = soapBody.getElementsByTagName("result");
            String result = nodeList.item(0).getTextContent();
            System.out.println(result);

            soapConnection.close();

        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
```

```
    }

    private static SOAPMessage createSOAPRequest() throws Exception {
        // Membuat request SOAP
        MessageFactory messageFactory = MessageFactory.newInstance();
        SOAPMessage soapMessage = messageFactory.createMessage();
        SOAPPart soapPart = soapMessage.getSOAPPart();

        // Menambahkan elemen-elemen pada request SOAP
        SOAPEnvelope envelope = soapPart.getEnvelope();
        envelope.addNamespaceDeclaration("ns", "http://example.com/soapserver");
        SOAPBody soapBody = envelope.getBody();
        SOAPElement soapElement = soapBody.addChildElement("sayHello", "ns");
        soapElement.addChildElement("name", "ns").addTextNode("John");

        soapMessage.saveChanges();

        return soapMessage;
    }
}

```

Pada contoh di atas, kita membuat koneksi ke Web Service PHP menggunakan kelas SOAPConnectionFactory dan SOAPConnection. Kita juga membuat request SOAP dengan menggunakan kelas MessageFactory, SOAPMessage, dan SOAPPart. Parameter untuk memanggil Web Service PHP ditentukan pada metode createSOAPRequest(), di mana kita menambahkan elemen-elemen pada request SOAP seperti nama operasi dan parameter yang diperlukan. Setelah request SOAP dibuat, kita memanggil Web Service PHP menggunakan SOAPConnection.call(). Kita juga memproses response dari Web Service PHP untuk mendapatkan hasil yang diinginkan.

Untuk memanggil Web Service PHP dengan protokol REST, kita dapat menggunakan kelas java.net.HttpURLConnection pada Java. Berikut adalah contoh kode untuk memanggil Web Service PHP dengan protokol REST pada Java:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Base64;

public class RestClient {

    public static void main(String[] args) {

        try {
            // Membuat koneksi ke Web Service PHP
            String url = "http://localhost:8000/restserver.php?name=John";
            URL obj = new URL(url);
            HttpURLConnection con = (HttpURLConnection) obj.openConnection();

            // Menentukan jenis metode dan authentikasi HTTP
            con.setRequestMethod("GET");
            String userCredentials = "username:password";
            String basicAuth = "Basic " + new String(Base64.getEncoder().encode
(userCredentials.getBytes()));
        // Menambahkan header HTTP
        con.setRequestProperty("Authorization", basicAuth);
```

```
24.
25.         // Memproses response dari Web Service PHP
26.         int responseCode = con.getResponseCode();
27.         BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
28.         String inputLine;
29.         StringBuffer response = new StringBuffer();
30.
31.         while ((inputLine = in.readLine()) != null) {
32.             response.append(inputLine);
33.         }
34.         in.close();
35.
36.         System.out.println(response.toString());
37.
38.     } catch (Exception e) {
39.         System.err.println(e.getMessage());
40.     }
41. }
42. }
43.
```

Pada contoh di atas, kita membuat koneksi ke Web Service PHP menggunakan kelas `URL` dan `HttpURLConnection`. Kita juga menentukan jenis metode dan authentikasi HTTP pada Web Service PHP. Parameter untuk memanggil Web Service PHP ditentukan pada URL, di mana kita menambahkan parameter pada query string seperti `?name=John`. Setelah Web Service PHP dipanggil, kita memproses response dari Web Service PHP untuk mendapatkan hasil yang diinginkan.

Perlu diingat bahwa cara memanggil Web Service PHP pada Java dapat berbeda-beda tergantung pada spesifikasi dari Web Service tersebut dan protokol yang digunakan. Oleh karena itu, sebaiknya kita mengetahui spesifikasi dari Web Service PHP terlebih dahulu sebelum memanggil Web Service tersebut pada Java.

Berikut adalah contoh koneksi ke REST API https://akademik.stmik-im.ac.id/

Contoh 1:

```
 1. import java.io.BufferedReader;
 2. import java.io.DataOutputStream;
 3. import java.io.InputStreamReader;
 4. import java.net.HttpURLConnection;
 5. import java.net.URL;
 6. import java.net.URLEncoder;
 7.
 8. public class APIClient {
 9.     public static void main(String[] args) {
10.         try {
11.             String apiUrl = "https://akademik.stmik-im.ac.id/api/login";
12.             URL url = new URL(apiUrl);
13.             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
14.
15.             // Atur metode permintaan
16.             connection.setRequestMethod("POST");
17.
18.             // Atur header permintaan
19.             connection.setRequestProperty("accept", "*/*");
20.             connection.setRequestProperty("Content-Type", "application/x-www-form-
urlencoded");
21.
22.             // Aktifkan output untuk mengirim data formulir
23.             connection.setDoOutput(true);
24.
25.             // Atur data formulir
```

```
26.             String formData = "username=" + URLEncoder.encode("36990031", "UTF-8") +
27.                               "&password=" + URLEncoder.encode("36990031", "UTF-8");
28.
29.             // Kirim data formulir
30.             try (DataOutputStream outputStream = new
DataOutputStream(connection.getOutputStream())) {
31.                 outputStream.writeBytes(formData);
32.                 outputStream.flush();
33.             }
34.
35.             // Baca respon dari API
36.             BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
37.             String line;
38.             StringBuilder response = new StringBuilder();
39.
40.             while ((line = reader.readLine()) != null) {
41.                 response.append(line);
42.             }
43.
44.             reader.close();
45.
46.             // Proses atau tampilkan data sesuai kebutuhan
47.             System.out.println("Response from API:\n" + response.toString());
48.
49.             // Tutup koneksi
50.             connection.disconnect();
51.         } catch (Exception e) {
52.             e.printStackTrace();
53.         }
54.     }
55. }
56.
57.
```

Contoh 2:

```
 1. import java.io.BufferedReader;
 2. import java.io.DataOutputStream;
 3. import java.io.InputStreamReader;
 4. import java.net.HttpURLConnection;
 5. import java.net.URL;
 6. import java.nio.charset.StandardCharsets;
 7.
 8. public class OrtuDns {
 9.
10.     // Fungsi untuk mendapatkan token API key
11.     public static String getApiKey(String username, String password) {
12.         try {
13.             // URL endpoint untuk mendapatkan token API key
14.             String loginUrl = "https://akademik.stmik-im.ac.id/api/login";
15.
16.             // Membuat objek URL
17.             URL url = new URL(loginUrl);
18.
19.             // Membuat objek HttpURLConnection
20.             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
21.
22.             // Menentukan metode request sebagai POST
23.             connection.setRequestMethod("POST");
24.
25.             // Mengaktifkan output stream untuk mengirim data
26.             connection.setDoOutput(true);
27.
28.             // Menyiapkan data login sebagai form-urlencoded
29.             String postData = "username=" + username + "&password=" + password;
30.             try (DataOutputStream wr = new DataOutputStream(connection.getOutputStream()))
{
```

```
31.                 wr.write(postData.getBytes(StandardCharsets.UTF_8));
32.             }
33.
34.             // Membaca response dari server
35.             try (BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()))) {
36.                 String line;
37.                 StringBuilder response = new StringBuilder();
38.                 while ((line = in.readLine()) != null) {
39.                     response.append(line);
40.                 }
41.                 // Mengembalikan token API key
42.                 // return response.toString();
43.                 // Mengambil nilai "JWT" dari respons JSON secara manual
44.                 String jsonResponse = response.toString();
45.                 int startIndex = jsonResponse.indexOf("\"JWT\":\"") + 7;
46.                 int endIndex = jsonResponse.indexOf("\"}", startIndex);
47.                 return jsonResponse.substring(startIndex, endIndex);
48.
49.
50.             }
51.         } catch (Exception e) {
52.             e.printStackTrace();
53.         }
54.         return null;
55.     }
56.
57.     // Fungsi untuk melakukan request GET dengan token API key
58.     public static void executeGetRequest(String apiKey) {
59.         try {
60.             // URL endpoint untuk request GET
61.             String apiUrl = "https://akademik.stmik-im.ac.id/api/list/OrtuDns";
62.
63.             // Membuat objek URL
64.             URL url = new URL(apiUrl);
65.
66.             // Membuat objek HttpURLConnection
67.             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
68.
69.             // Menentukan metode request sebagai GET
70.             connection.setRequestMethod("GET");
71.
72.             // Menambahkan header X-Authorization dengan token API key
73.             connection.setRequestProperty("X-Authorization", apiKey);
74.
75.             // Membaca response dari server
76.             try (BufferedReader in = new BufferedReader(new
InputStreamReader(connection.getInputStream()))) {
77.                 String line;
78.                 StringBuilder response = new StringBuilder();
79.                 while ((line = in.readLine()) != null) {
80.                     response.append(line);
81.                 }
82.                 // Menampilkan response dari request GET
83.                 System.out.println("Response from GET request: " + response.toString());
84.             }
85.
86.             // Menutup koneksi
87.             connection.disconnect();
88.         } catch (Exception e) {
89.             e.printStackTrace();
90.         }
91.     }
92.
93.     public static void main(String[] args) {
94.         try {
95.             // Data login (ganti sesuai kebutuhan)
96.             String username = "360663001";
97.             String password = "digimon";
```

```
 98.
 99.             // Mendapatkan token API key dengan fungsi baru
100.             String apiKey = getApiKey(username, password);
101.
102.             // Menampilkan token API key
103.             System.out.println("Token API Key: " + apiKey);
104.
105.             // Melanjutkan dengan request GET menggunakan apiKey
106.             executeGetRequest(apiKey);
107.         } catch (Exception e) {
108.             e.printStackTrace();
109.         }
110.     }
111. }
112.
```

Tugas Rumah :

Buat program untuk mengexplore REST API di https://akademik.stmik-im.ac.id/ seperti menampilkan KRS, KHS dan lain lain .