Service-based Architecture Style

Requirement

The electronic recycling system used to recycle old electronic devices (such as an iPhone or Galaxy cell phone). The processing flow of recycling old electronic devices works as follows:

- First, the customer asks the company (via a website) how much money they can get for the old electronic device (called quoting). If satisfied, the customer will send the electronic device to the recycling company,
- The recycling company in turn will receive the physical device (called receiving).
- Once received, the recycling company will then assess the device to determine if the device is in good working condition or not (called assessment).
- If the device is in good working condition, the company will send the customer the money promised for the device (called accounting).
- Through this process, the customer can go to the website at any time to check on the status of the item (called item status).
- Based on the assessment, the device is then recycled by either safely destroying it or reselling it (called recycling).
- Finally, the company periodically runs ad hoc and scheduled financial and operational reports based on recycling activity (called reporting).

Architecture design of this system was given by **Figure 1**.

Assume that we need to create 1 projects for each group of services. The services was built based on springboot technology, the user interface was built using the ***react*** technology. Database in this case use MariaDB.

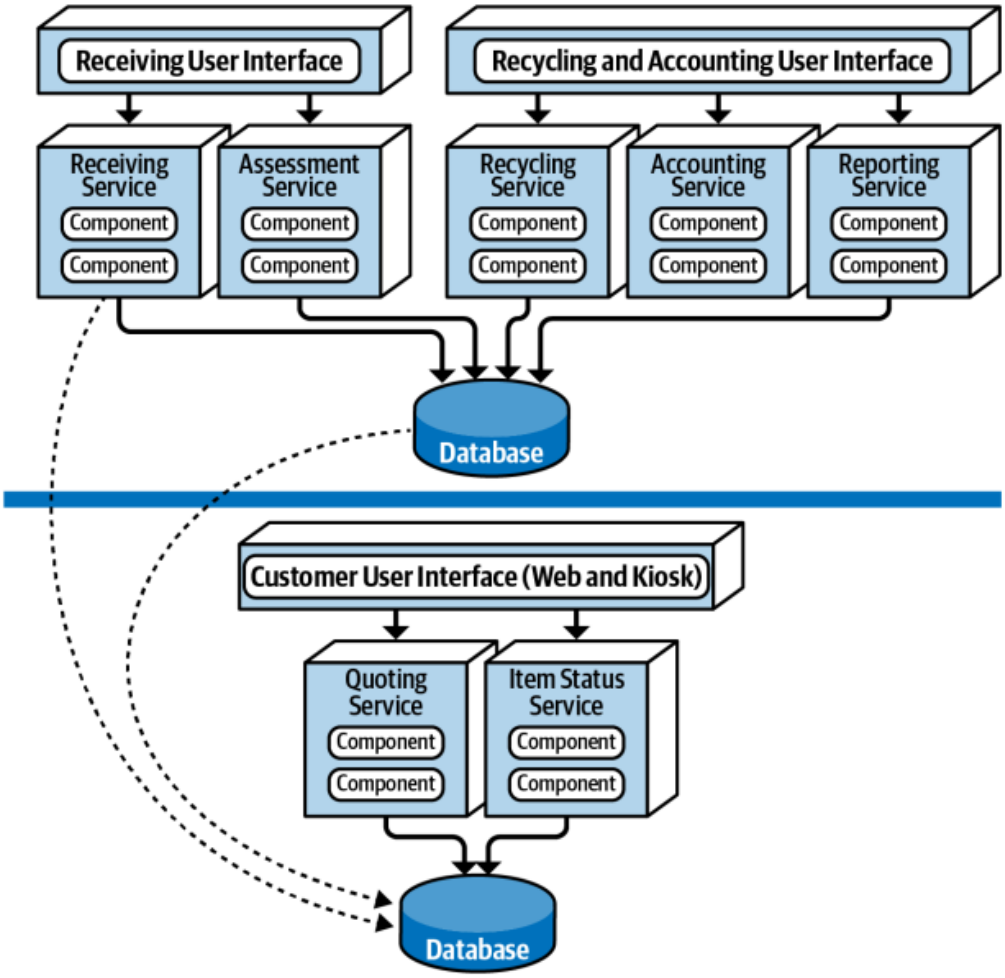
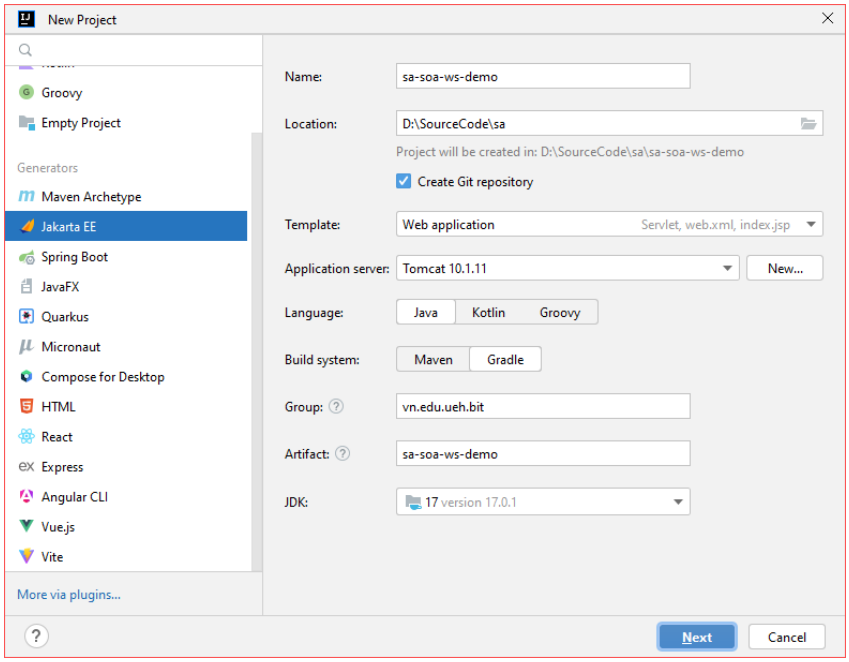
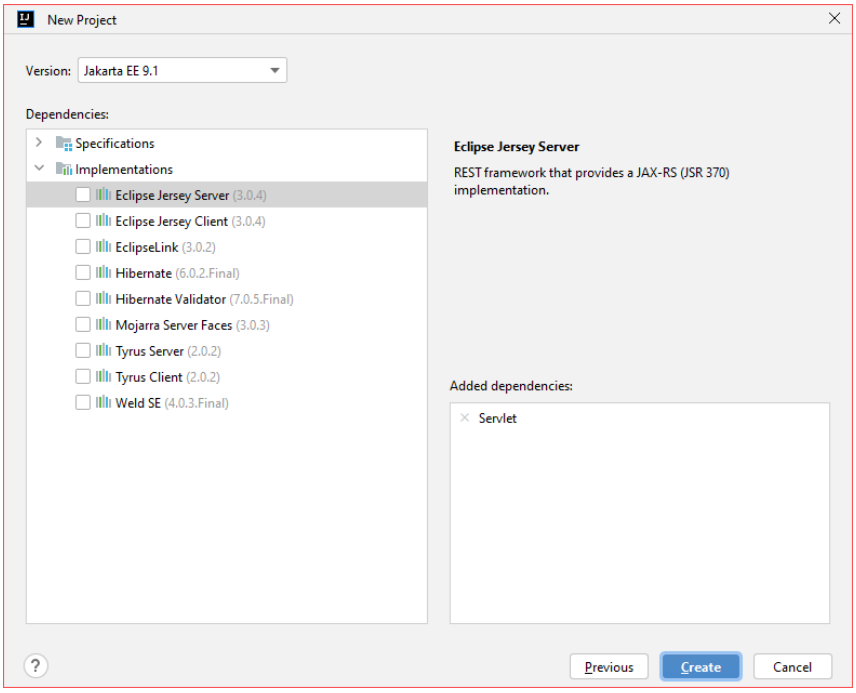


***Figure 1**: Electronics recycling example using service-based architecture.*

SOAP Web Service Example

1. Create New Project





2. Add dependencies as follow:

```
dependencies {
    compileOnly('jakarta.servlet:jakarta.servlet-api:5.0.0')
    // This dependency provides the Jakarta API for Java API for XML Web Services
(JAX-WS).
    implementation 'jakarta.jws:jakarta.jws-api:3.0.0'
    //This dependency includes the JAX-WS Reference Implementation (RI) from Oracle.
    implementation 'com.sun.xml.ws:jaxws-ri:4.0.2'
```

```
    //This dependency includes the JAX-WS runtime libraries.
    implementation 'com.sun.xml.ws:jaxws-rt:4.0.2'

    testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")
}
```

3. Add the interface and implement class

```
package vn.edu.ueh.bit.services;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

@WebService
public interface CalculatorWs {
    @WebMethod
    int add(int num1, int num2);
}
```

```
package vn.edu.ueh.bit.services;

import jakarta.jws.WebMethod;
import jakarta.jws.WebService;

@WebService
public class CalculatorWsImpl implements CalculatorWs {
    @WebMethod
    public int add(int num1, int num2) {
        return num1 + num2;
    }
}
```

4. Config web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
         version="5.0">
    <display-name>SOAPWithJaxWS</display-name>
    <listener>
        <listener-class>
            com.sun.xml.ws.transport.http.servlet.WSServletContextListener
        </listener-class>
    </listener>
    <servlet>
        <servlet-name>CalculatorWebService</servlet-name>
        <servlet-class>
            com.sun.xml.ws.transport.http.servlet.WSServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>CalculatorWebService</servlet-name>
        <url-pattern>/calculatorWebService</url-pattern>
    </servlet-mapping>
</web-app>
```

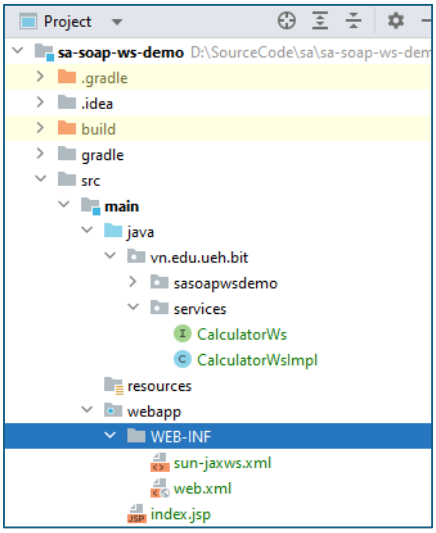
5. Create sun-jaxws.xml

The sun-jaxws.xml file is used to configure the endpoint for your SOAP web service. This file specifies details about the web service endpoint, such as the implementation class and the URL pattern.

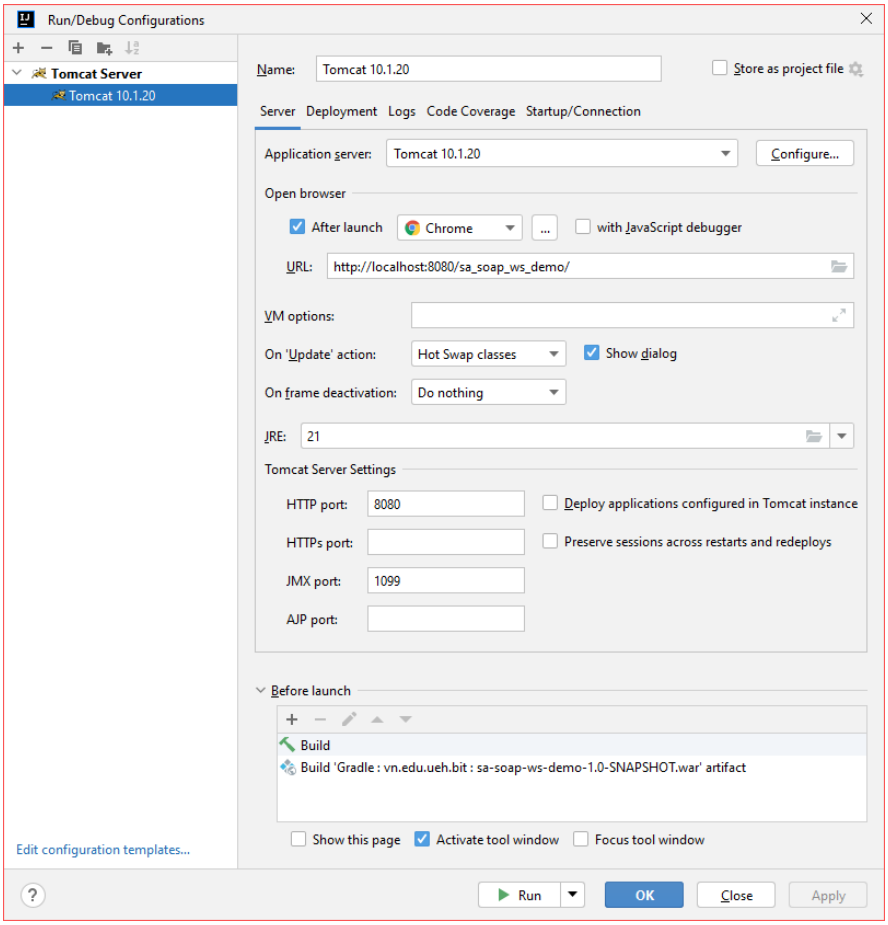
```xml
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
    <endpoint
            implementation="vn.edu.ueh.bit.services.CalculatorWsImpl"
            name="CalculatorWebService"
            url-pattern="/calculatorWebService" />
</endpoints>
```

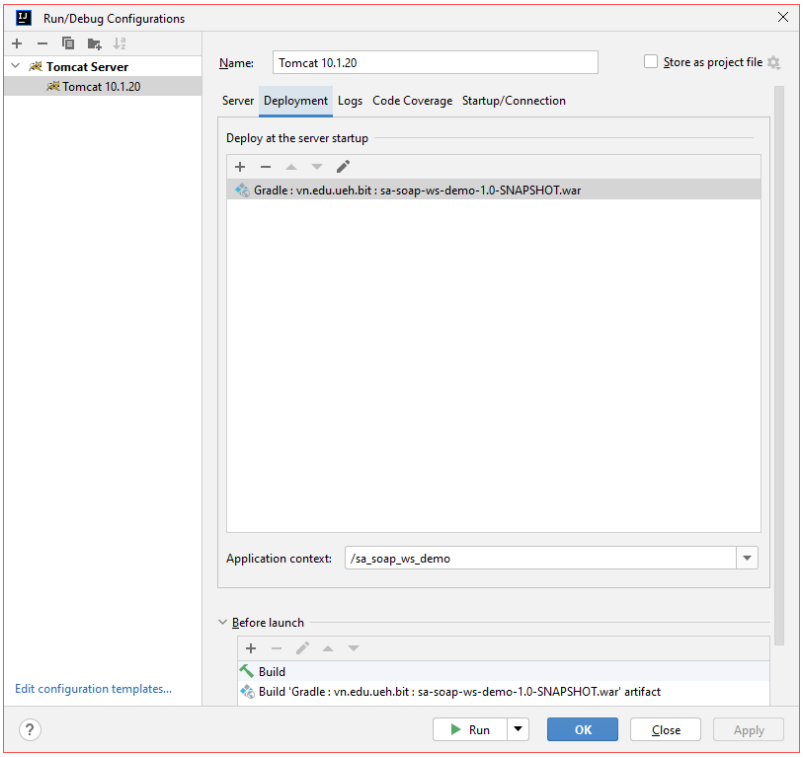
You may add other endpoints to this file.

6. The structure of project



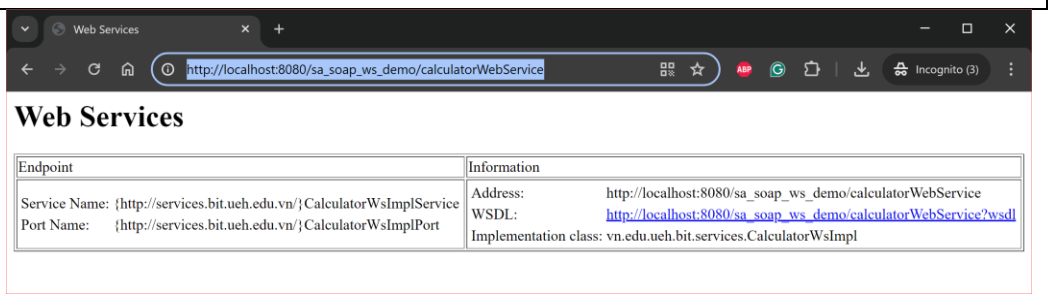
7. Configure your web server:



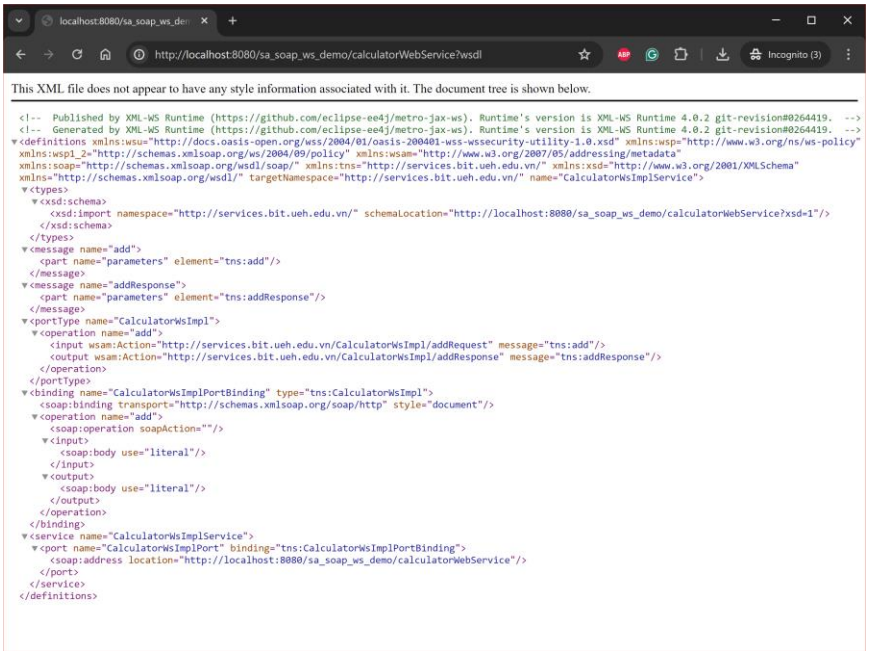


8. Access endpoint

http://localhost:8080/sa_soap_ws_demo/calculatorWebService



Address: http://localhost:8080/sa_soap_ws_demo/calculatorWebService?wsdl



9. Client

```
import jakarta.xml.ws.Service;
import vn.edu.teo.services.CalculatorWs;

import javax.xml.namespace.QName;
import java.net.URL;
import java.util.Iterator;

public class Ws_Client {
    public static void main(String[] args) throws Exception {
        // Create URL of .wsdl file
        URL wsdlURL = new
URL("http://localhost:8080/sa_soap_ws_demo/calculatorWebService?wsdl");

        // Create a QName using targetNamespace and name
        QName qname = new QName("http://services.bit.ueh.edu.vn/",
"CalculatorWsImplService");

        // Creates a Service instance with the specified WSDL document location and
        // service qualified name
        Service service = Service.create(wsdlURL, qname);

        // We need to pass interface and model beans to client
        Iterator<QName> ports = service.getPorts();
        ports.forEachRemaining(System.out::println);

        QName portName = new QName("http:// services.bit.ueh.edu.vn/",
"CalculatorWsImplPort");

        CalculatorWs userService = service.getPort(portName,CalculatorWs.class);

        int a = userService.add(3, 4);
        System.out.println(a);
    }
}
```

Applied exercise.

Apply SOA architecture to create database connection services and then perform CRUD on any table.

Springboot REST web service example

Assume that that we need create customer service.

The customer information should be created using the following properties: *customer id*, *customer full name*, *email*, ***address***, *phone number*, (***bank account number and bank name***).