Note: No portfolio task for WS labs (Labs 9 and 10). These are for your interests to learn.

Software
To finish the lab, you may need the following software:
1. NetBeans IDE version 12.2 or 15.2
2. JDK version 1.8.0 update 121 or later
3. GlassFish Server Open Source Edition version 5.1.0 [JavaDB is the database server that comes with GlassFish.]

Aim
Program a RESTful Web Services sitting on the Web Server and program a client application that interacts with the relevant RESTful Web Services

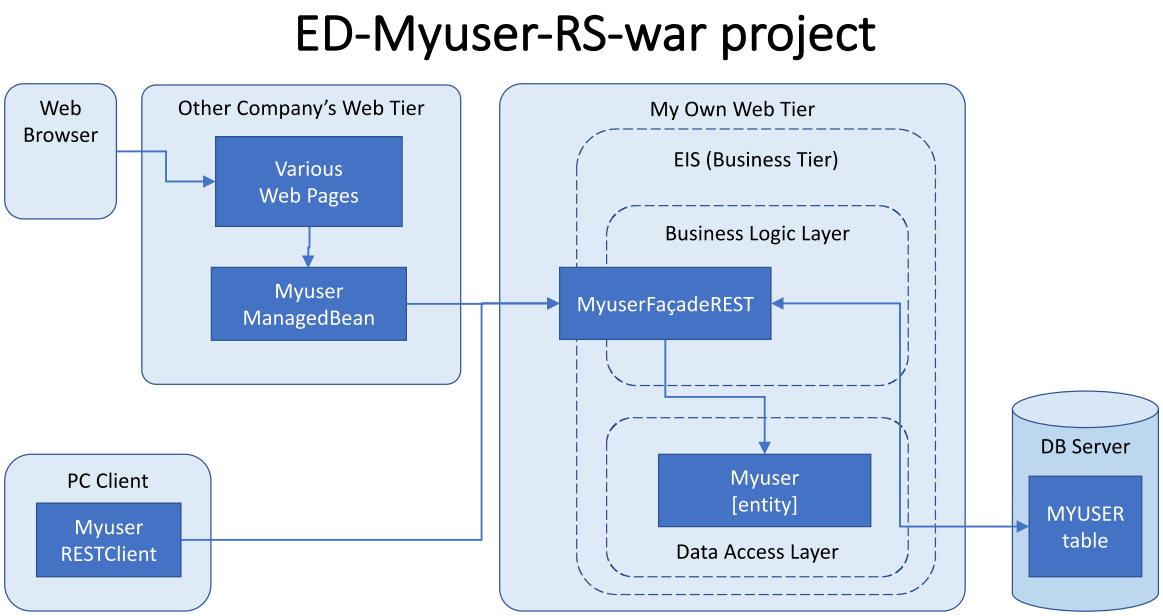

Figure 1 A rough architecture diagram of ED-Myuser-RS-war Web Application

Overview of the RESTful Web Application
In this Lab, we redo the ED-JEE-DTO project done in Labs 04 and 05 as a RESTful Web Services Web Application.

Side Note: Originally, I planned to extend the capability of what we have in Labs 04 and 05 so that the Stateless Session Bean (MyuserFacade) can expose the RESTful Web Services, just like Lab_09_WS. However, this does not work properly after a week's trial. Also, it seems that the RESTful WS, as a war project, could not integrate will with the ejb project. The only integration is to have "@Stateless" in the RESTful WS java file, making it a software module on Web tier and a stateless EJB in the EIS tier. So, we need to build this from scratch. It turns out that it is much easier than I thought. The downside of this is that we have to put everything (i.e. Business Logic Layer and Data Access Layer) to the Web Server, note the architecture diagram in Figure 1. So, there is not much flexibility. This violates certain "design" principles and also the Web server handles all the business logics now. Probably, that may be a reason why the "deployment" architecture of GlassFish server (starting from version 4) changed to what it is now.

PreLab Tasks and Points to Note
Assume that you have completed Lab_02, giving you "ED-JDBC", and Lab_04 and Lab_05, giving you "ED-JEE-DTO" (including an entity class Myuser and a database table MYUSER).

1. Since this Lab will be using the database table MYUSER, you need to make sure that no other enterprise application is using this MYUSER table when you do this Lab. So, start your GlassFish server, "undeploy" any enterprise project that uses this MYUSER table, stop the GlassFish server. Also stop the JavaDB server (GlassFish server starts JavaDB server automatically). Do not keep GlassFish server and JavaDB server running when doing this Lab. It will save you some trouble. When it is time to "start" the GlassFish server, I will let you know.
2. In the middle of the Lab, we will create the "Myuser" entity class. DO NOT copy it from any of your existing projects. It won't work! Again, follow the steps to create the "Myuser" entity class. You will be safe.
3. Also, if you have changed the MYUSER database table, you can rerun the "ED-JDBC" project to recreate the table and reload some data into the table, again assuming you have not changed "ED-JDBC" project. If you do, please fix it yourself. ☺ After doing this, remember to stop the JavaDB server. Again, it will save you some trouble.

Overview of Lab Tasks

In this Lab, you will learn to perform the following tasks in programming a web application using RESTful web services and a client program to request the services provided:

LT1. Create a web application project
LT2. Add an entity class in the web application project
LT3. Create a RESTful web services in the web application project
LT4. Deploy the web application
LT5. Create a test client to test the RESTful web services
LT6. Run the test client
LT7. Create a Java application project
LT8. Create a RESTful client to request the RESTful web services
LT9. Add a Java class to store the information from the database
LT10. Complete the RESTful client
LT11. Run the RESTful Java client

Assumption

Assume that you have the MYUSER database table in "sun-appserv-samples", all enterprise applications that use the MYUSER table should be "undeployed", and, finally, both JavaDB and GlassFish server have been stopped.

Lab Tasks

This Lab should be run on MS Windows Platform

LT1. Create a new Web Application Project in NetBeans called "`ED-Myuser-RS-WAR`"
- LT1.1 Select "File" > "New Project"
- LT1.2 Choose "Java Web" > "Web Application", then click "Next"
- LT1.3 Enter "ED-Myuser-RS-WAR" in the Project Name field, then click "Next"
- LT1.4 Use the default server "GlassFish" and Java EE version "Java EE 7 Web", then click "Next"
- LT1.5 Select "JavaServer Faces" in the Frameworks section
- LT1.6 Click "Finish"
- *Note: NetBeans will then create the project for you. The "index.xhtml" will be opened in the editor window. We do not need it for the time being.

LT2. Add the Entity class "Myuser" to the Web Application Project
- LT2.1 Select the "ED-Myuser-RS-WAR" project
- LT2.2 Right click the mouse and select "New" > "Entity Class from Database…"
- LT2.3 In the "New Entity Classes from Database" window, select "jdbc/__default" in the Data Source field and put the "MYUSER" table in the "Selected Tables" text area (Make sure the "Include Related Tables" check box is checked)
  - Note: If this is the first time you do this, NetBeans will try to get the connection to the database. It will display the "New Connection Wizard" window, you then need to check the following settings:

```
Driver Name:    Java DB (Network)
Host:           localhost
Port:           1527
User Name:      APP
Password:       ●●● (should be APP anyway)
JDBC URL:       jdbc:derby://localhost:1527/sun-appserv-samples;create=true
```

After checking these, try click on "Test Connection". Result should be "Connection Succeeded" – this is to make sure things can be done properly. If there is any connection problem, probably you need to sort it out first (e.g. database server has not been started or database connection strings while setting up the database in Lab_1a is not correct). There are so many reasons to go wrong, please do let me know and I will try my best to help; and then enrich this "Note".

In case, the "Test Connection" is successful. Then, Click "Next". You should select schema "APP" (already been pre-selected for you by NetBeans). NetBeans then displayed the connection name as "…" (Sorry could not remember these "…", next time when I do this, I will remember to fill it in. Probably it will be next time I teach Ent Dev. ☺ I do not want to do the following, because it is too scary and too risky for me. If I could not make it work, I may need to reinstall my GlassFish server and re-run all my examples and the labs to make sure things are working fine).

Warning: Risky information (Don't try): After that, NetBeans will put the above information into the file `"${GLASSFISH_ROOT}/glassfish/domains/domain1/config/domain.xml"` using "jdbc-resource" and "jdbc-connection-pool" tags. Do not try to modify this file unless you are sure what to do because it is **risky**. If you want to try, make sure you have a backup copy of the "domain.xml" file to restore the server to its previous settings.

Note: If you cannot find the "MYUSER" table in the "Available Tables:" field, make sure your database connection string (See JDBC URL) is right and you have set up the database properly with the proper username "APP" and password "APP". You may want to remove the database and recreate one with the right settings. See Lab_01a_Setup_JavaEE.

Note: When asked about the database username and password, enter "APP" for both (or, whatever you put during the setting up of your own databases).

LT2.4 Click "Next"

Note: A new dialog box appears with "MYUSER" in the "Class Names" field and "ED-Myuser-RS-WAR" in the Project field

LT2.5 Enter "entity" in the "Package" field

LT2.6 Make sure that the following check boxes are checked

a. "Generate Named Query Annotations for Persistent Fields"

Note: By checking the box, NetBeans will create some standard named queries like

"Myuser.findAll"	to find all myusers in the database table
"Myuser.findByUserid"	to find a particular myuser using userid (userid is the primary key of the table)

… (others omitted) …

b. "Generate JAXB Annotations"

Note: By checking the box, NetBeans will create some standard annotations that allow the entity class to be used via JAXB web services. We now need this to help make it work. JAXB is about XML Binding. It automatically "converts" XML contents in the HTTP protocol to Java objects and vice versa.

c. "Create Persistence Unit"

LT2.7 Click "Finish"

Note: NetBeans will automatically generate the entity POJO "Myuser.java" for you, including some named queries.

Note: This entity class is a Data Access Object (DAO) that is responsible for holding the required information for the records in the MYUSER database table.

Note: In the "Configuration Files" tag, you can see "META-INF" and "persistence.xml". This "persistence.xml file" contains the information of Myuser class, in xml format, and acts as a configuration file.

LT2.8 This task will fix some "unknown" bug in NetBeans

a. Expand on "Configuration Files…"
b. Double click on "persistence.xml"
c. In the "persistence.xml" pane, uncheck "Include All Entity …"
d. Click on "Add Classes…"
e. In the "Add Entity Class" window, select "entity.Myuser" and click "OK"
f. Click "Save"

Qn: If you still remember, this is exactly the same as Lab 04 before. Why can't we copy the previous code to here? It saves us lots of trouble.

Ans: Well, I lie. Yes, it is possible to copy and paste the "Myuser" to here and changes the "persistence.xml" file accordingly. But, if you make some changes on that "Myuser" class, it may not work. Or, if you enter the wrong information to the "persistence.xml" file, I cannot help as well. Since I have no prior knowledge of what you have done individually, it is difficult for me to fix things if it does not work out the way I want it to work. So, I lie for consistency reasons and for the purpose of having every student on the same page. Hope you understand my reason behind this. Sorry!

LT2.9 Add some annotations in "Myuser" class to make the JAXB work for us (JAXB is the automatic binding services provided by JavaEE to bind the Java object to and from XML file for web communications, so no extra coding to convert Java object to XML and vice versa)

a. Open "Myuser.java"
b. Change "@XmlRootElement" to "`@XmlRootElement(name = "Myuser")`"
c. Add "`@XmlAccessorType(XmlAccessType.FIELD)`" just below the "@XmlRootElement(…)" line

Note: After (b) and (c), it looks like the following. By the way, do not use "Edge" in Windows 10 to do the copy and paste. It sucks! Use Acrobat reader. It will safe you some trouble. This advice is from a student. Since I am using a Mac, I have no idea about this. The copy and paste from Preview (Mac's pdf reader) works fine for me. All the lines break in the right position after my copy and paste actions.

```
@Entity
@Table(name = "MYUSER")
@XmlRootElement(name = "Myuser")
@XmlAccessorType(XmlAccessType.FIELD)
...
```

d. Add "@XmlElement(required=true)" just before **each** instance variable declaration. For example, you will have something like the following for the instance variable "userid". Do this for every one of them (instance variable only).

```
@Column(name = "USERID")
@XmlElement(required = true)
private String userid;
```

e. Remember to "Fix Imports", "Format the code" and "Save the file"

LT3. Create the RESTful web services for Myuser entity class

LT3.1 Select the "ED-Myuser-RS-WAR" project

LT3.2 Right click the mouse and select "New" > "RESTful Web Services from Entity Classes…"

Note: Recall that if you could not find this in the context menu, you need to choose "Other…" > "Web Services" in the "Categories" > "RESTful Web Services from Entity Classes" in "File Types"

LT3.3 In the "New RESTful Web Services from Entity Classes" window, select "entity.Myuser" in the "Available Entity Classes" text area and put it in the "Selected Entity Classes" text area

LT3.4 Click "Next"

Note: A new dialog box appears with "service" in the "Resource Package:" field

LT3.5 Click "Finish"

Note: NetBeans then creates the source codes and config files for the RESTful web services.

LT3.6 Expand on the "service" package

Note: You will see three java files there. They are

a. AbstractFacade.java – An abstract class providing DB access methods [We did not use this feature in Labs 04 and 05, because it will create lots of unnecessary confusion, especially you are not good at "inheritance"]

b. ApplicationConfig.java – this is to set up the relevant classes and "hooks" for our uses

c. MyuserFacadeREST.java – This is basically the Stateless Session Bean (MyuserFacade) combined with the RESTful web services code.

Note: If you open this file, you will see "@Stateless" annotation for SLSB and "@Path("entity.myuser")" annotation for RESTful web services. So, being a Stateless SB, you can expect the usual bean instance pooling and SLSB life-cycle management are there in the GlassFish server. So, scalability has been addressed.

Note: My guess is that, if you put "@DeclareRoles" in the class level and "@RolesAllowed" in the method level, you address certain security issues and concerns. I have not tried it. Try it yourself.

LT3.7 Fix message format issue in "MyuserFacadeREST.java"

a. Open "MyuserFacadeREST.java" in the editor tab

b. Remove "MediaType.APPLICATION_JSON" from the "@Consumes" and "@Produces" annotations. Do this for every method that has the "MediaType.APPLICATION_JSON". For example, you will have something like the following for the "create()" method after removing the "MediaType.APPLICATION_JSON".

```
@POST
@Override
@Consumes({MediaType.APPLICATION_XML})
public void create(Myuser entity) {
...
```

Note: The "@Consumes" annotation specifies the message formats that can be accepted by this method in the web service. So, if the client wants to call this method, it needs to communicate with this format. The server will then take the message with this format, "consume" it and provide the required Java object (Myuser in this case) or parameter for the programmer to use.

Note: The "@Produces" annotation specifies the message formats that will be produced and sent to client for communication.

Note: Although NetBeans generates this "JSON" type and Java EE Tutorial has some "code snippets" to show that it should work, I could not find any "working examples" in their GitHub repository. So, I guess it is still not working properly at the moment. Hence, I decided to remove it rather than leave it there. I do not mind you try it to see whether it works. Let me know if it works.

Note: If you want to try the "JSON" message format, you can put back the "MediaType.APPLICATION_JSON" in the "@Consumes" or "@Produces" annotations and test the message format with your client. We will do the client program in later LTs.

c. Remember to save the file

LT4. Deploy the Web Application "ED-Myuser-RS-WAR" project (Recap)

LT4.1 Select "ED-Myuser-RS-WAR" web application project in the "Projects" window

LT4.2 Right click the mouse and select "Deploy" (You may want to select "Clean and Build" first to clean up any mess that has been done.)

LT5. Create a test client to test the RESTful web services

LT5.1 Select "ED-Myuser-RS-WAR" web application project in the "Projects" window

LT5.2 Right click the mouse and select "Test RESTful Web Services"

LT5.3 In the "Configure REST Test Client" window, select "Web Test Client in Project" and make sure it is "ED-Myuser-RS-WAR" project (via the "Browse" button)

LT5.4 Click "OK"

Note: Netbeans will then generate the "test client" automatically and run it in a web browser.

Note: NetBeans generated many files in the Web Pages tag. Expand the "Web Pages" tag, you will see these files. The file "test-resbeans.html" is the one that you run to test the RESTful web services.

LT6. Run the test client generated by NetBeans (NetBeans already run it if you follow the instructions in LT5). Alternatively, you can follow the instructions below. Make sure you have generated a test client (run LT5) and deployed ED-Myuser-RS-WAR beforehand.

LT6.1 Select "ED-Myuser-RS-WAR" web application project in the "Projects" window

LT6.2 Right click the mouse and select "Run"

LT6.3 In the browser window, type in the following url

http://localhost:8080/ED-Myuser-RS-WAR/test-resbeans.html

Note: You will then see "Test RESTful Web Services" and "ED-Myuser-RS-WAR" in your browser.

LT6.4 Expand on "entity.myuser" and select "{id}"

Note: On the right pane, the "GET (application/xml)" method has been selected.

LT6.5 Enter "000001" in "Id:" text field and click the "Test" button

Note: You will see something like the following. It is the information stored on the MYUSER table with userid = "000001" in your database server.

```
<?xml version="1.0" encoding="UTF-8"?>
   <Myuser>
       <userid>000001</userid>
       <name>Edmonds Lau</name>
       <password>123456</password>
       <email>elau@swin.edu.au</email>
       <phone>9876543210</phone>
       <address>Swinburne EN510a</address>
       <secqn>What is my name?</secqn>
       <secans>Edmonds</secans>
   </Myuser>
```

LT6.6 Try other methods - but remember to use this "xml" format for your own data to perform the CRUD operations – "Create" (@POST), "Review" (@GET), "Update" (@PUT) and "Delete" (@DELETE).

In the next two tasks, we are going to program a normal Java application to act as a client for the RESTful Web Services.

LT7. Create a normal Java Application project called "ED-Myuser-RS-WAR-Client"
- LT7.1 Select "File" > "New Project"
- LT7.2 Choose "Java" > "Java Application", then click "Next"
- LT7.3 Enter "ED-Myuser-RS-WAR-Client" in the Project Name field
- LT7.4 Uncheck the "Create Main Class" check box (We will create it later)
- LT7.5 Click "Finish"
- Note: NetBeans will create the project folder.

LT8. Program a desktop client to access the RESTful Web Services
- LT8.1 Select the "ED-Myuser-RS-WAR-Client" project
- LT8.2 Right click the mouse and select "New" > "RESTful Java Client…"
- LT8.3 In the "New RESTful Java Client" window,
  a. Enter "MyuserRESTClient" in the "Class Name" field
  b. Enter "ed.rest" in the Package field
  c. Select "From Project" radio button in the "Select the REST resource" section
  d. Click "Browse" button
  e. In the "Available REST Resources" window
     1. Expand on "ED-Myuser-RS-WAR"
     2. Select "MyuserFacadeREST [entity.myuser]"
     3. Click "OK"

     Note: NetBeans now put "MyuserFacadeREST [entity.myuser]" in the "REST Resource Name" field
  f. Click "Finish"

     Note: NetBeans now generates the code for client side. This is only the skeleton code for us to use. It is quite "extensive" and there is a lot of "chaining method calls". Do not worry at the moment. You can find the discussions in Java EE Tutorial later when you have time to go through them properly.
- Note: We have not done the "public static void main()" for this client. Hence, it is incomplete at the moment. However, we will leave this class as is for the time being. We will come back to this later.

In the next LT, we need a Java class to hold our information. Remember, using Web Services, everything is communicated via HTTP. It does not matter whether you are using XML or JSON. So, everything is "text" based. We need a way to convert our Myuser Java object to "text" and send it via HTTP to the Web Services (normal WS or RESTful WS) for processing; and convert the "text" responses from the Web Services back to our Myuser Java object for us to do our normal programming. Now, since we could not use DTO (actually, there is no DTO anymore). I call this Myuser Java class on the client side as "**Myuser_ClientSide.java**" to signal that it is a client side object. I can call it "Myuser", but then it will be very confusing because the two "Myuse" classes will have different methods and different annotations etc.

LT9. Program the "Myuser_ClientSide" Java class to handle the information
- LT9.1 Select the "ED-Myuser-RS-WAR-Client" project
- LT9.2 Right click the mouse and select "New" > "Java Class…"
- LT9.3 In the "New Java Class" window,
  a. Enter "Myuser_ClientSide" in the "Class Name" field
  b. Enter "data" in the Package field (I used "data" instead of "entity" – to tell the differences between this client side "data.Myuser_ClientSide" and the server side "entity.Myuser" objects
  c. Click "Finish"

     Note: NetBeans creates the "Myuser_ClientSide.java" file and displays it in the editor tab.

LT9.4 Add the first two lines of the following code segment in front of the "Myuser_ClientSide" class. So, it reads like

```
@XmlRootElement(name = "Myuser")
@XmlAccessorType(XmlAccessType.FIELD)
public class Myuser_ClientSide {
```

Remember to "fix …", "format …" and "save …"

LT9.5 Copy and paste the following code segment to the "Myuser_ClientSide" class

```
private String userid;
private String name;
private String password;
private String email;
private String phone;
private String address;
private String secqn;
private String secans;
```

Remember to "fix …", "format …" and "save …"

LT9.6 Add the annotation "@XmlElement(required = true)" just **before** each individual instance variable. For example, you will have something like the following for the instance variable "userid". Do this for every one of them (instance variable only).

```
@XmlElement(required = true)
private String userid;
```

LT9.7 Generate an empty constructor (a no argument constructor) for "Myuser_ClientSide"

LT9.8 Generate the getters and setters for all instance variables for "Myuser_ClientSide"

LT9.9 Copy and paste the following code segment to the "Myuser_ClientSide" class (this method just display the information to the console, no fancy code here)

```
public void displayAllInfo() {
    System.out.println("User id\t\t: " + userid);
    System.out.println("Name\t\t: " + name);
    System.out.println("Password\t: " + password);
    System.out.println("Email\t\t: " + email);
    System.out.println("Phone\t\t: " + phone);
    System.out.println("Address\t\t: " + address);
    System.out.println("Security Qn\t: " + secqn);
    System.out.println("Security Ans\t: " + secans);
}
```

LT9.10 Remember to fix import, …, and save the file

LT10. Now, we are ready to program the main method in the RESTful client

LT10.1 Go to "MyuserRESTClient.java"

LT10.2 Copy and paste the following code segment

```
public static void main(String[] args) {
    MyuserRESTClient client = new MyuserRESTClient();

    Myuser_ClientSide myuser099 = new Myuser_ClientSide();
    myuser099.setUserid("000099");
    myuser099.setName("George Swinburne");
    myuser099.setPassword("George");
    myuser099.setEmail("gs@swin.edu.au");
    myuser099.setPhone("0392145678");
    myuser099.setAddress("Swinburne GS Building");
    myuser099.setSecqn("Who is George Swinburne?");
    myuser099.setSecans("Founder of Eastern Suburbs Technical College");

    restClient.create(myuser099);
    restClient.displayUser(myuser099.getUserid());
}

public void displayUser(String userid) {
    Myuser_ClientSide myuser = this.find(Myuser_ClientSide.class, userid);
    if (myuser == null) {
```

```
            System.out.println("No such user whose id is " + userid);
        } else {
            myuser.displayAllInfo();
        }
}
```

LT10.3 Remember to save the file

LT11. Run the client project (Remember to set the main class as "ed.rest.MyuserRESTClient", asked by NetBeans; If not, follow the steps below)

LT11.1 Select the project "ED-Myuser-RS-WAR-Client"

LT11.2 Right click the mouse and select "Properties"

LT11.3 In the "Project Properties" window

a. Click "Run" under the "Categories" pane

b. Select "ed.rest.MyuserRESTClient" as the "Main Class" using the "Browse…" button

c. Click "OK" until you are back to the project window

LT11.4 Run the project (Right click and select "Run")

LT11.5 Remember to check the database table

Finally, Take several deep breaths and relax.

Further Discussions on Interest

Question: How about a web application to access the RESTful Web Services?

Answer: To be developed; interested parties can try – just put the Java client code in a ManagedBean in your Web server using one IP address (say, 192.168.1.100) and point it to the BASE_URI of ED-Myuser-RS-WAR in another Web server using a different IP address (say 192.168.1.101) [The base uri will then be http://192.168.1.101:8080/ED-Myuser-RS-WAR/webresources]

Question: Why do we need two different Web Servers? Why not using Javascript on the browser to send the RESTful requests?

Answer: If you want to know how to do this using Javascript, you can look at the "test client" code generated by NetBeans, especially the "test-resbeans.js". You can learn from it.

Question: If not using browser scripting / programming technology, we still can do this using one web server, by making our software modules communicate to each other using Web Services. Why not showing us this?

Answer: This is a bad example / usage scenario. Under the traditional programming model, software modules within the same Web Server can be called **locally**, hence no need to "package" things in HTTP and "unpackage" it back to Java objects or any objects. It is a waste of resources, computation time and network bandwidth. Doing this violates the principle of building "scalable" software!!!

Back to the key question (How to make a RESTful Java client call the RESTful web services in the same web server), all you need to do is to copy and paste the Restful Java client code to a ManagedBean class. The ManagedBean object, once instantiated by the web server, will then act as a RESTful Java client to send the request to and receive the response from the RESTful web services in the same web server. Try it yourself. Not recommended though.

If you have other ideas, please do let me know via email. ☺