Don’t Print This Page.

|  |  |  |  |
| --- | --- | --- | --- |
| Developed By | Date | Approved By | Course Type (New/Update) |
|  | 20XX-XX-XX | Wang Xiaomin | Update |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Course Code | Product | Product Version | Course Issue |
|  |  |  |  |

Revision Record

IT OM Automation Tool RESTful web services design and implementationLab Guide



Huawei Technologies CO., LTD.

# Contents

[Contents ii](#_Toc164728056)

1. [Lab Environment 1](#_Toc164728057)

[1.1 About This Course 1](#_Toc164728058)

[1.2 Lab Envirnoment of Ansible Systems 2](#_Toc164728059)

1. [Create a Jersey hello world JAX-RS Application 3](#_Toc164728060)

[2.1 Introduction 3](#_Toc164728063)

[2.1.1 About this lab 3](#_Toc164728070)

[2.1.2 Objectives 3](#_Toc164728071)

[2.2 Lab Setup Guide 3](#_Toc164728072)

2.2.[1 Set Up Your Development Environment 3](#_Toc164728073)

2.2.[2 Create a New React Native Project 3](#_Toc164728074)

2.2.[3. Run Your React Native App on Android 3](#_Toc164728075)

[2.3 Configuration Steps 3](#_Toc164728076)

2.3.[1. Create New Dynamic Web Project 3](#_Toc164728077)

2.3.[2. Add JAX-RS / Jersey Dependent JAR files 4](#_Toc164728078)

2.3.[3.Create REST Service : 5](#_Toc164728079)

2.3.[4. Create Web.xml configuration File 5](#_Toc164728080)

1. [Create an annotation JAX-RS Application 8](#_Toc164728084)

[3.1 Introduction 8](#_Toc164728086)

[3.1.1 About this lab 8](#_Toc164728089)

[3.1.2 Objectives 8](#_Toc164728090)

[3.2 Configuration Steps 9](#_Toc164728091)

1. [Create a GET method 11](#_Toc164728092)

[4.1 Introduction 11](#_Toc164728094)

[4.1.1 About this lab 11](#_Toc164728100)

[4.1.2 Objectives 11](#_Toc164728101)

[4.2 Configuration Steps 11](#_Toc164728102)

1. [Create an UPDATE method 13](#_Toc164728103)

[5.1 Introduction 13](#_Toc164728105)

[5.1.1 About this lab 13](#_Toc164728112)

[5.1.2 Objectives 13](#_Toc164728113)

[5.2 Configuration Steps 13](#_Toc164728114)

1. [Validation & Exception Handling 15](#_Toc164728115)

[6.1 Introduction 15](#_Toc164728118)

[6.1.1 About this lab 15](#_Toc164728125)

[6.1.2 Objectives 15](#_Toc164728126)

[6.2 Configuration Steps 15](#_Toc164728127)

1. [MultiValuedMap and @FormParam 17](#_Toc164728128)

[7.1 Introductions 17](#_Toc164728131)

[7.1.1 About this lab 17](#_Toc164728139)

[7.1.2 Objectives 17](#_Toc164728140)

[7.2 Configuration Steps 17](#_Toc164728141)

1. [Create a DELETE Method 20](#_Toc164728142)

[8.1 Introduction 20](#_Toc164728144)

[8.1.1 About this lab 20](#_Toc164728147)

[8.1.2 Objectives 20](#_Toc164728148)

[8.2 Configuration Steps 20](#_Toc164728153)

1. [Matrix Parameters and Pagination 21](#_Toc164728154)

[9.1 Introduction 21](#_Toc164728156)

[9.1.1 About this lab 21](#_Toc164728157)

[9.1.2 Objectives 22](#_Toc164728158)

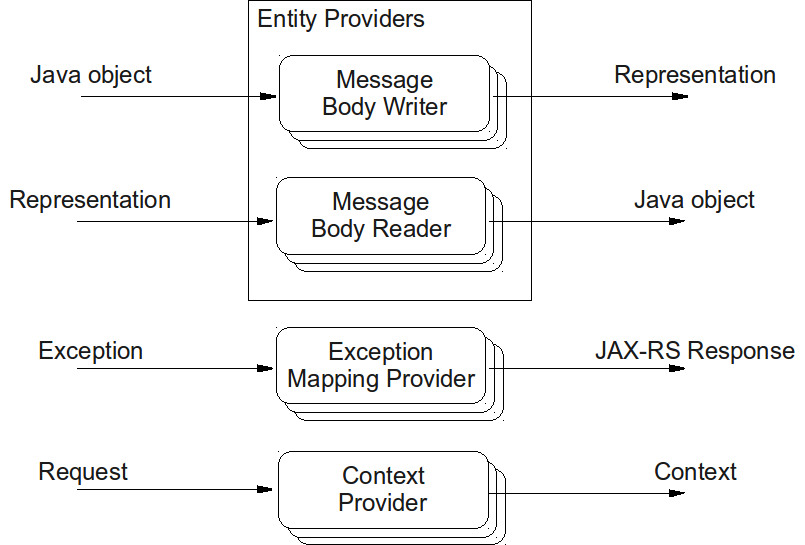
[9.2 Configuration Steps 22](#_Toc164728163)

# Lab Environment

## About This Course

* Create Java application by applying principles of RESTful Web Services using JAX-RS
* Create Web clients and Java Client for consuming RESTful Web Services
* Develop Restful services that handles failure scenarios using Exception handling mechanism provided by JAX-RS
* Apply Filters and Interceptors to RESTful Web Services
* Validate inputs supplied to a RESTful Web Services
* Apply Pagination and HATEOS in RESTful Web Services
* Create multiple versions of RESTful Web Services

## Lab Envirnoment of Java Systems



# Create a Jersey hello world JAX-RS Application



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  + Create and test a Jersey hello world JAX-RS Application.

### Objectives

* Upon completion of this task, you will be able to:
* Install Ansible
* Create user devops on all the nodes
* Setup the ssh keys and copy ssh keys for password less Authentication

## Lab Setup Guide

### Set Up Your Development Environment

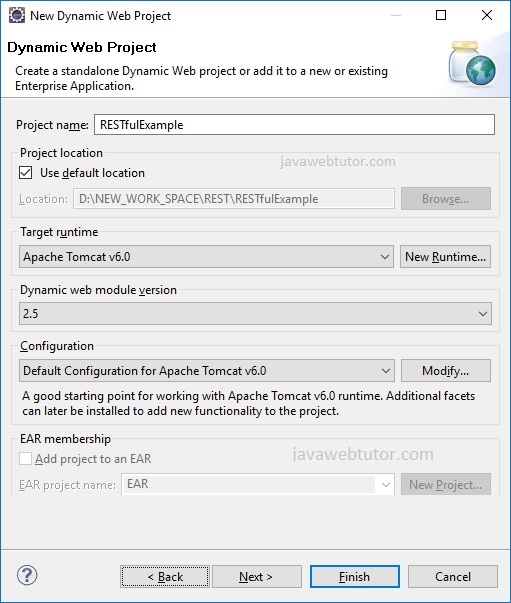
### Create a New React Native Project

### Run Your React Native App on Android

## Configuration Steps

### 1. Create New Dynamic Web Project

The first step is to create a Dynamic Web Project using Eclipse IDE. Open eclipse IDE and click on **File -> New -> Project** and select Dynamic Web Project wizard from the wizard list. Now name your project as "RESTfulExample" and Click on Finish button using the wizard window as shown below.



### 2. Add JAX-RS / Jersey Dependent JAR files

Add following jars in your project build path.

* asm-3.1.jar
* jersey-client-1.17.jar
* jersey-core-1.17.jar
* jersey-server-1.17.jar
* jersey-servlet-1.17.jar
* jsr311-api-1.1.1.jar

### 3 : Create REST Service :

Create a java class "HelloWorldService" in a package com.jwt.service and add following code into this.

#### HelloWorldService.java

package com.jwt.service;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.core.Response;

@Path("/hello")

public class HelloWorldService {

@GET

@Path("/{name}")

public Response getMsg(@PathParam("name") String name) {

String output = "Welcome : " + name;

return Response.status(200).entity(output).build();

}

}

### 4. Create Web.xml configuration File

Specify Jersey framework Servlet for our application in web.xml file.In web.xml, register "com.sun.jersey.  
spi.container.servlet.ServletContainer", and puts your Jersey service folder under "init-param","com.sun.jersey.  
config.property.packages".

#### web.xml

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

xmlns="<http://java.sun.com/xml/ns/javaee>"

xmlns:web="<http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd>"

xsi:schemaLocation="<http://java.sun.com/xml/ns/javaee>

<http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd>" id="WebApp\_ID" version="2.5">

<display-name>RESTfulExample</display-name>

<servlet>

<servlet-name>jersey-serlvet</servlet-name>

<servlet-class>

com.sun.jersey.spi.container.servlet.ServletContainer

</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>com.jwt.service</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

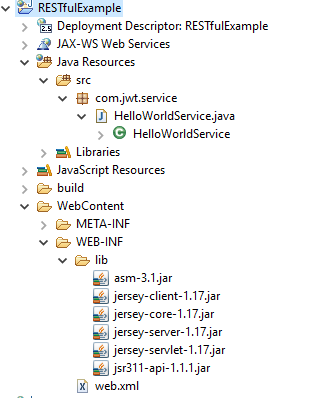
<servlet-name>jersey-serlvet</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

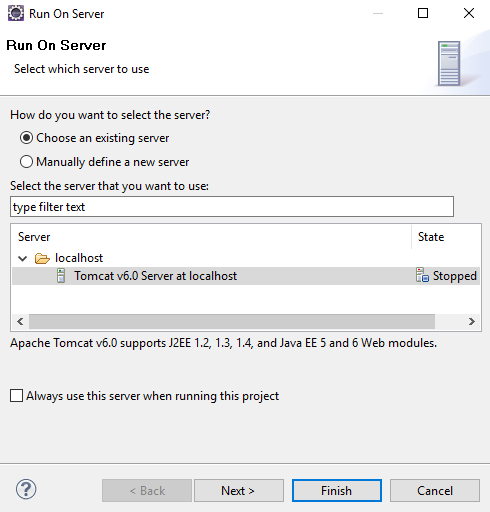
</web-app>

### Directory Structure of project



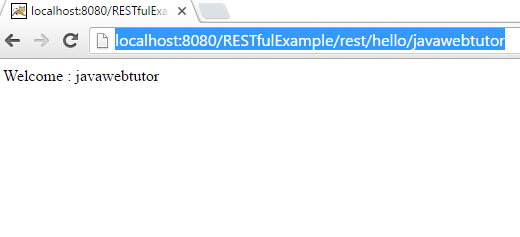
### Run the application :

Right click on project -> run as -> run on server Select Apache tomcat and click on finish as shown below.



By default eclipse will open <http://localhost:8080>  
/RESTfulExample/ . As In web.xml we have specified URL pattern as /rest/\* (line number 22) and in HelloWorldService.java we specified class level @path as /hello [ line number 8 ] and method level @path as {name} [ line number 1 ],So the final URL should be "<http://localhost:8080>  
/RESTfulExample/rest/hello/javawebtutor".

Test your REST service under: "<http://localhost:8080>  
/RESTfulExample/rest/hello/javawebtutor".You will get following output in browser:



You can download the source code of the example by clicking on the Download link below.

|  |
| --- |
| Source + Lib : [Download](https://www.javawebtutor.com/code/rest/RESTfulExample.zip) |



# Create an annotation JAX-RS Application



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  + Create an annotation JAR-XS Application
  + Test an annotation JAR-XS Application

### Objectives

Create a simple JAX-RS application using annotations to define RESTful resources and methods. Test the application using a web browser or a tool like Postman.

## Configuration Steps

 **Set Up Your Project:**

* Create a new Java web project in your IDE.
* Add the necessary JAX-RS dependencies to your project. You can use Maven or manually include the JAX-RS library (e.g., Jersey).

 **Write a JAX-RS Resource Class:**

* Create a new Java class to define your JAX-RS resource. Use annotations to specify the resource path and HTTP methods.
* Example (HelloResource.java):

java

import javax.ws.rs.GET;

import javax.ws.rs.Path;

@Path("/hello")

public class HelloResource {

@GET

public String sayHello() {

return "Hello, World!";

}

}

 **Configure Servlet Container:**

* Configure your servlet container (e.g., Apache Tomcat) to deploy JAX-RS applications. This typically involves adding JAX-RS servlet mappings to web.xml or using Servlet annotations.
* Example (web.xml):

xml

<servlet>

<servlet-name>Jersey Servlet</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>your.package.name</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey Servlet</servlet-name>

<url-pattern>/api/\*</url-pattern>

</servlet-mapping>

 **Deploy and Run the Application:**

* Deploy your application to the servlet container.
* Start the servlet container and access the application's endpoint in a web browser or using a tool like Postman.
* The endpoint URL should be http://localhost:8080/<context-path>/api/hello.

 **Test the Application:**

* Open a web browser or Postman and send a GET request to the endpoint URL (http://localhost:8080/<context-path>/api/hello).
* You should receive a response with the text "Hello, World!".

 **Experiment and Enhance:**

* Experiment with adding more JAX-RS resource classes, methods, and annotations to create a more comprehensive RESTful API.
* Test different HTTP methods (e.g., POST, PUT, DELETE) and handle request parameters and payloads as needed.

# Create a GET method



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
* Create a GET Method JAR-XS Application
* Test a GET Method JAR-XS Application

### Objectives

Create a JAX-RS application with a resource that implements a GET method to retrieve data. Test the application using a web browser or a tool like Postman.

## Configuration Steps

 **Set Up Your Project:**

* Create a new Java web project in your IDE.
* Add the necessary JAX-RS dependencies to your project. You can use Maven or manually include the JAX-RS library (e.g., Jersey).

 **Write a JAX-RS Resource Class:**

* Create a new Java class to define your JAX-RS resource. Use annotations to specify the resource path and HTTP methods.
* Example (HelloResource.java):

java

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/hello")

public class HelloResource {

@GET

@Produces(MediaType.TEXT\_PLAIN)

public String sayHello() {

return "Hello, World!";

}

}

 **Configure Servlet Container:**

* Configure your servlet container (e.g., Apache Tomcat) to deploy JAX-RS applications. This typically involves adding JAX-RS servlet mappings to web.xml or using Servlet annotations.
* Example (web.xml):

xml

<servlet>

<servlet-name>Jersey Servlet</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>your.package.name</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey Servlet</servlet-name>

<url-pattern>/api/\*</url-pattern>

</servlet-mapping>

 **Deploy and Run the Application:**

* Deploy your application to the servlet container.
* Start the servlet container and access the application's endpoint in a web browser or using a tool like Postman.
* The endpoint URL should be http://localhost:8080/<context-path>/api/hello.

 **Test the Application:**

* Open a web browser or Postman and send a GET request to the endpoint URL (http://localhost:8080/<context-path>/api/hello).
* You should receive a response with the text "Hello, World!".

 **Experiment and Enhance:**

* Experiment with adding more JAX-RS resource classes, methods, and annotations to create a more comprehensive RESTful API.
* Test different response types (e.g., JSON, XML) by using different @Produces annotations.

# Create an UPDATE method



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
* Create an UPDATE method in JAX-RS Application.
* Test an UPDATE method in JAX-RS Application

### Objectives

Create a JAX-RS application with a resource that implements an UPDATE method to update data. Test the application using a tool like Postman.

## Configuration Steps

**1.**  **Set Up Your Project:**

* Create a new Java web project in your IDE.
* Add the necessary JAX-RS dependencies to your project. You can use Maven or manually include the JAX-RS library (e.g., Jersey).

 **Write a JAX-RS Resource Class:**

* Create a new Java class to define your JAX-RS resource. Use annotations to specify the resource path and HTTP methods.
* Example (UserResource.java):

java

 import javax.ws.rs.PUT;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

@Path("/users")

public class UserResource {

@PUT

@Path("/{id}")

@Produces(MediaType.TEXT\_PLAIN)

public String updateUser(@PathParam("id") int id, String newData) {

// Update user with the specified ID using the newData

return "User with ID " + id + " updated successfully.";

}

}

 **Configure Servlet Container:**

* Configure your servlet container (e.g., Apache Tomcat) to deploy JAX-RS applications. This typically involves adding JAX-RS servlet mappings to web.xml or using Servlet annotations.
* Example (web.xml):

xml

 <servlet>

<servlet-name>Jersey Servlet</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>jersey.config.server.provider.packages</param-name>

<param-value>your.package.name</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey Servlet</servlet-name>

<url-pattern>/api/\*</url-pattern>

</servlet-mapping>

 **Deploy and Run the Application:**

* Deploy your application to the servlet container.
* Start the servlet container and access the application's endpoint in a tool like Postman.
* The endpoint URL should be http://localhost:8080/<context-path>/api/users/{id}. Replace {id} with the ID of the user to update.

 **Test the Application:**

* Use Postman or a similar tool to send a PUT request to the endpoint URL (http://localhost:8080/<context-path>/api/users/{id}).
* Include the new data for the user in the request body.
* You should receive a response indicating that the user was updated successfully.

 **Experiment and Enhance:**

* Experiment with handling different types of data and error scenarios in your updateUser method.
* Consider adding validation logic and error handling to ensure the stability and security of your application.

# Validation & Exception Handling



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  + Create a JAX-RS Application with Validation.
  + Create a JAX-RS Application with Exception Handling.

### Objectives

Create a JAX-RS application that handles resource creation with input validations and proper exception handling. Test the application using a tool like Postman.

## Configuration Steps

 **Set Up Your Project:**

* Create a new Java web project in your IDE.
* Add the necessary JAX-RS dependencies to your project. You can use Maven or manually include the JAX-RS library (e.g., Jersey).

 **Write a JAX-RS Resource Class:**

* Create a new Java class to define your JAX-RS resource. Use annotations to specify the resource path and HTTP methods.
* Implement input validations and exception handling logic in the resource methods.
* Example (UserResource.java):

java

import javax.validation.Valid;

import javax.validation.constraints.NotBlank;

import javax.ws.rs.\*;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

@Path("/users")

public class UserResource {

@POST

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public Response createUser(@Valid User user) {

if (user.getName() == null || user.getName().trim().isEmpty()) {

throw new BadRequestException("User name is required");

}

// Persist user in database or perform other actions

return Response.status(Response.Status.CREATED).entity(user).build();

}

}

 **Create a User Model:**

* Create a Java class to represent the user model with appropriate fields and annotations for validation.
* Example (User.java):

java

import javax.validation.constraints.NotBlank;

public class User {

@NotBlank(message = "Name cannot be blank")

private String name;

// Getter and setter methods

}

 **Configure Servlet Container:**

* Configure your servlet container (e.g., Apache Tomcat) to deploy JAX-RS applications. This typically involves adding JAX-RS servlet mappings to web.xml or using Servlet annotations.

 **Deploy and Run the Application:**

* Deploy your application to the servlet container.
* Start the servlet container and access the application's endpoint in a tool like Postman.

 **Test the Application:**

* Use Postman or a similar tool to send a POST request to the endpoint URL (http://localhost:8080/<context-path>/api/users).
* Include a JSON payload representing the user data in the request body.
* Test different scenarios, such as valid input, missing required fields, and invalid data, to observe how the application handles them.

 **Experiment and Enhance:**

* Experiment with adding more validations and error handling logic to your resource methods.
* Consider logging error messages and returning appropriate HTTP status codes and error responses to clients.

# MultiValuedMap and @FormParam



## Introductions



### About this lab

* Upon completion of this task, you will be able to:
* Create a JAX-RS Application with MultiValuedMap and @FormParam.
* Test a JAX-RS Application with MultiValuedMap and @FormParam

### Objectives

Create a JAX-RS application that accepts form data using @FormParam and MultiValuedMap. Test the application using a tool like Postman.

## Configuration Steps

 **Set Up Your Project:**

* Create a new Java web project in your IDE.
* Add the necessary JAX-RS dependencies to your project. You can use Maven or manually include the JAX-RS library (e.g., Jersey).

 **Write a JAX-RS Resource Class:**

* Create a new Java class to define your JAX-RS resource. Use annotations to specify the resource path and HTTP methods.
* Use @FormParam to extract form parameters from the request.
* Example (FormDataResource.java):

java

import javax.ws.rs.\*;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.MultivaluedMap;

@Path("/form")

public class FormDataResource {

@POST

@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)

@Produces(MediaType.TEXT\_PLAIN)

public String processFormData(@FormParam("name") String name, @FormParam("email") String email) {

return "Received form data: Name = " + name + ", Email = " + email;

}

@POST

@Path("/map")

@Consumes(MediaType.APPLICATION\_FORM\_URLENCODED)

@Produces(MediaType.TEXT\_PLAIN)

public String processFormDataMap(MultivaluedMap<String, String> formData) {

String result = "Received form data:";

for (String key : formData.keySet()) {

result += "\n" + key + " = " + formData.getFirst(key);

}

return result;

}

}

 **Configure Servlet Container:**

* Configure your servlet container (e.g., Apache Tomcat) to deploy JAX-RS applications. This typically involves adding JAX-RS servlet mappings to web.xml or using Servlet annotations.

 **Deploy and Run the Application:**

* Deploy your application to the servlet container.
* Start the servlet container and access the application's endpoints in a tool like Postman.

 **Test the Application:**

* Use Postman or a similar tool to send POST requests to the endpoint URLs:
  + http://localhost:8080/<context-path>/api/form for processFormData method.
  + http://localhost:8080/<context-path>/api/form/map for processFormDataMap method.
* Include form parameters in the request body.
* Verify that the application processes the form data correctly and returns the expected response.

 **Experiment and Enhance:**

* Experiment with handling different types of form data, such as checkboxes, radio buttons, and file uploads.
* Add validation and error handling logic to handle invalid or missing form data gracefully.
* Explore other JAX-RS annotations and features to extend the functionality of your application.

# Create a DELETE Method



## Introduction



### About this lab

* Upon completion of this task, you will be able to:
  + Create an UPDATE method in JAX-RS Application.
  + Test an UPDATE method in JAX-RS Application

### Objectives

Create a JAX-RS application with a resource that implements a DELETE method to delete resources. Test the application using a tool like Postman.



## Configuration Steps

 **Set Up Your Project:**

* Create a new Java web project in your IDE.
* Add the necessary JAX-RS dependencies to your project. You can use Maven or manually include the JAX-RS library (e.g., Jersey).

 **Write a JAX-RS Resource Class:**

* Create a new Java class to define your JAX-RS resource. Use annotations to specify the resource path and HTTP methods.
* Example (UserResource.java):

java

import javax.ws.rs.\*;

@Path("/users")

public class UserResource {

@DELETE

@Path("/{id}")

public void deleteUser(@PathParam("id") String id) {

// Delete user with the specified ID

// Perform deletion operation here

}

}

 **Configure Servlet Container:**

* Configure your servlet container (e.g., Apache Tomcat) to deploy JAX-RS applications. This typically involves adding JAX-RS servlet mappings to web.xml or using Servlet annotations.

 **Deploy and Run the Application:**

* Deploy your application to the servlet container.
* Start the servlet container and access the application's endpoint in a tool like Postman.

 **Test the Application:**

* Use Postman or a similar tool to send a DELETE request to the endpoint URL (http://localhost:8080/<context-path>/api/users/{id}). Replace {id} with the ID of the user to delete.
* Verify that the application successfully deletes the user resource and returns an appropriate response.

 **Experiment and Enhance:**

* Experiment with handling different scenarios, such as deleting non-existent resources or handling errors gracefully.
* Add authentication and authorization mechanisms to restrict access to the DELETE endpoint.
* Explore other JAX-RS annotations and features to extend the functionality of your application.

# Matrix Parameters and Pagination



## Introduction

### About this lab

* Upon completion of this task, you will be able to:
  + Create and test a JAX-RS Application with Matrix Parameters and Pagination.
  + Test a JAX-RS Application with Matrix Parameters and Pagination.

### Objectives

Create a JAX-RS application with a resource that accepts matrix parameters for filtering data and implements pagination. Test the application using a tool like Postman.







## Configuration Steps

 **Set Up Your Project:**

* Create a new Java web project in your IDE.
* Add the necessary JAX-RS dependencies to your project. You can use Maven or manually include the JAX-RS library (e.g., Jersey).

 **Write a JAX-RS Resource Class:**

* Create a new Java class to define your JAX-RS resource. Use annotations to specify the resource path and HTTP methods.
* Use matrix parameters for filtering data and implementing pagination.
* Example (ProductResource.java):

java

 import javax.ws.rs.\*;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

@Path("/products")

public class ProductResource {

@GET

@Produces(MediaType.APPLICATION\_JSON)

public Response getProducts(@MatrixParam("category") String category,

@MatrixParam("page") int page,

@MatrixParam("limit") int limit) {

// Implement logic to fetch products based on category and pagination

// Dummy implementation for demonstration

String response = String.format("Fetching products: category=%s, page=%d, limit=%d", category, page, limit);

return Response.ok(response).build();

}

}

 **Configure Servlet Container:**

* Configure your servlet container (e.g., Apache Tomcat) to deploy JAX-RS applications. This typically involves adding JAX-RS servlet mappings to web.xml or using Servlet annotations.

 **Deploy and Run the Application:**

* Deploy your application to the servlet container.
* Start the servlet container and access the application's endpoint in a tool like Postman.

 **Test the Application:**

* Use Postman or a similar tool to send GET requests to the endpoint URL (http://localhost:8080/<context-path>/api/products). Include matrix parameters for filtering and pagination in the request URL, e.g., ;category=electronics;page=1;limit=10.
* Verify that the application fetches products based on the specified category and pagination parameters and returns an appropriate response.

 **Experiment and Enhance:**

* Experiment with different combinations of matrix parameters to filter data and paginate results.
* Implement additional features such as sorting, searching, or advanced filtering using matrix parameters.
* Explore other JAX-RS annotations and features to extend the functionality of your application.