

## Cloud Computing Practicals

**Name:** Omkar Sawardekar

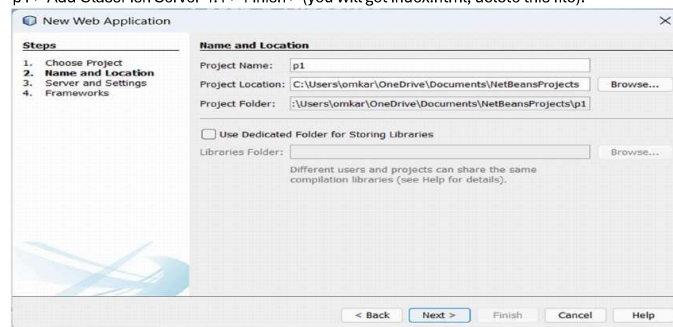
**Roll no.:** 22109

### Practical 1 - Create a Simple SOAP service

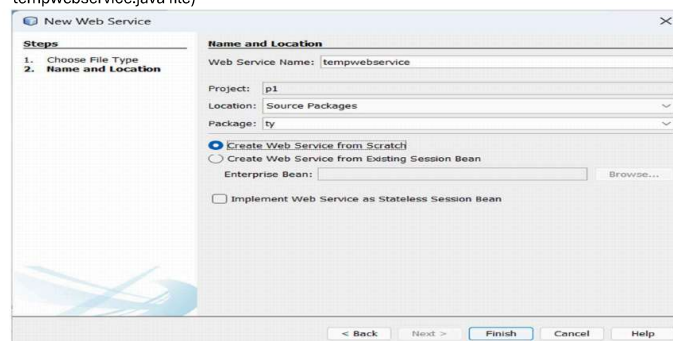
**Aim:** To create a Simple Web Service that converts the temperature from Celsius to Fahrenheit and vice versa.

**Steps:**

Step 1: Create a Web Application: File > New Project > Java Web > Web Application > Name: p1 > Add GlassFish Server 4.1 > Finish > (you will get index.html, delete this file).



Step 2: Create a Web Service in that project: Right Click on Project Name > New > Web Service... > Name: tempwebservice and Package Name: ty > OK > (you will get tempwebservice.java file)



Step 3: As we have created our WEB SERVICE (i.e. tempwebservice), WE'LL NOW ADD SOME FUNCTIONALITIES (i.e. temperature conversion methods). So, in the class tempwebservice, add 2 methods by doing:

a) Right Click > Insert Code > Add Web Service Operation...>

Name: convertFtoC >

Return Type: float >

Add a Parameter 'a' with Type: float > OK >

Now add formula in return statement  $((a-32) * 5/9)$  .

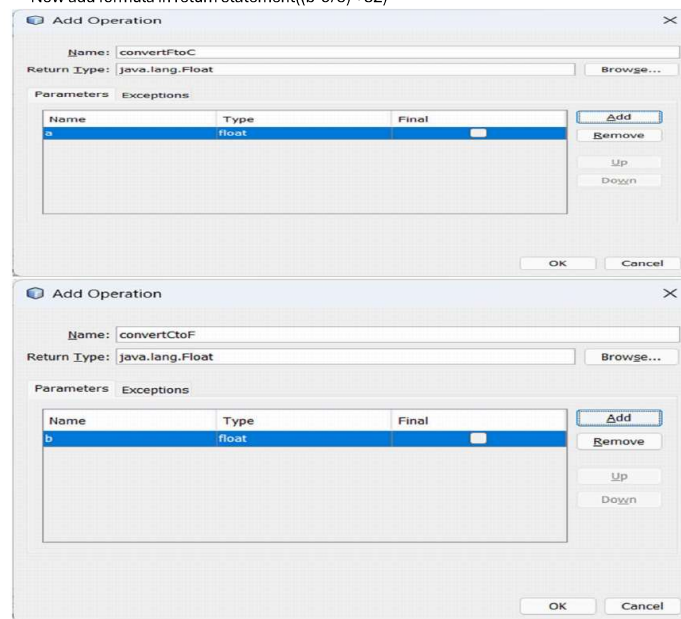
b) Right Click > Insert Code > Add Web Service Operation...>

Name: convertCtoF >

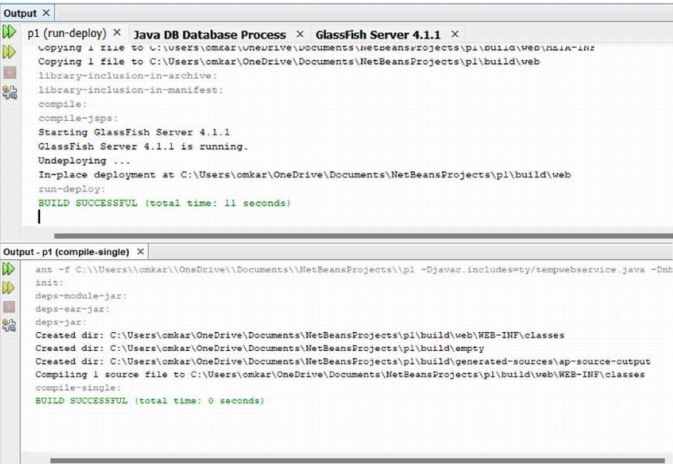
Return Type: float >

Add a Parameter 'b' with Type: float > OK >

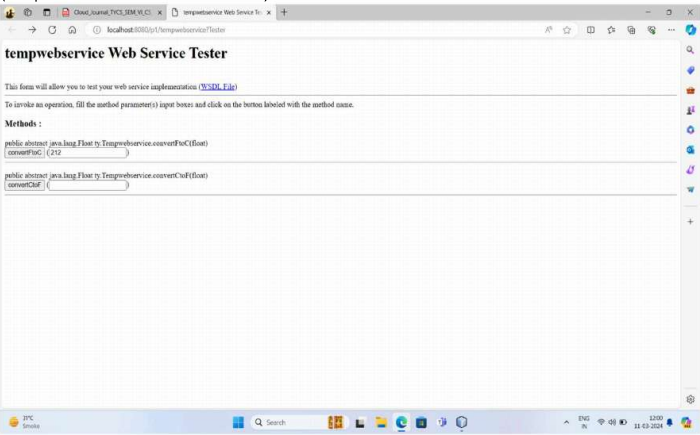
Now add formula in return statement  $((b*9/5) + 32)$

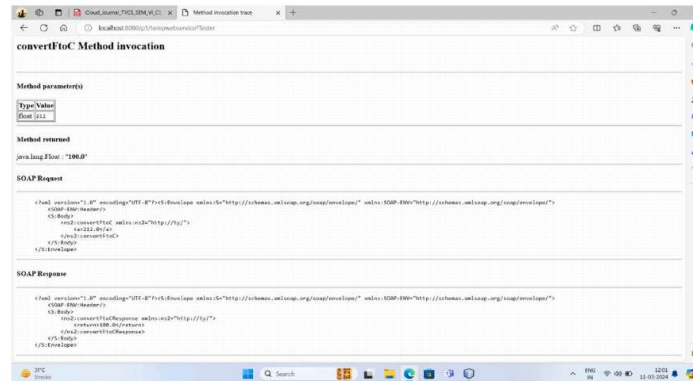


Step 4: Compile(F9) and Deploy (right click on Project Name > Deploy).

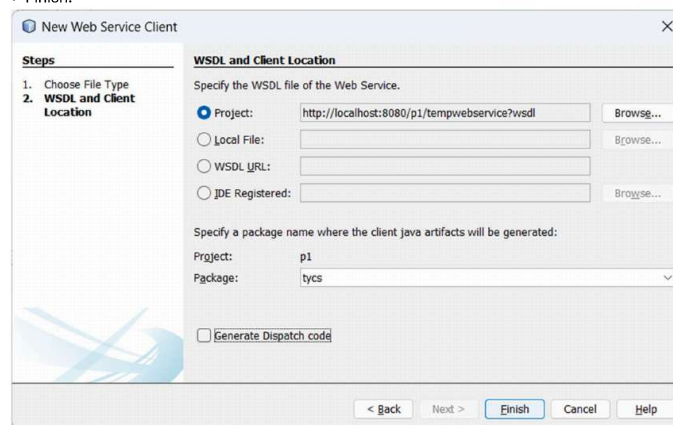


Step 5: Now we will test our web service. So, navigate to: Projects > Project Name > Web Services > right click on tempwebservice > click Test Web Service > (the browser will open). (tempwebservice here is a WSDL file).





Step 6: Now we will be using the created Web Service (i.e. tempwebservice) by creating our WEB SERVICE CLIENT. So, Right Click on Project Name > New > Other > Web Services > Web Service Client... > Browse your Project Name and select WSDL file > Add Package Name: tycs > Finish.



Step 7: Now to make front page, we need to create a JSP file. So, Right Click on Project Name > New > JSP > File Name: index > Finish. Now from the palette, add HTML Form (Action: indexaction.jsp, Type: text, Method: Post) Inside Form tag, add Text Input (Name: txt) Inside Form tag, add Button (Label: Convert, Type: submit, Name: convert).

**New JSP**

**Steps**

1. Choose File Type
2. Name and Location

**Name and Location**

File Name:

Project:

Location:

Folder:

Browse...

Created File: C:\Users\omkar\OneDrive\Documents\NetBeansProjects\p1\web\index.jsp

Options:

☒ JSP File (Standard Syntax) ☐ Create as a JSP Segment

☐ JSP Document (XML Syntax)

Description: A JSP file using JSP standard syntax.

< Back Next > Finish Cancel Help

Step 8: Now to create the action behind that, we will create another JSP file, with a name that is declared in the previous JSP file. So, Right Click on Project Name > New > JSP > File Name: indexaction.jsp > Finish. Then add some `<%JSP code%>` under body.

**New JSP**

**Steps**

1. Choose File Type
2. Name and Location

**Name and Location**

File Name:

Project:

Location:

Folder:

Browse...

Created File: I:\Users\omkar\OneDrive\Documents\NetBeansProjects\p1\web\indexaction.jsp

Options:

☒ JSP File (Standard Syntax) ☐ Create as a JSP Segment

☐ JSP Document (XML Syntax)

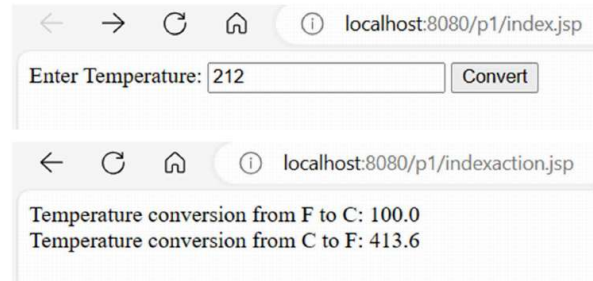
Description: A JSP file using JSP standard syntax.

< Back Next > Finish Cancel Help

Step 9: Now RUN "index.jsp" file.

Step 10: Finish!

**Outputs:**



The first screenshot shows a web browser window with the address bar displaying 'localhost:8080/p1/index.jsp'. Below the address bar, there is a form with the label 'Enter Temperature:' followed by a text input field containing the value '212'. To the right of the input field is a button labeled 'Convert'.

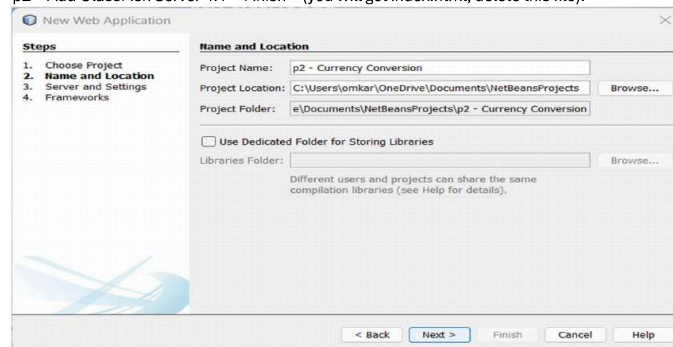
The second screenshot shows the same browser window after the 'Convert' button was clicked. The address bar now displays 'localhost:8080/p1/indexaction.jsp'. The page content shows two lines of text: 'Temperature conversion from F to C: 100.0' and 'Temperature conversion from C to F: 413.6'.

## Practical 2 - Create a Simple SOAP service:

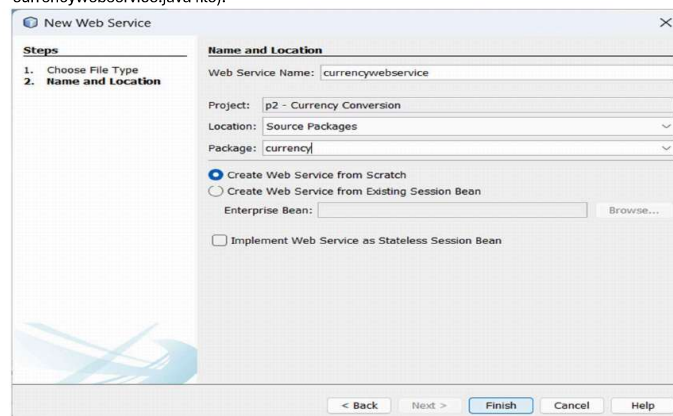
**Aim:** Define a simple web service like converting Rs into Dollar and call it from different platform like JAVA and .NET.

### Steps:

Step 1: Create a Web Application: File > New Project > Java Web > Web Application > Name: p2 > Add GlassFish Server 4.1 > Finish > (you will get index.html, delete this file).



Step 2: Create a Web Service in that project: Right Click on Project Name > New > Web Service... > Name: currencywebservice and Package Name: currency > OK > (you will get currencywebservice.java file).



Step 3: As we have created our WEB SERVICE (i.e. currencywebservice), WE'LL NOW ADD SOME FUNCTIONALITIES (i.e. temperature conversion methods). So, in the class currencywebservice, add 2 methods by doing:

a) Right Click > Insert Code > Add Web Service Operation...>

Name: INRtoDOLL >

Return Type: string >

Add a Parameter 'INR' with Type: double > OK >

Now add formula in return statement:  $(\text{INR} + \text{" rupees in dollar are " } + (\text{INR}/83.11))$  .

b) Right Click > Insert Code > Add Web Service Operation...>

Name: DOLLtoINR >

Return Type: string >

Add a Parameter 'DOLL' with Type: double > OK >

Now add formula in return statement:  $(\text{DOLL} + \text{" dollars in rupees are " } + (\text{DOLL} * 83.11))$ .

The screenshot shows the 'Add Operation' dialog box with the following details:

- Name: INRtoDOLL
- Return Type: java.lang.String
- Parameters table:

Name	Type	Final
INR	double	<input type="checkbox"/>
- Buttons: Add, Remove, Up, Down, OK, Cancel

The screenshot shows the 'Add Operation' dialog box with the following details:

- Name: DOLLtoINR
- Return Type: java.lang.String
- Parameters table:

Name	Type	Final
DOLL	double	<input type="checkbox"/>
- Buttons: Add, Remove, Up, Down, OK, Cancel



Step 4: Compile(F9) and Deploy (right click on Project Name > Deploy).

Step 5: Now we will test our web service. So, navigate to: Projects > Project Name > Web Services > right click on currencywebservice > click Test Web Service > (the browser will open). (currencywebservice here is a WSDL file).

localhost:8080/p2/\_Currency\_Conversion/currencywebservice?Tester

currencywebservice Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract java.lang.String currency.Currencywebservice.dollToINR(double)  
dollToINR ( 10 )

public abstract java.lang.String currency.Currencywebservice.inRtoDOLL(double)  
inRtoDOLL ( 200 )

dollToINR Method invocation

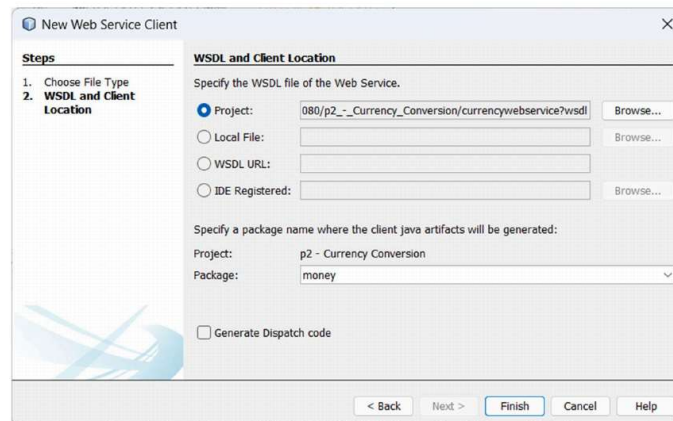
Method parameter(s)

Type	Value
double	10

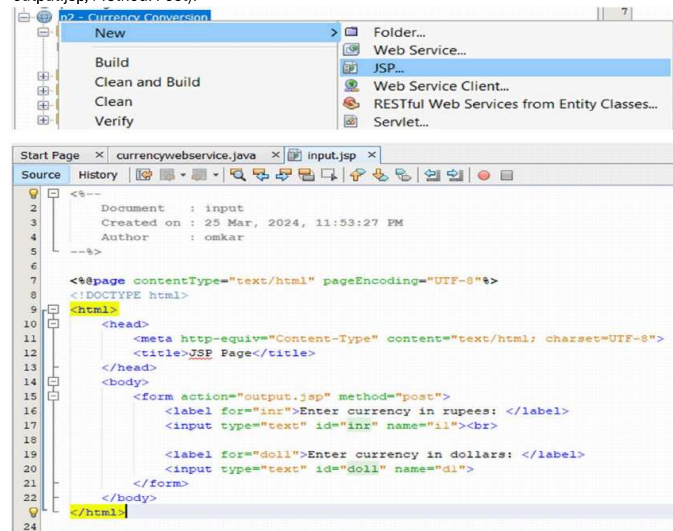
Method returned

java.lang.String : "10.0 dollars in rupees is 831.1"

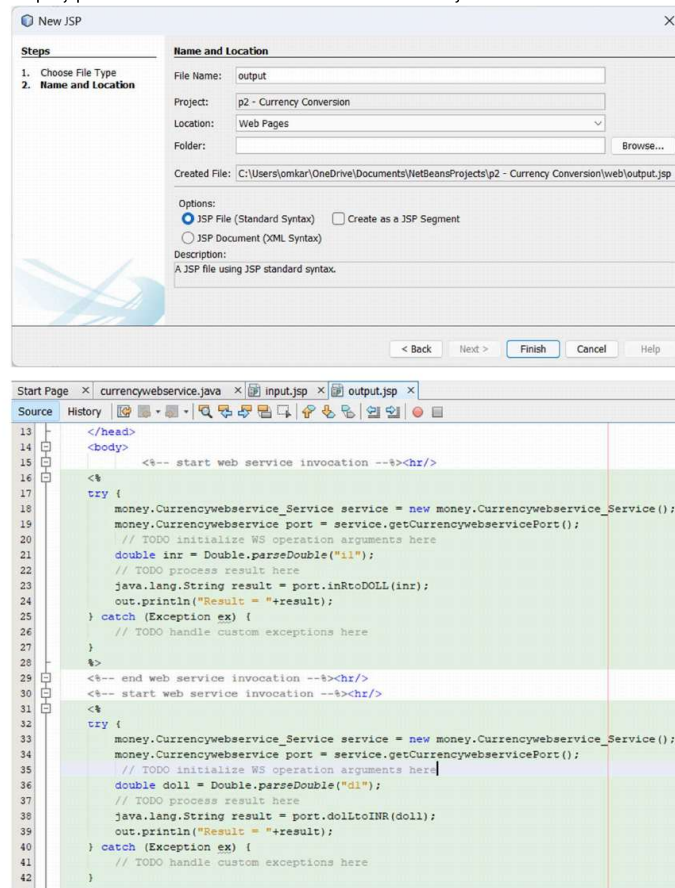
Step 6: Now we will be using the created Web Service (i.e. currencywebservice) by creating our WEB SERVICE CLIENT. So, Right Click on Project Name > New > Other > Web Services > Web Service Client... > Browse your Project Name and select WSDL file > Add Package Name: money > Finish.



Step 7: Now to make front page, we need to create a JSP file. So, Right Click on Project Name > New > JSP > File Name: input > Finish. Now from the palette, add HTML Form (Action: output.jsp, Method: Post).



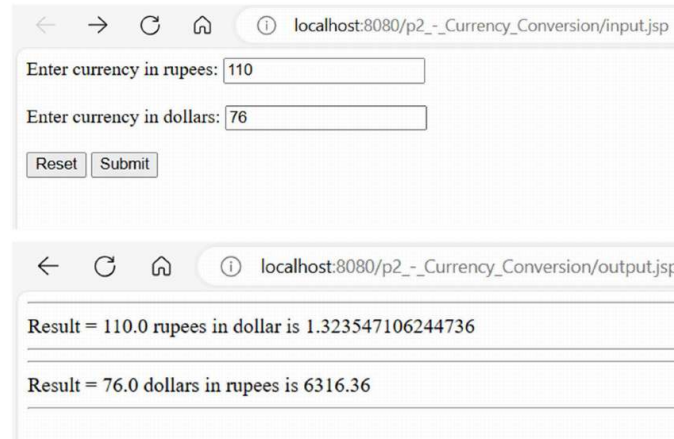
Step 8: Now to create the action behind that, we will create another JSP file, with a name that is declared in the previous JSP file. So, Right Click on Project Name > New > JSP > File Name: output.jsp > Finish. Then add some <%JSP code%> under body.



Step 9: Now RUN "index.jsp" file.

Step 10: Finish!

**Outputs:**



The image displays two screenshots of a web browser interface for a currency conversion application.

The first screenshot shows the input page (`localhost:8080/p2_-_Currency_Conversion/input.jsp`). It contains two text input fields: "Enter currency in rupees:" with the value "110" and "Enter currency in dollars:" with the value "76". Below the fields are two buttons: "Reset" and "Submit".

The second screenshot shows the output page (`localhost:8080/p2_-_Currency_Conversion/output.jsp`). It displays the results of the conversion in two separate rows, each with a horizontal line above and below the text:

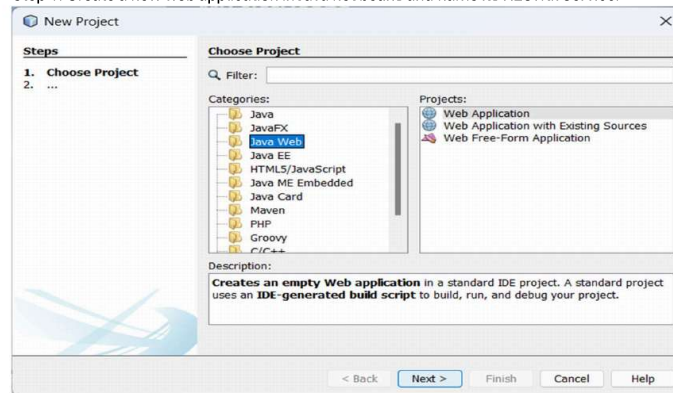
- Result = 110.0 rupees in dollar is 1.323547106244736
- Result = 76.0 dollars in rupees is 6316.36

### Practical 3 - Create RESTful Web service:

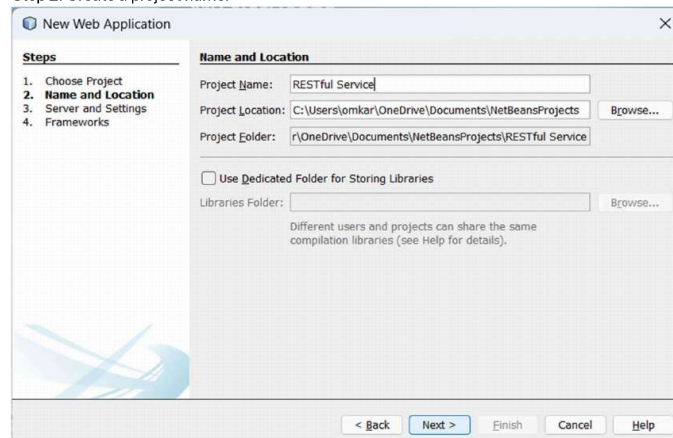
**Aim:** Define a RESTful web service that accepts the details to be stored in a database and performs CRUD operation.

#### Steps:

Step 1: Create a new web application in Java net beans and name its RESTful Service.



Step 2: Create a project name.



New Web Application

Steps

1. Choose Project
2. Name and Location
3. **Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application: <None>

Server: GlassFish Server 4.1.1

Add...

Java EE Version: Java EE 7 Web

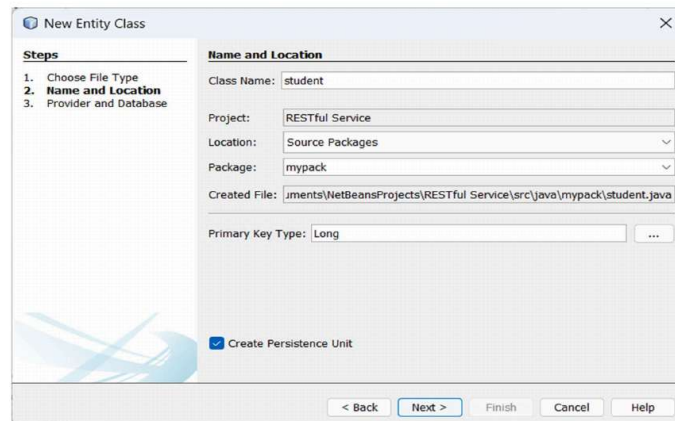
Context Path: /RESTful\_Service

The screenshot shows the IntelliJ IDEA IDE. The 'New' menu is open, and 'Entity Class...' is selected. The background shows a project named 'RESTful\_Service' with a 'pom.xml' file open in the editor. The 'pom.xml' file contains the following content:

```

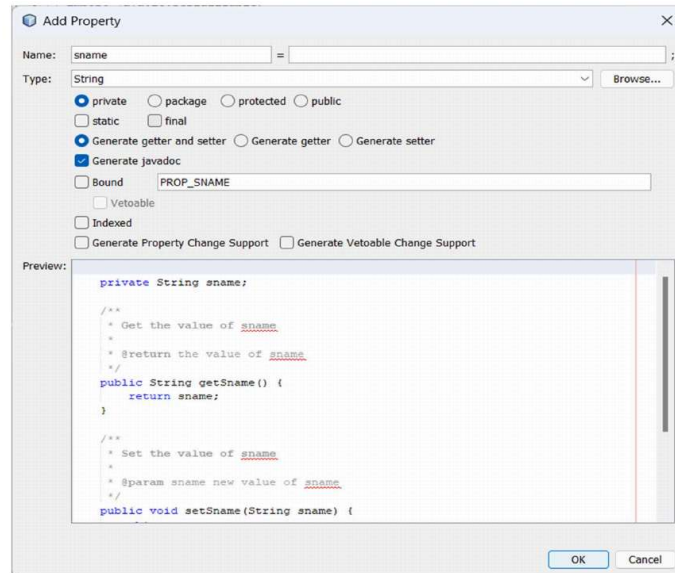
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>RESTful_Service</artifactId>
  <version>1.0.0</version>
  <packaging>war</packaging>
  <name>RESTful_Service</name>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>5.3.12</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.3.12</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>

```



The 'New Entity Class' dialog box is shown. It has a 'Steps' sidebar on the left with three items: '1. Choose File Type', '2. Name and Location' (which is selected), and '3. Provider and Database'. The main area is titled 'Name and Location' and contains the following fields: 'Class Name' (student), 'Project' (RESTful Service), 'Location' (Source Packages), 'Package' (mypack), 'Created File' (j:\ments\NetBeansProjects\RESTful Service\src\java\mypack\student.java), and 'Primary Key Type' (Long). At the bottom, there is a checkbox for 'Create Persistence Unit' which is checked. Navigation buttons at the bottom include '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

Step 5: Insert code after set id property.



The 'Add Property' dialog box is shown. It has a 'Name' field (sname) and a 'Type' dropdown (String). There are radio buttons for 'private' (selected), 'package', 'protected', and 'public'. There are checkboxes for 'static' and 'final'. There are radio buttons for 'Generate getter and setter' (selected), 'Generate getter', and 'Generate setter'. There is a checked checkbox for 'Generate javadoc'. There is a checkbox for 'Bound' with a text field containing 'PROP\_SNAME'. There is a checkbox for 'Vetoable'. There is a checkbox for 'Indexed'. There are checkboxes for 'Generate Property Change Support' and 'Generate Vetoable Change Support'. A 'Preview' section at the bottom shows the following code:

```
private String sname;

/**
 * Get the value of sname
 *
 * @return the value of sname
 */
public String getName() {
    return sname;
}

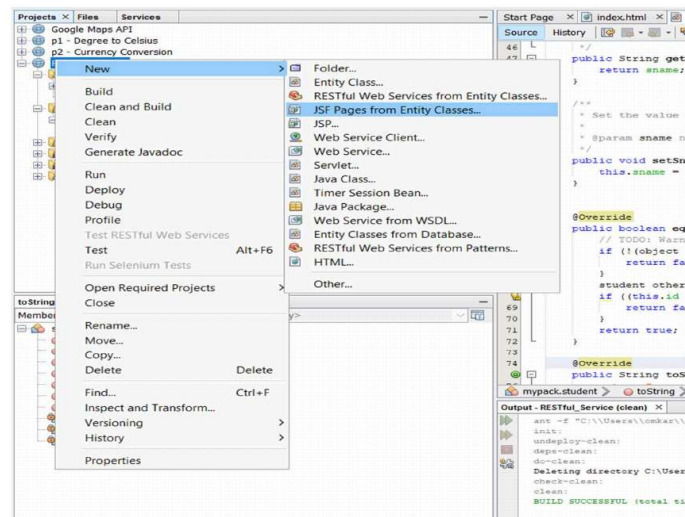
/**
 * Set the value of sname
 *
 * @param sname new value of sname
 */
public void setName(String sname) {
```

Buttons at the bottom are 'OK' and 'Cancel'.

Step 6: Highlighted text below in the image will be generated.

```
public int hashCode() {  
    int hash = 0;  
    hash += (id != null ? id.hashCode() : 0);  
    return hash;  
}  
  
private String sname;  
  
/**  
 * Get the value of sname  
 * @return the value of sname  
 */  
public String getName() {  
    return sname;  
}  
  
/**  
 * Set the value of sname  
 * @param sname new value of sname  
 */  
public void setName(String sname) {  
    this.sname = sname;  
}
```

Step 7: Add JSF pages from the entity class.





The image displays three sequential screenshots of the 'New JSF Pages from Entity Classes' wizard in NetBeans, showing the progression from selecting entity classes to final configuration.

**Screenshot 1: Entity Classes Selection**

- Steps:** 1. Choose File Type, 2. Entity Classes, 3. Generate JSF Pages and Classes, 4. JavaServer Faces Configuration.
- Entity Classes:** Available Entity Classes: (empty list). Selected Entity Classes: mypack.student.
- Buttons:** Add >, < Remove, Add All >>, << Remove All.
- Checkbox:** ☒ Include Referenced Classes.
- Navigation:** < Back, Next >, Finish, Cancel, Help.

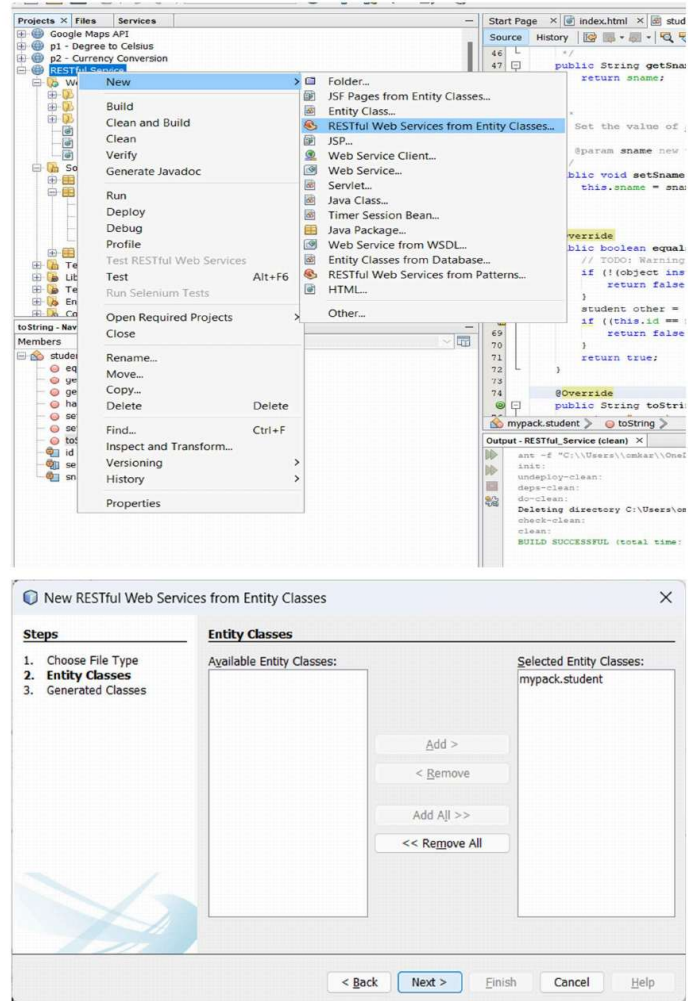
**Screenshot 2: Generate JSF Pages and Classes**

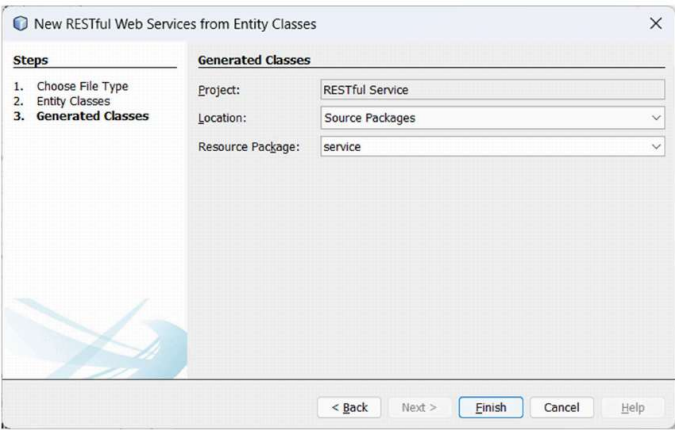
- Steps:** 1. Choose File Type, 2. Entity Classes, 3. Generate JSF Pages and Classes, 4. JavaServer Faces Configuration.
- Generate JSF Pages and Classes:** Specify the package of existing or new EJBs and the package of JSF classes.
  - Project: RESTful Service
  - Location: Source Packages
  - Session Bean Package: mypack
  - JSF Classes Package: mypack
- Specify the location of new JSF pages:**
  - JSF Pages Folder: (empty) Browse...
  - Localization Bundle Name: /bundle
  - ☐ Override existing files
- Choose Templates:** Standard JavaServer Faces Customize Template
- Navigation:** < Back, Next >, Finish, Cancel, Help.

**Screenshot 3: JavaServer Faces Configuration**

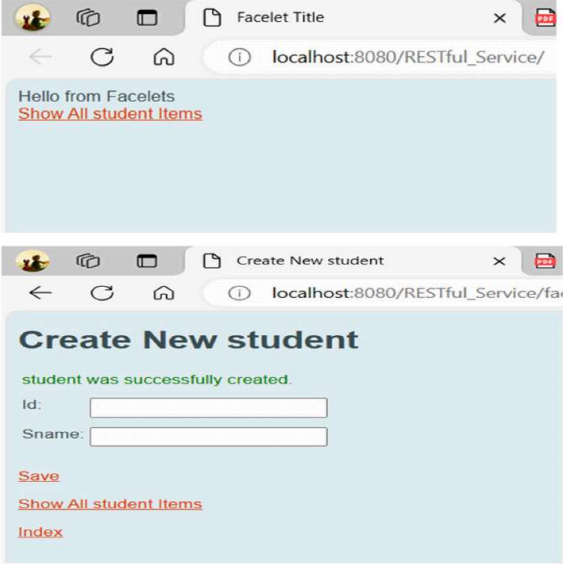
- Steps:** 1. Choose File Type, 2. Entity Classes, 3. Generate JSF Pages and Classes, 4. JavaServer Faces Configuration.
- JavaServer Faces Configuration:** Specify JavaServer Faces configuration and library information.
  - Libraries:** Configuration Components
  - ☒ Server Library: JSF 2.2
  - ☐ Registered Libraries: JSF 2.2
  - ☐ Create New Library
    - JSF Folder or JAR: (empty) Browse...
    - Library Name: (empty)
- Navigation:** < Back, Next >, Finish, Cancel, Help.

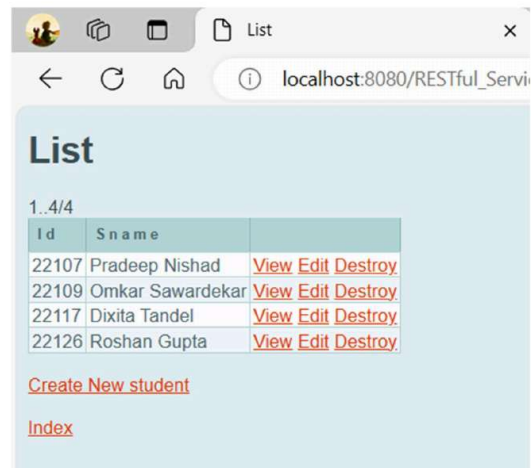
Step 8: Now add RESTful web services from entity class as shown below in the image.





Step 9: Now deploy and run the project (make sure Oracle is shut on port 8080).





The screenshot shows a web browser window with a single tab titled 'List'. The address bar displays 'localhost:8080/RESTful\_Servi'. The main content area has a light blue background and features a heading 'List' in bold. Below the heading is a pagination indicator '1..4/4'. A table with two columns, 'Id' and 'S name', lists four students. Each row includes three red links: 'View', 'Edit', and 'Destroy'. Below the table are two more red links: 'Create New student' and 'Index'.

Id	S name	
22107	Pradeep Nishad	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Destroy</a>
22109	Omkar Sawardekar	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Destroy</a>
22117	Dixita Tandel	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Destroy</a>
22126	Roshan Gupta	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Destroy</a>

[Create New student](#)

[Index](#)

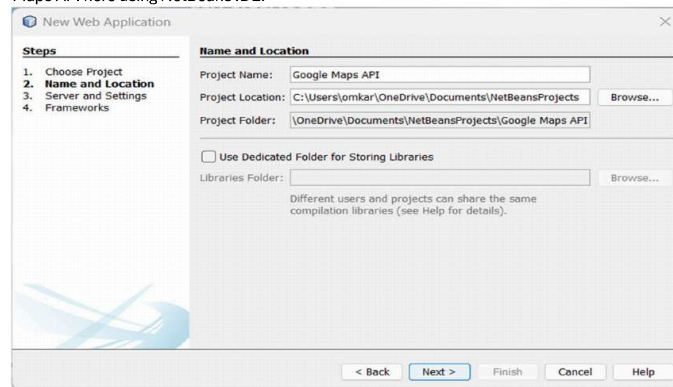
Step 10: Conclusion: we have successfully defined a RESTful web service that accepts the details to be stored in a database and performs CRUD operations. We are not storing data in DBMS we are just accepting in the list as we have to accept details.

#### Practical 4 - Create a Simple RESTful Web service:

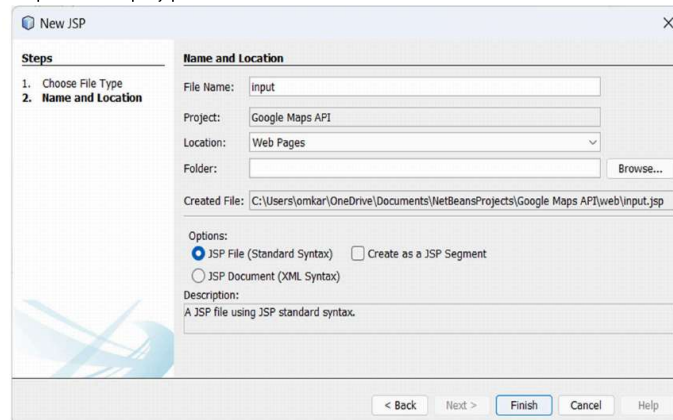
**Aim:** Develop application to consume Google's search / Google's Map RESTful Web service.

**Steps:**

Step 1: First of all, we need to create a Java Web Application with any name, let it be Google Maps API here using NetBeans IDE.



Step 2: Create input.jsp.



Step 3: The code inside the input.jsp will be:

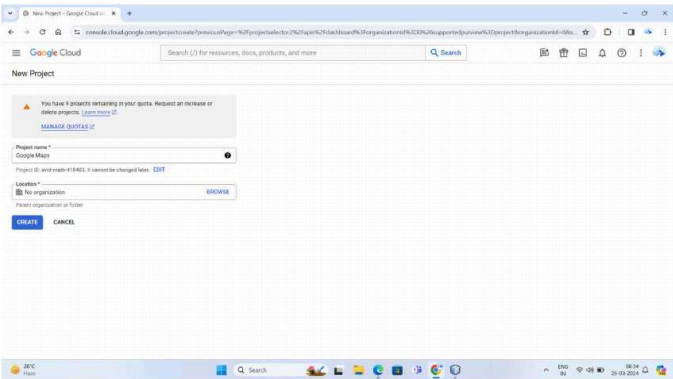
**Input.jsp**

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <form action="index.jsp">
      <pre>
        Enter latitude:<input type="text" name="t1" />
        Enter longitude:<input type="text" name="t2" />
        <input type="submit" value="Show" />
      </pre>
    </form>
  </body>
</html>
```

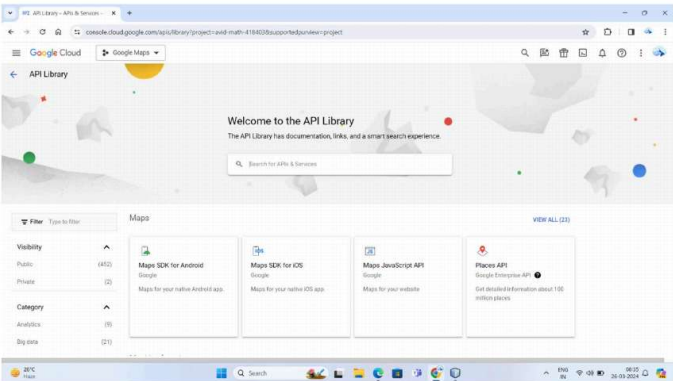
Step 4: Before running the application, we need the Google API key. The steps are shown here:

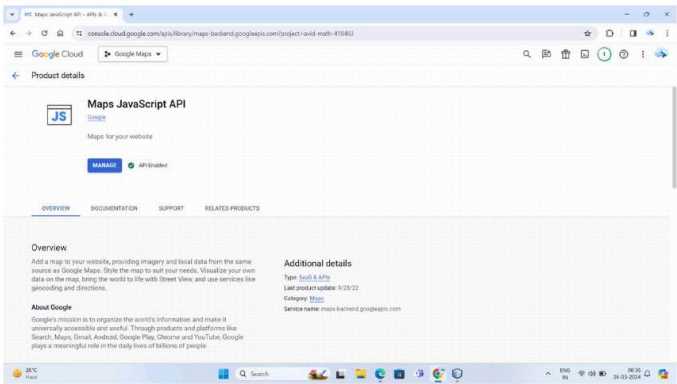
- Visit Google APIs Console (<https://console.developers.google.com>, you have to login with your Google account).

a) Create a new API Project.

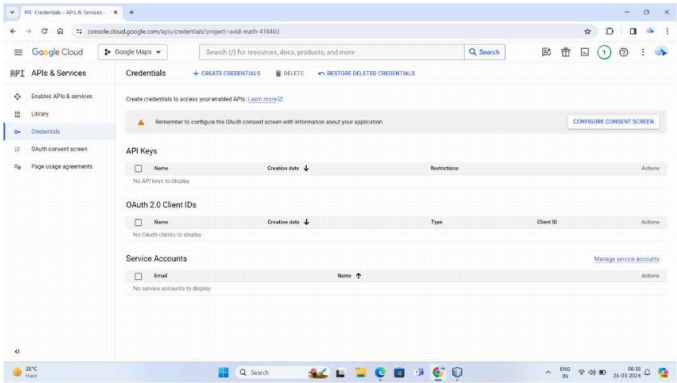


b) Enable the Google Maps API V3.

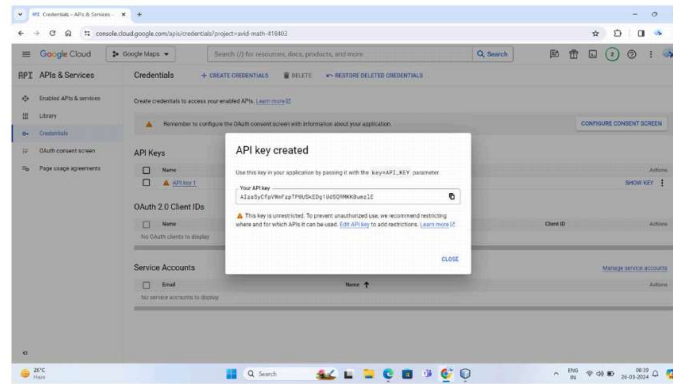




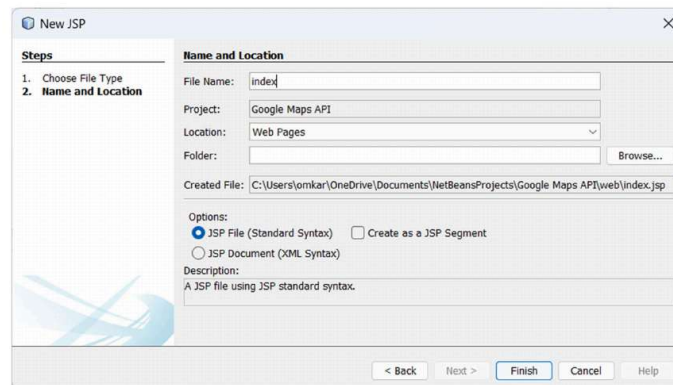
Step 5: Now, to create an API key, go to Google Cloud > Enabled API and Services > Credentials > Create Credentials > Create API Key > Copy the API key.







Step 6: Create another file index.jsp.



Index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
```

```
<style>

#map
{
height: 400px;
width: 100%;
}

</style>
</head>
<body>

<%
    double lati=Double.parseDouble(request.getParameter("t1"));
    double longi=Double.parseDouble(request.getParameter("t2"));
%>

<h3> Google Maps </h3>
<div id="map"></div>
<script lang="javascript">
function initMap()
{
    var info={lat: <%=lati%>, lng: <%=longi%>};
    var map = new google.maps.Map(document.getElementById('map'),
    {
        zoom: 4, center: info
    });
    var marker = new google.maps.Marker
    ({
        position: info, map: map
    });
}
</script>
```

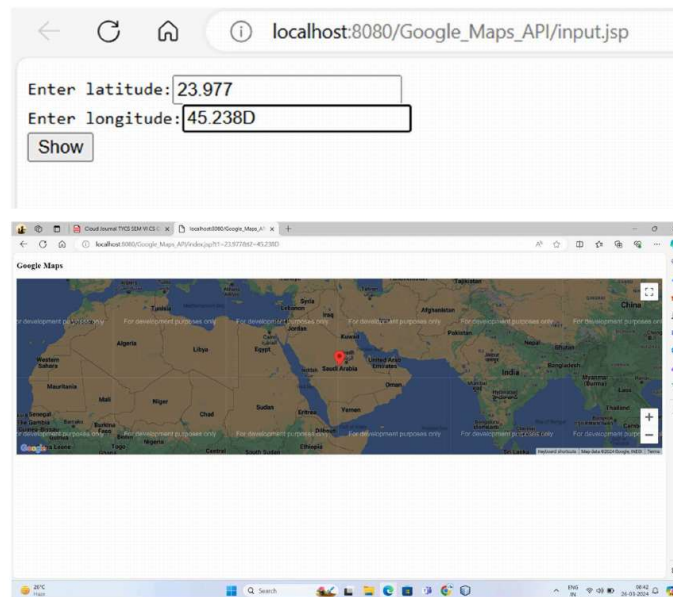
```
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=AlzaSyCfpVWmF
zpTP0USkEdg1UdSQRMKK0umzIE&callback=initMap">
</script>

</body>

</html>
```

Step 7: Run "input.jsp" file.

#### Outputs:



## Practical 5 – KVM:

**Aim:** To install and configure virtualization using KVM.

### Steps:

#### Step 1: Check Hardware Virtualization Support:

Ensure that your CPU supports hardware virtualization (Intel-VT or AMD-V) and that is enabled in the BIOS settings

#### Step 2: Install KVM Packages:

- 1) Update your package repository: `sudo apt update`
- 2) Install the KVM and QEMU packages: `sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils`

#### Step 3: Start and Enable Libvirt Service:

- 1) Start the libvirt service: `sudo systemctl start libvirtd`
- 2) Enable libvirt to start on boot: `sudo systemctl enable libvirtd`

#### Step 4: Configure Networking:

- 1) Create the network bridge interface: `sudo nano/etc/network/interfaces`
- 2) Add the following lines: `auto br0 iface br0 inet dhcp bridge_ports enp0s3`
- 3) Restart the networking service: `sudo systemctl restart networking`

#### Step 5: Create a Virtual Machine:

You can create and manage virtual machines using tools like virt-manager or virsh.

#### Step 6: Install Virt-Manager:

If you prefer GUI tool for managing virtual machine, you can install virt-manager: `sudo apt install virt-manager`

#### Step 7: Launch Virt-Manager:

You can launch Virt-Manager from the application menu or using the command `virt-manager` in the terminal.

#### Step 8: Create Virtual Machine:

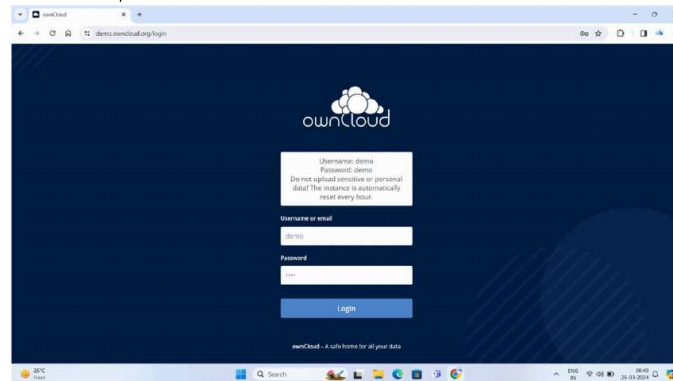
Using Virt-Manager, you can create, configure and manage virtual machines on your KVM hypervisor

**Practical 6 – OwnCloud:**

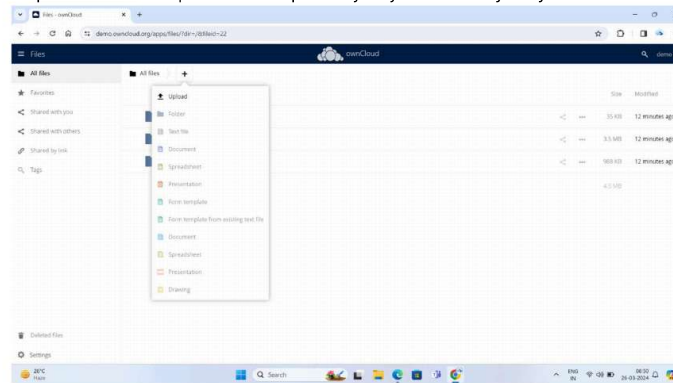
**Aim:** To upload file on OwnCloud and share it across internet.

**Steps:**

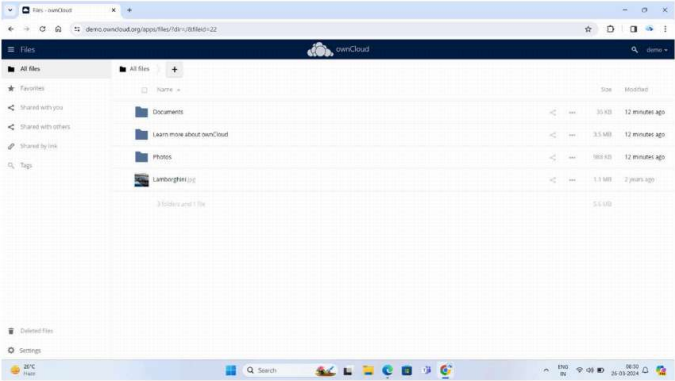
Step 1: Go to Google Chrome, search “OwnCloud demo”. Then open the website, fill up the username and password as “demo”.



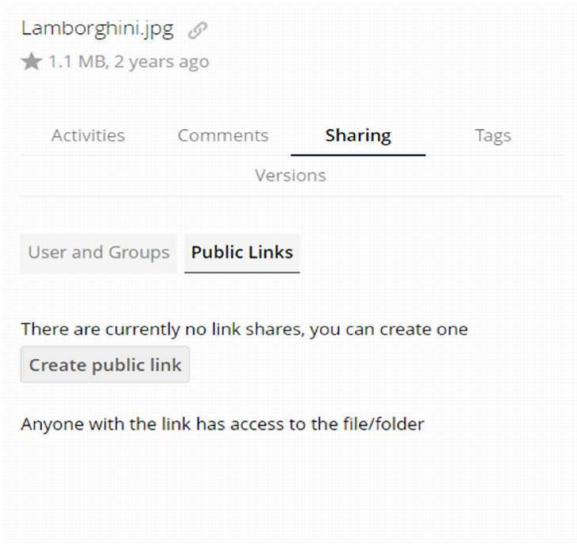
Step 2: Click on “+” > Upload File and upload any file you want from your system.



Step 3: It will take few seconds to upload the file. Once uploaded, you will see it as added at bottom.



Step 4: Click on the file. A menu will open on right hand side of the web page. Go to Sharing window.



Step 5: Click on Create public link. A window will open, give the name of your link.

×

Create link share: /Lamborghini.jpg

Link name

Omkar's Link

☒ Download / View

Recipients can view or download contents.

☐ Download / View / Edit

Recipients can view, download and edit contents.

Password

Choose a password

Expiration

Choose an expiration date

Cancel

Share

Step 6: Now a link will be created at right hand side menu. You can now share it across internet via email or any other social media platform.

Lamborghini.jpg

★ 1.1 MB, 2 years ago

Activities

Comments

Sharing

Tags

Versions

User and Groups

Public Links

Copy to clipboard

🔗 Omkar's Link

📎

⚙️

➦

🗑️

Create public link

Anyone with the link has access to the file/folder

about:blank

31/31