

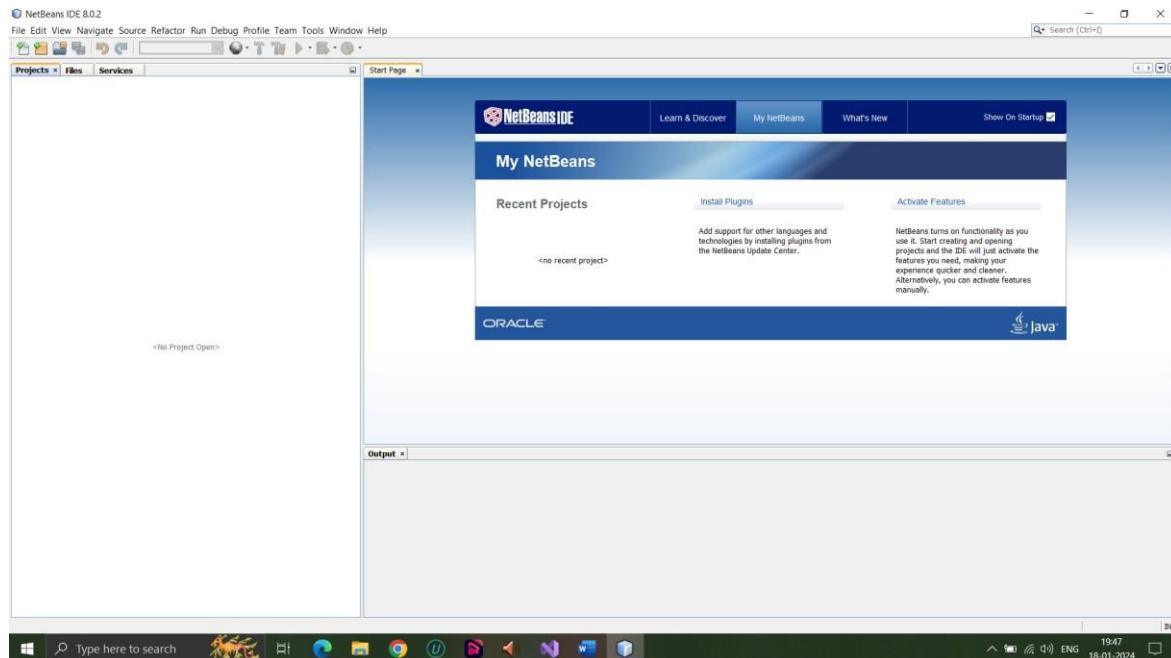
TYCS SEM-VI Cloud Computing and Webservices Journal

Practical-1

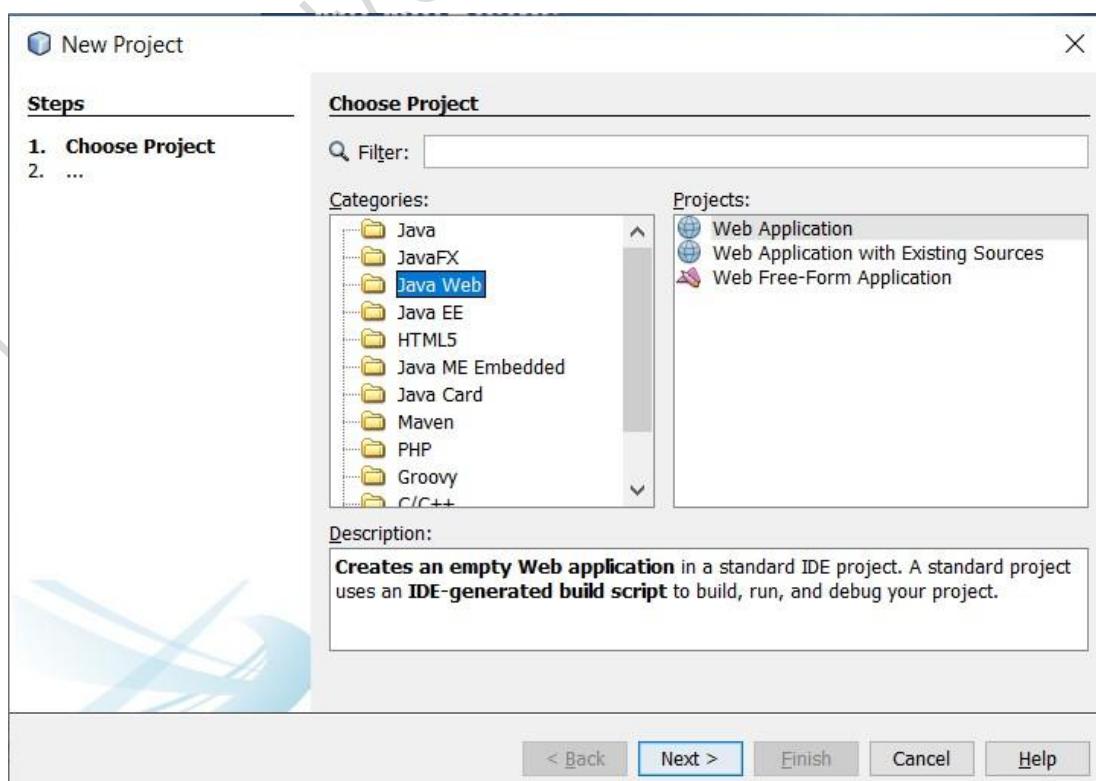
Create a Simple SOAP service.

Steps:

- 1] Open the NetBeans , and you will get the following screen. Close the start page.

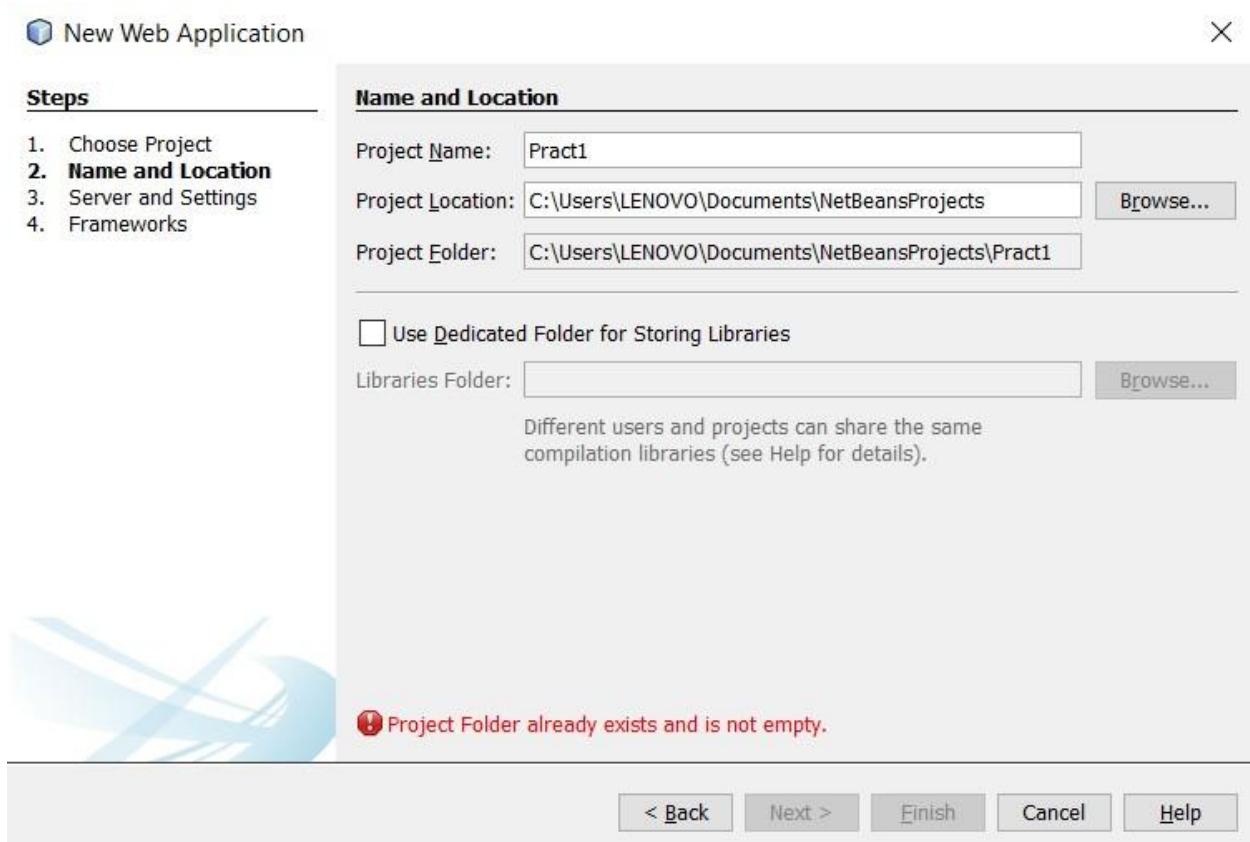


- 2] Now click on the file tab and click on new project you will get the following screen :

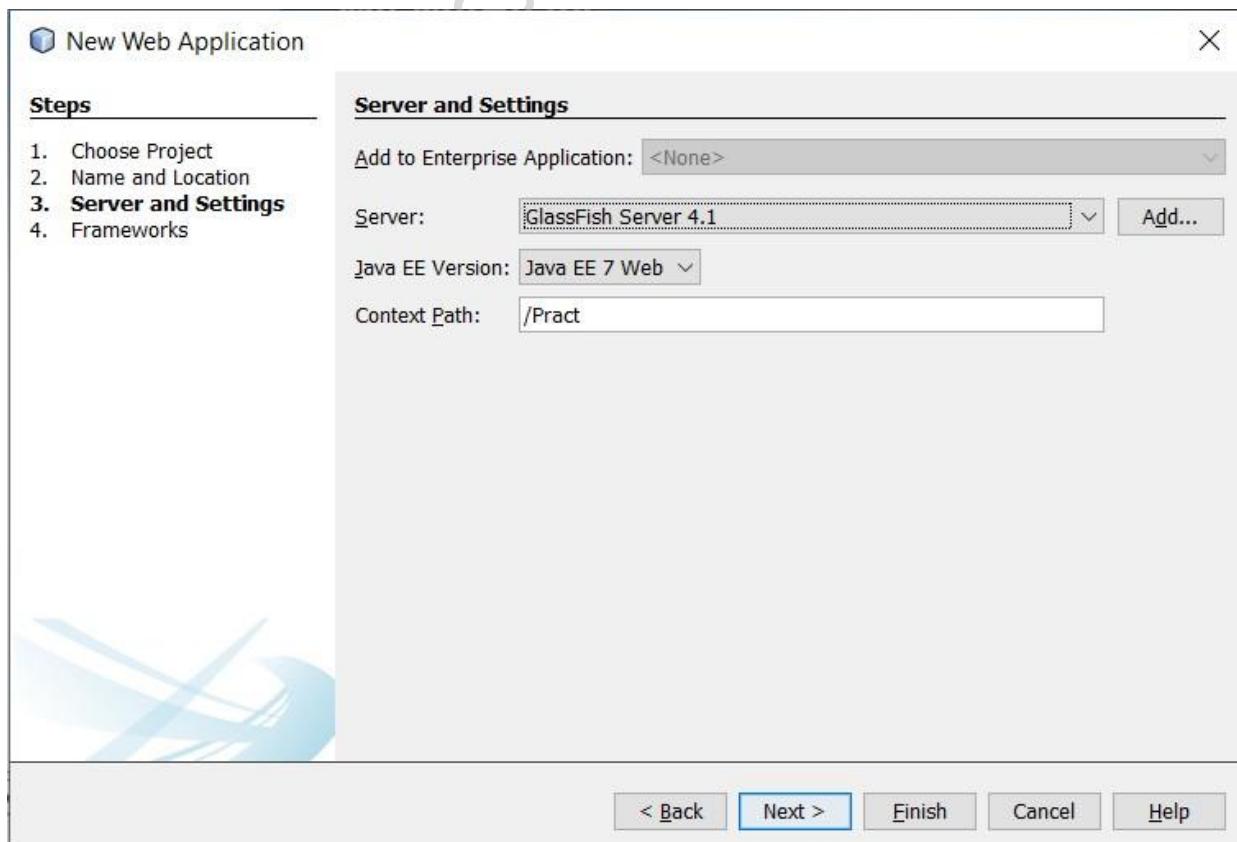


TYCS SEM-VI Cloud Computing and Webservices Journal

3] In Categories select Java Web and in Projects , Select Web Application .After selecting click on next . You will get the following window:

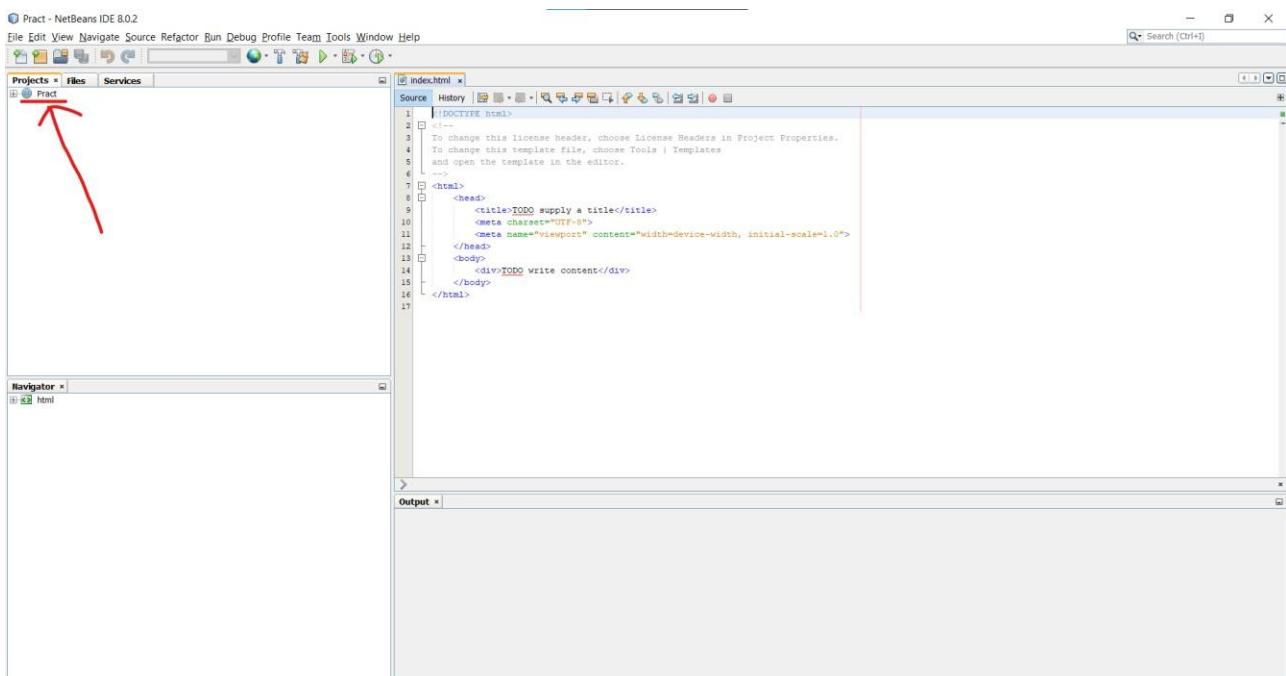


4] Now Give name to the Project Name:, and click on next . you will get the following window then
. again click on finish:

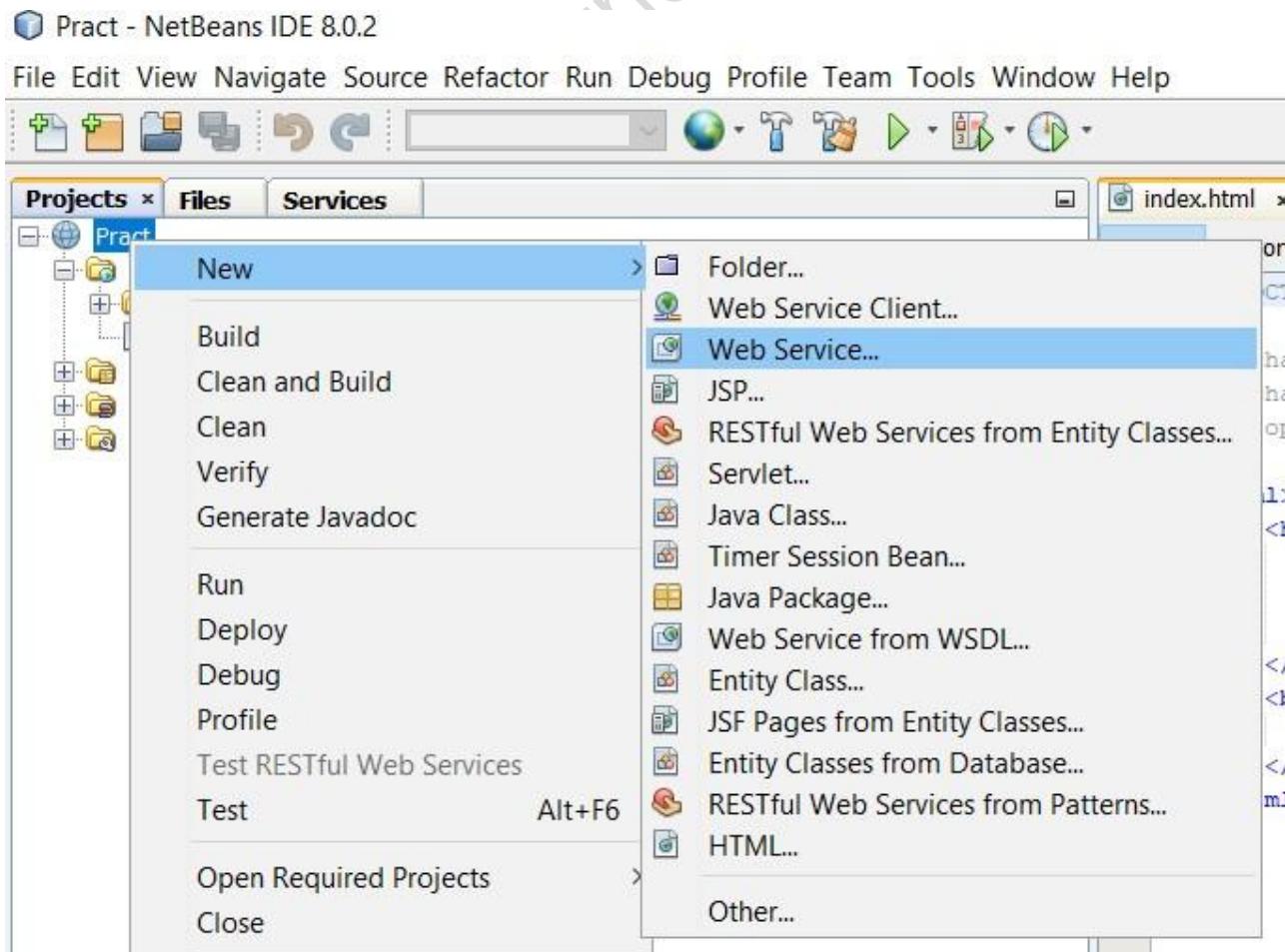


TYCS SEM-VI Cloud Computing and Webservices Journal

5] You will get the following screen now carefully see in projects section your recently created project appears double click to expand it:

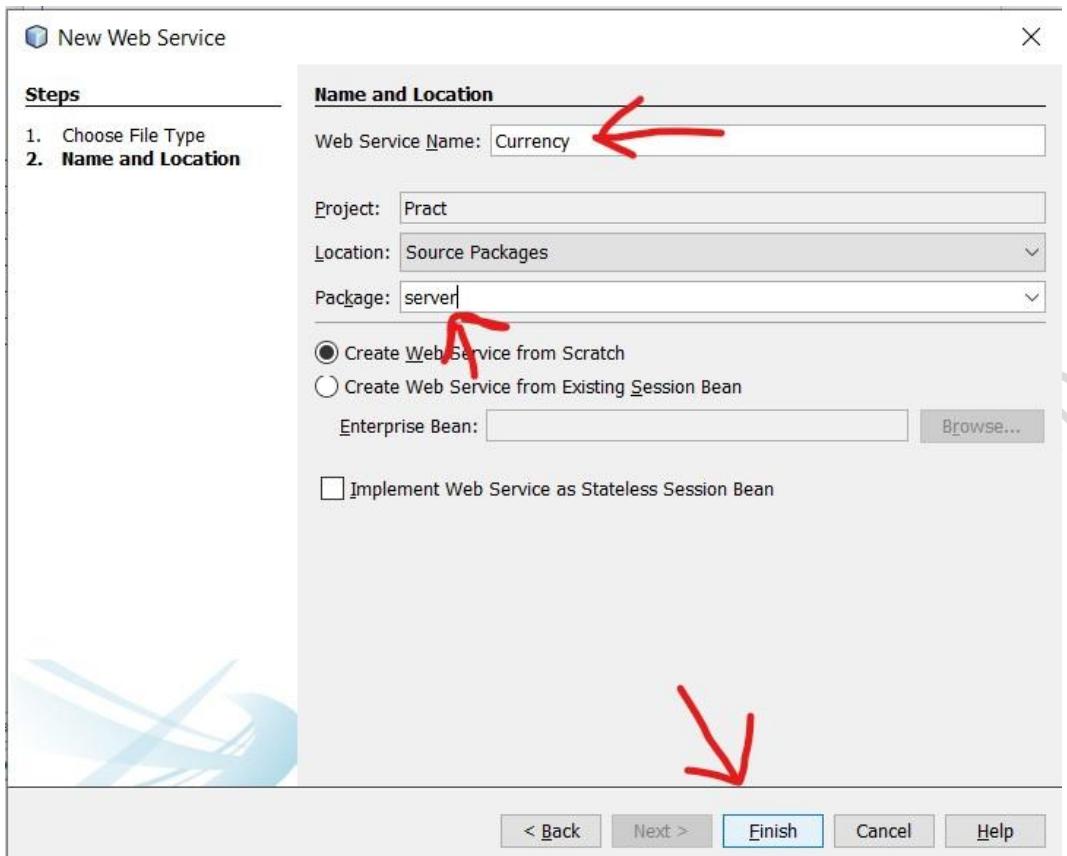


6] Now Right click on the project and select new and then select Web Service ,As shown Below:



TYCS SEM-VI Cloud Computing and Webservices Journal

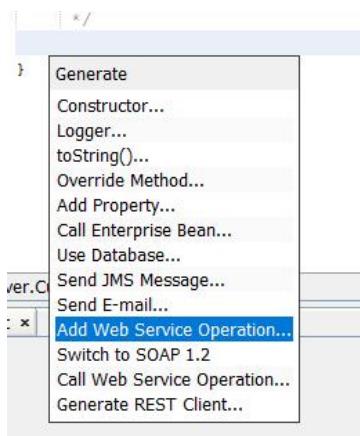
7] after clicking Web Service following window should appear , now give name to web service and give package as "server". As shown in the image:



After clicking finish you should get the following window erase the mentioned code :

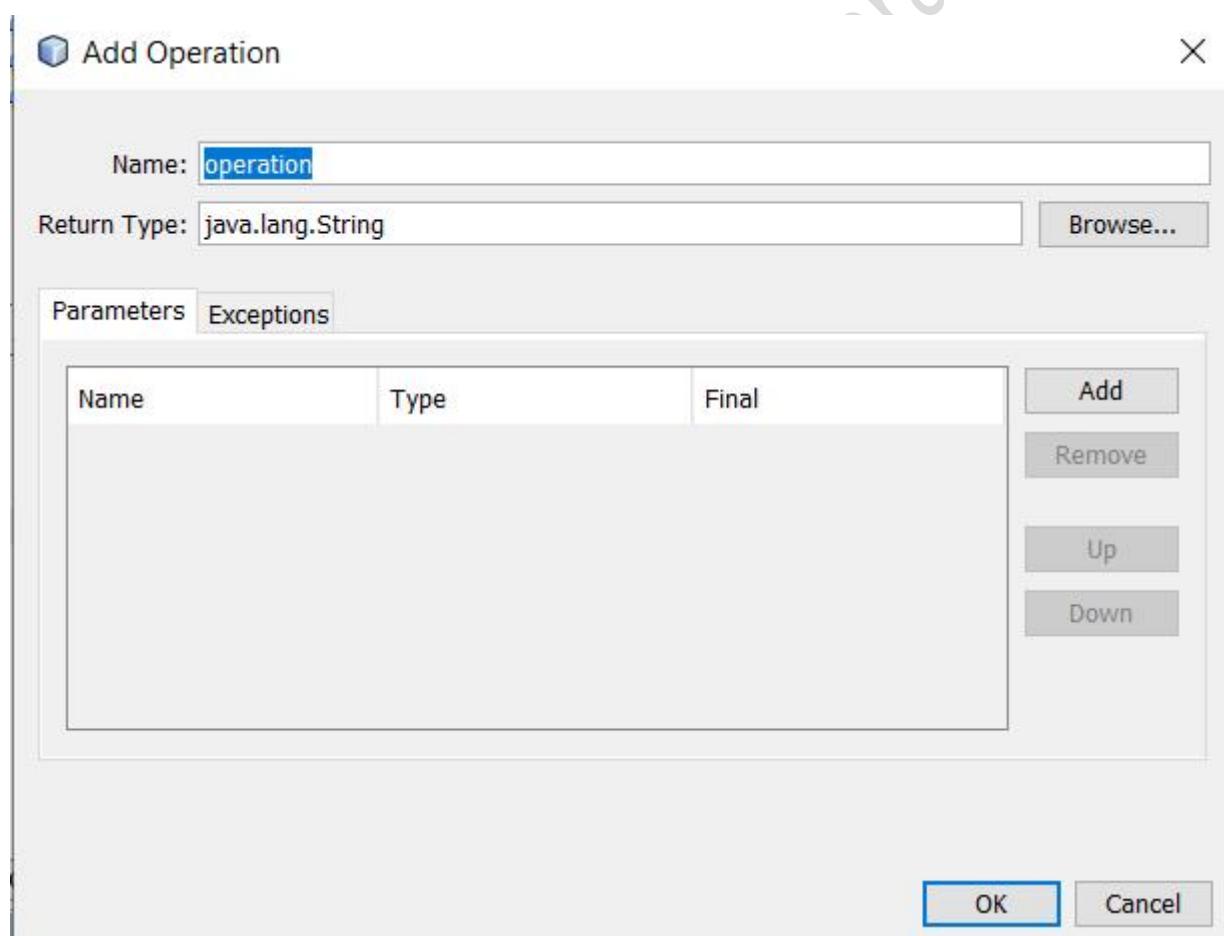
```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package server;
7
8  import javax.jws.WebService;
9  import javax.jws.WebMethod;
10 import javax.jws.WebParam;
11
12 /**
13  *
14  * @author LENOVO
15  */
16 @WebService(serviceName = "Currency")
17 public class Currency {
18
19     /**
20      * This is a sample web service operation
21      */
22     @WebMethod(operationName = "hello")
23     public String hello(@WebParam(name = "name") String txt) {
24         return "Hello " + txt + " !";
25     }
26 }
```

8] Now right click any where and click on insert code and select Add Web Service Operation :

TYCS SEM-VI Cloud Computing and Webservices Journal

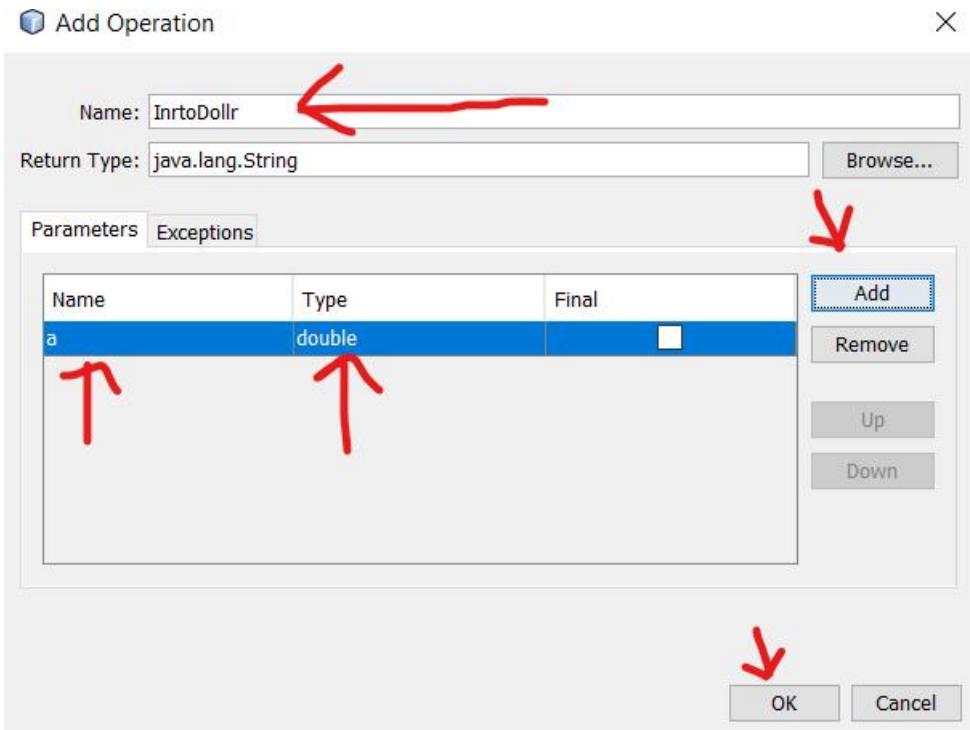
9] the following window would appear :

Just give name to the method or operation and click on add button to add parameters to the method as here we are converting dollar to rupees we should need only one parameter .



10] give the name to the variable select data type as double and click on ok as shown below:

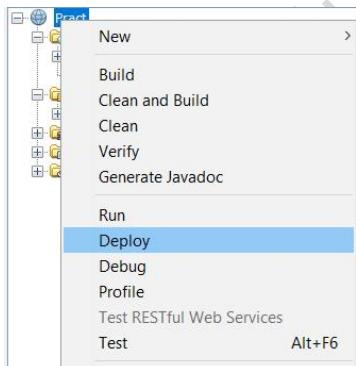
TYCS SEM-VI Cloud Computing and Webservices Journal



11] After clicking ok code will autogenerate , make changes In that code as mentioned below:

```
@WebMethod(operationName = "InrtoDollar")
public String InrtoDollar(@WebParam(name = "a") double a) {
    //TODO write your implementation code here:
    return "The Indian rupees "+a+" in Dollars is "+(a/83.17);
}
```

12] now our web service is ready now right click on project and click on deploy :



13] now right click on webservice and click on test web service you will get the following output:

TYCS SEM-VI Cloud Computing and Webservices Journal

WSDL File)'. It shows a code snippet for the 'inrtoDollar' method."/>

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 server.CurrencyConvertor.inrtoDollar(double)
inrtoDollar ( )
```

Method invocation trace

localhost:8080/practical1/CurrencyConvertor?Tester

inrtoDollar Method invocation

Method parameter(s)

Type	Value
double	1233

Method returned

java.lang.String : "The Indian rupees 1233.0 in Dollars is 14.825057111939401"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
<ns2:InrtoDollar xmlns:ns2="http://server/">
<a>1233.0</a>
</ns2:InrtoDollar>
</S:Body>
</S:Envelope>
```

SOAP Response

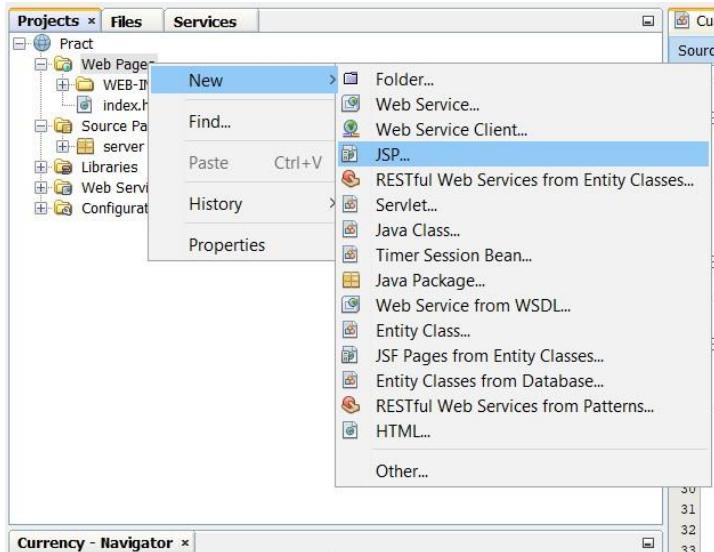
```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
<ns2:InrtoDollarResponse xmlns:ns2="http://server/">
<return>The Indian rupees 1233.0 in Dollars is 14.825057111939401</return>
</ns2:InrtoDollarResponse>
</S:Body>
</S:Envelope>
```

So this is how we created our web service and deployed it.

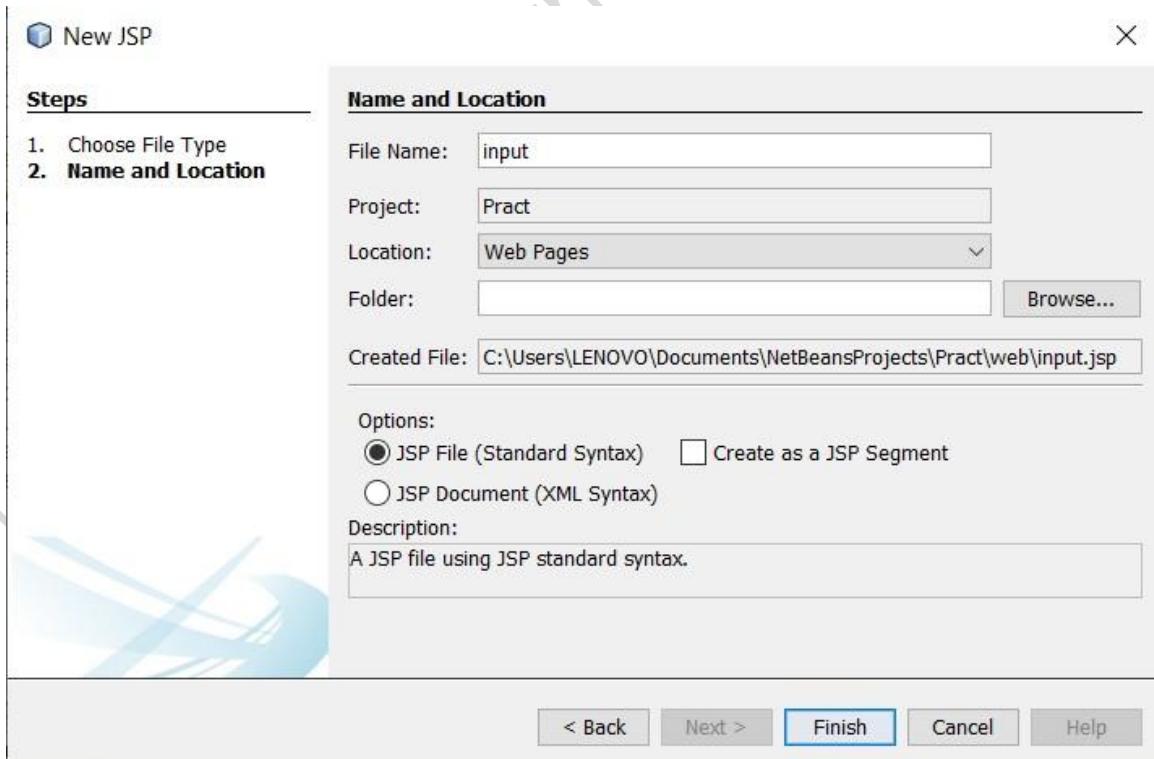
TYCS SEM-VI Cloud Computing and Webservices Journal

Creating a java Client using jsp

14] now our web service is successfully deployed. Right click on web pages and select new and select jsp as shown below:



15] give name and click on finish as shown below do it 2 times on for input and one for output:



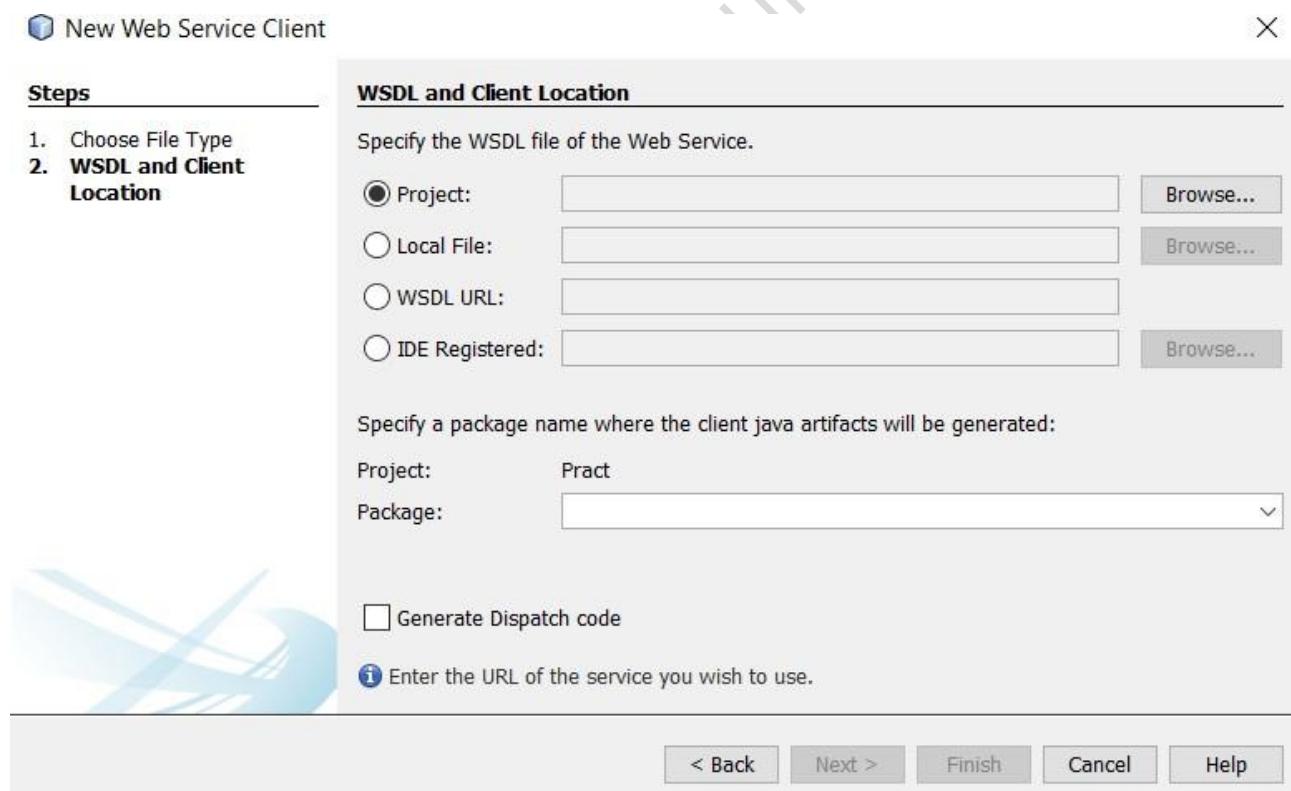
16] In input.jsp create a form for taking user input as shown below:

TYCS SEM-VI Cloud Computing and Webservices Journal

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <form action="output.jsp">
            <pre>
                Enter the currency in rupees : <input type="text" name="t1">
                <input type="submit"> <input type="reset">
            </pre>
        </form>
    </body>
</html>
```

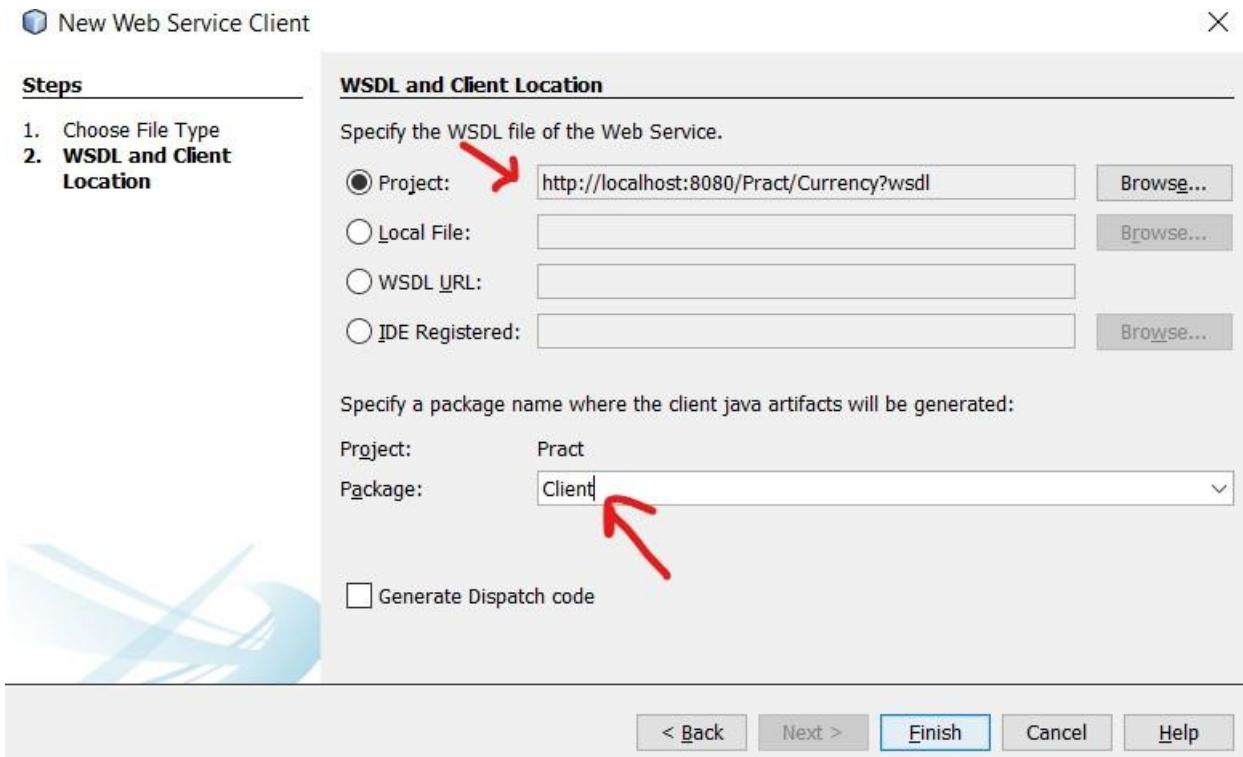
In this code set action = the jsp file where u want output and give name to textbox input.

17] now we have to create web service client, for same right click on project and select new then select web service client you will get the following screen:



At this step click on browse and select your project and click ok after clicking ok you will get wsdl url as shown below:

TYCS SEM-VI Cloud Computing and Webservices Journal



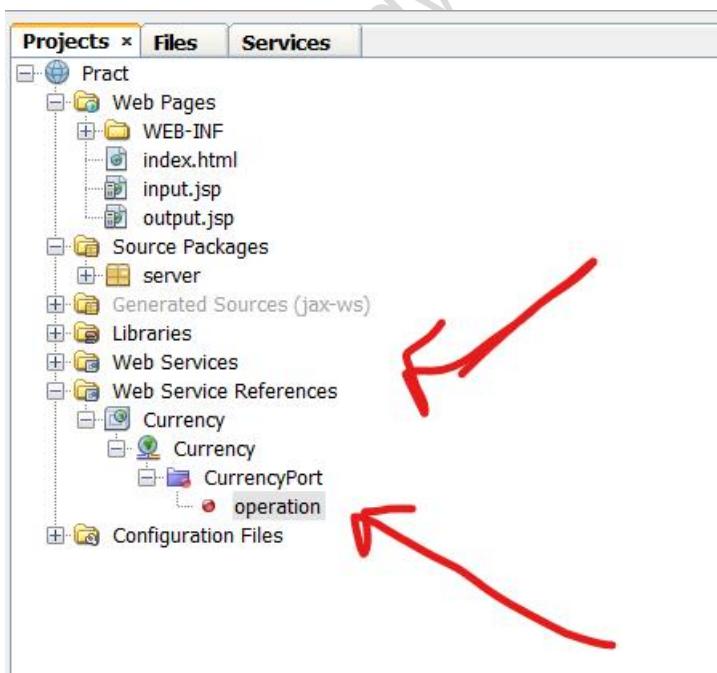
At this stage copy the url and paste it in notepad for future.

Give package name as "Client".

Now click on finish.

18] now in project section you can see a new folder is been created named "web service reference".

double click to expand it you should get the following :



TYCS SEM-VI Cloud Computing and Webservices Journal

Hold the operation or your operation name and drag it into the output.jsp in body tag as shown

: You will get the following auto generated code:

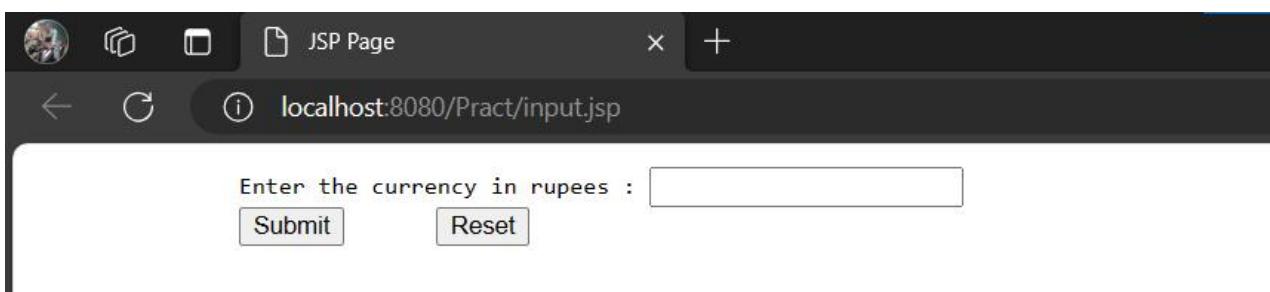
```
<body>
    <%-- start web service invocation --%><hr/>
<%
try {
    Client.Currency_Service service = new Client.Currency_Service();
    Client.Currency port = service.getCurrencyPort();
    // TODO initialize WS operation arguments here
    double a = 0.0d;
    // TODO process result here
    java.lang.String result = port.operation(a);
    out.println("Result = "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
<%-- end web service invocation --%><hr/>

</body>
```

19] now make changes as shown below in the code:

```
<%-- start web service invocation --%><hr/>
<%
try {
    Client.Currency_Service service = new Client.Currency_Service();
    Client.Currency port = service.getCurrencyPort();
    // TODO initialize WS operation arguments here
    double a = Double.parseDouble(request.getParameter("t1"));
    // TODO process result here
    java.lang.String result = port.operation(a);
    out.println(result); [Red arrow pointing to this line]
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
<%-- end web service invocation --%><hr/>
```

20] now Deploy our project again and right click on input.jsp and click run file you will redirect to a browser page as shown :



TYCS SEM-VI Cloud Computing and Webservices Journal

Enter any numerical value in text box and click on submit you will get the following output:



TYCS SEM-VI Cloud Computing and Webservices Journal

Python client:

1] for consuming java web services in python we should repeat the above steps till step np.19 .

Note: for getting output the project or web service should be deployed .

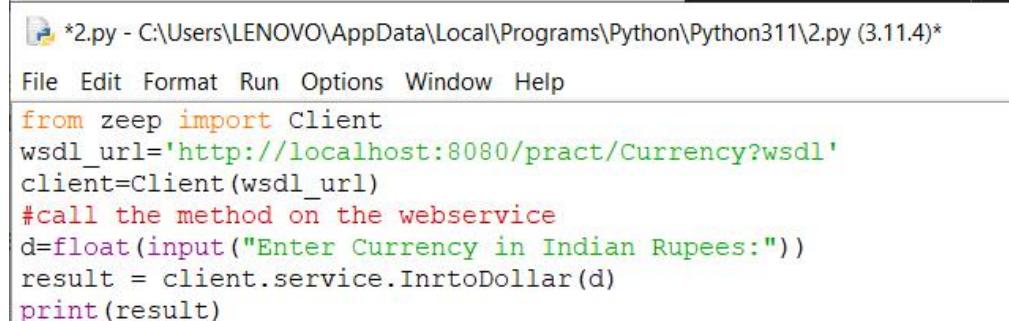
After doing all of the steps write the following code in python idle:

Code :

```
from zeep import Client
wsdl_url='http://localhost:8080/pract/Currency?wsdl'
client=Client(wsdl_url)
#call the method on the webservice
d=float(input("Enter Currency in Indian Rupees:"))
result = client.service.InrtoDollar(d)
print(result) ] replace wsdl_url with the url which was copied at the time of web service client creation .
```

In step no.17 . and replace the **InrtoDollar** with your method or operation name. and pass parameter to it

3] final code should look like following:



```
*2.py - C:\Users\LENOVO\AppData\Local\Programs\Python\Python311\2.py (3.11.4)*
File Edit Format Run Options Window Help
from zeep import Client
wsdl_url='http://localhost:8080/pract/Currency?wsdl'
client=Client(wsdl_url)
#call the method on the webservice
d=float(input("Enter Currency in Indian Rupees:"))
result = client.service.InrtoDollar(d)
print(result)
```

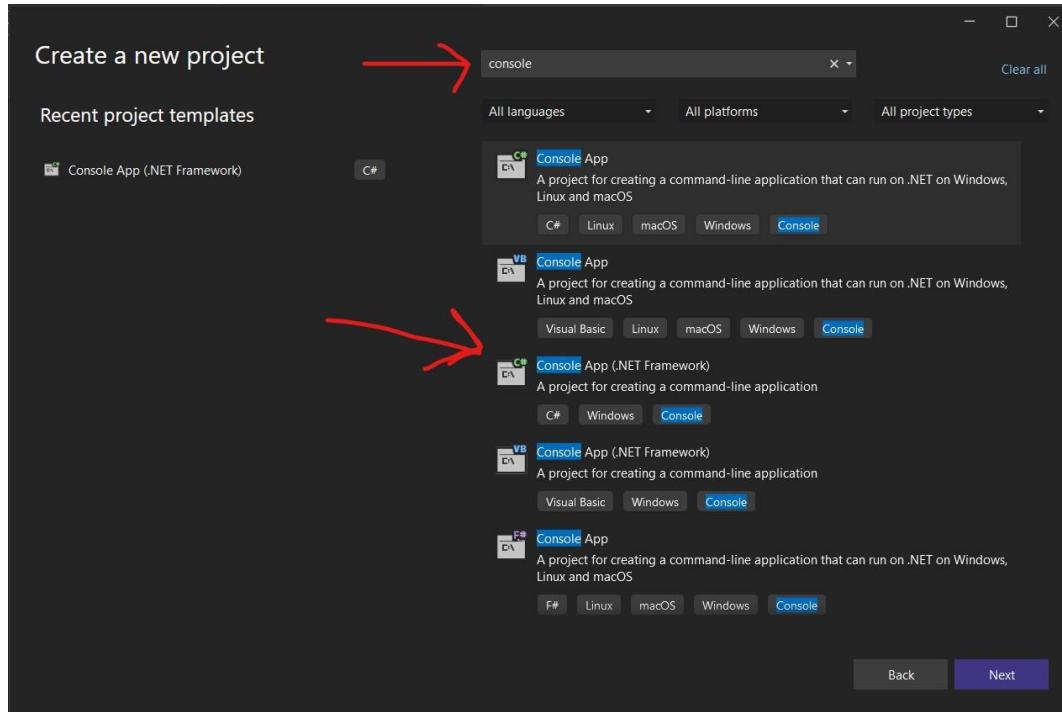
4] after running the above code you should get the following output:

```
= RESTART: C:\Users\LENOVO\AppData\Local\Programs\Python\Python311\2.py
Enter Currency in Indian Rupees:12000
The Indian rupees 12000.0 in Dollars is 144.28279427678248
```

TYCS SEM-VI Cloud Computing and Webservices Journal

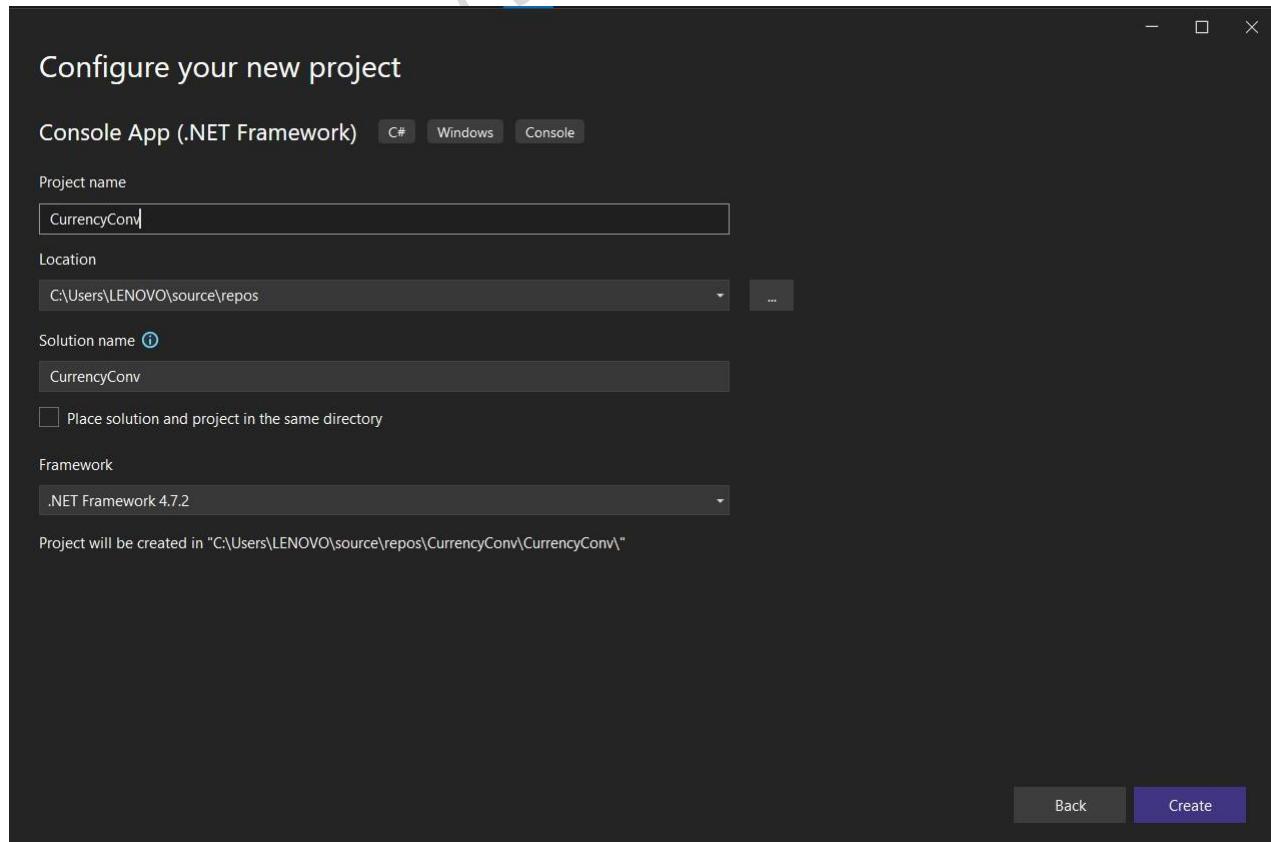
.Net Client

1] open visual studio and select create a new project the following window will appear:

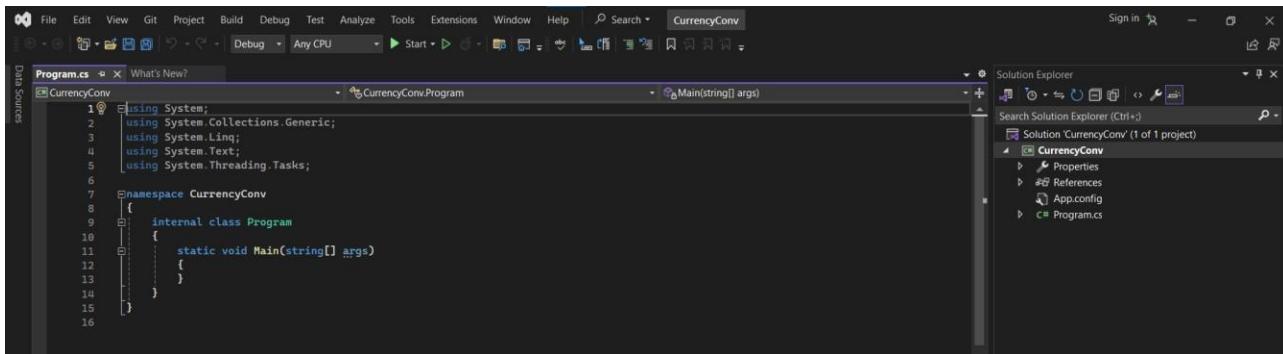


Here in search bar type “console c#” and select console app for .Net frame work as shown in above image.

2] Give name to the project and press create button as shown in the image:



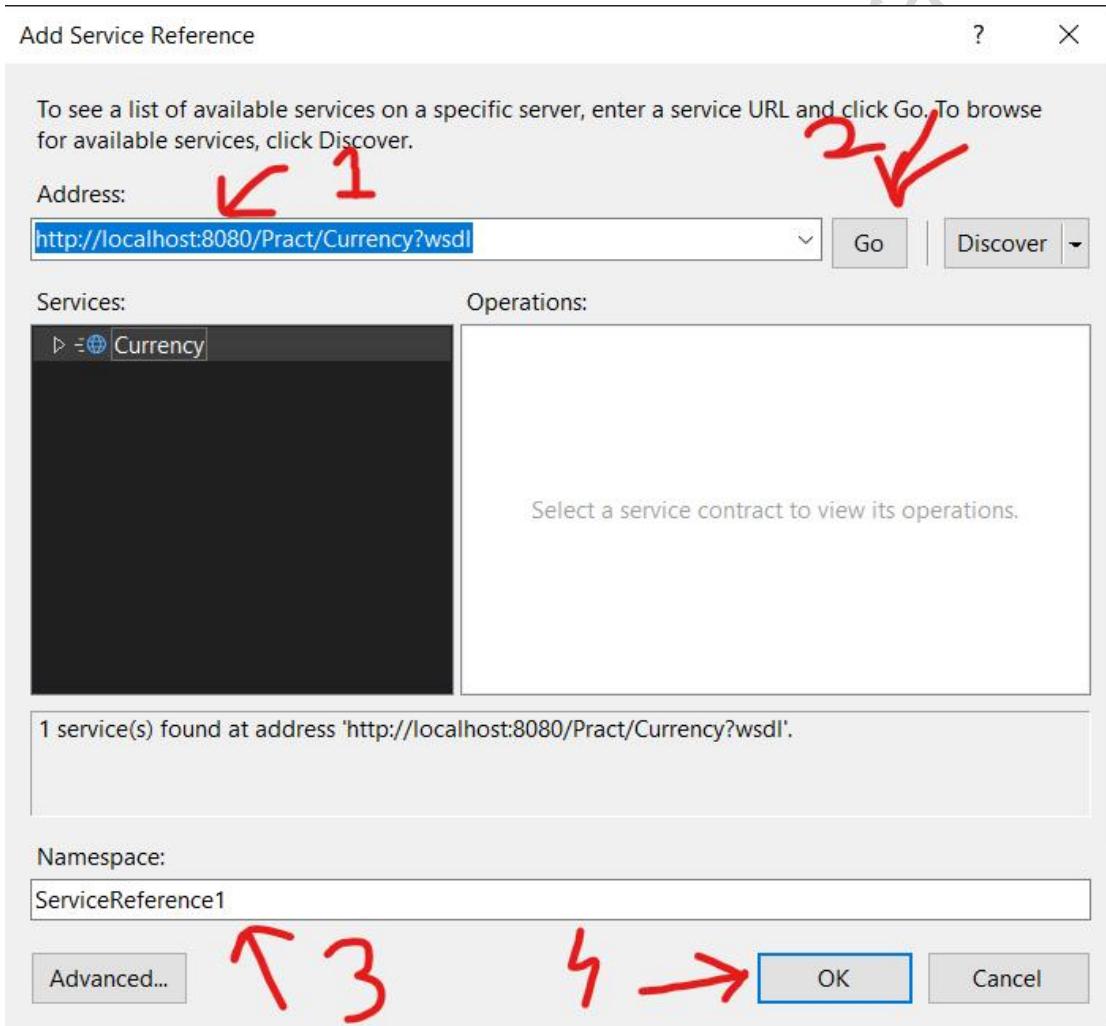
TYCS SEM-VI Cloud Computing and Webservices Journal



This window should appear after clicking create button now In right side there is solution Explorer.

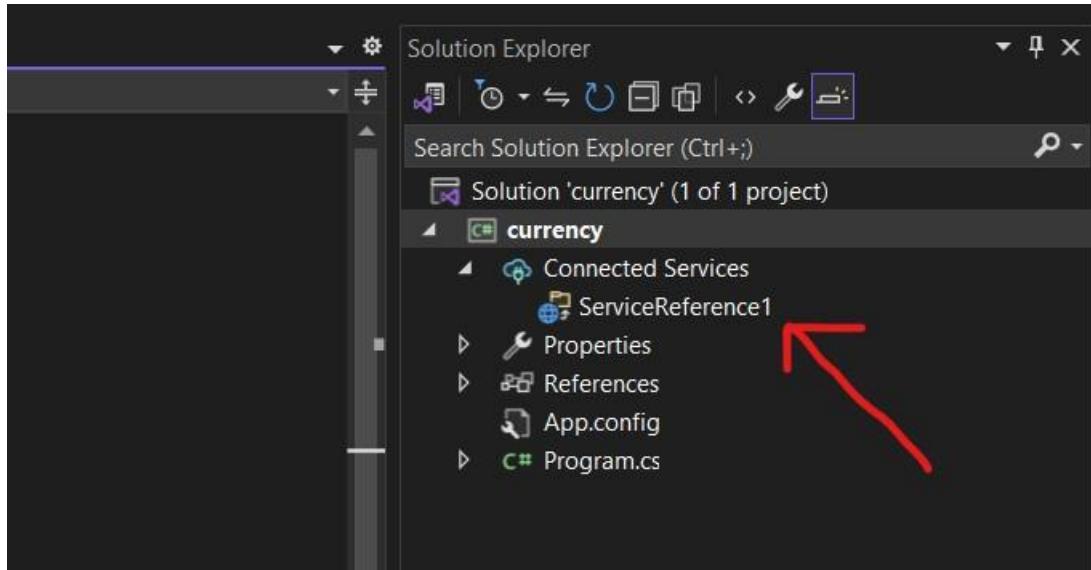
In that right click on your project name and click on add > Service Reference . You will get the following window :

Note: before executing the code make sure that the web service is deployed.



Here in address bar paste the wsdl url the same which is used I python code. And press go button . agter set the namespace In my case I am letting is as it is then press ok button.

In following image u can check in solution Explorer service refrence is created.:.

TYCS SEM-VI Cloud Computing and Webservices Journal

3] now in code write the following line as shown in image:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using CurrencyConv.ServiceReference1;
7
8  namespace CurrencyConv
9  {
10     internal class Program
11     {
12         static void Main(string[] args)
13         {
14         }
15     }
16 }
17
```

Here type:

Using your_projectname . your_serviceReference_name ;

4] now write the code as shown in the following image:

TYCS SEM-VI Cloud Computing and Webservices Journal

```
namespace CurrencyConv
{
    internal class Program
    {
        static void Main(string[] args)
        {
            CurrencyClient client = new CurrencyClient();
            Console.WriteLine("Enter the Currency in Indian Rupees : ");
            double d = double.Parse(Console.ReadLine());
            Console.WriteLine(client.InrtoDollar(d));
            Console.WriteLine("Enter any key to Exit...");
            Console.ReadKey();
            Console.ReadLine();
        }
    }
}
```

Note:

Here **CurrencyClient** is my webservice class name and **operation()** is my method name as show you can also check your own:



5] when you run the above code you get the following output:

```
C:\Windows\system32\cmd.exe
Enter the Currency in indian rupees:
1000
The Indian rupees 100.0 in Dollars is 1.2023566189731874
Enter any key to Exit...
```

Practical-2

Define a simple services like Converting Rs into Dollar and Call it from different platform like JAVA and .NET

Steps:

Note : for all these steps in brief refer practical no.1 document

- 1] create a new project and create a new webservice
- 2] now right click on the code and click on insert code > add new web service operation
- 3] after give name to operation **FtoC** and add one parameter **a** of double data type and click ok the code will auto generate .
- 4] now do the following changes in that auto generated code:

```
/*
@WebMethod(operationName = "FtoC")
public String FtoC(@WebParam(name = "a") double a) {
    //TODO write your implementation code here:
    return "The Fahrenheit Temperature "+a+" in Celsius is "+((a-32)*5/9);
}
```

Now deploy the model.

- 5] After deploying the model just right click on your web service and click on test web service . and you will get redirected to a browser page where you can check that your web service is working or not.

Creating java Client using jsp

- 5] now right click on web pages > new > jsp . and give name as input.jsp.

Again right click on web pages > new > jsp . and give name as output.jsp.

- 6] In input.jsp write the following code :

TYCS SEM-VI Cloud Computing and Webservices Journal

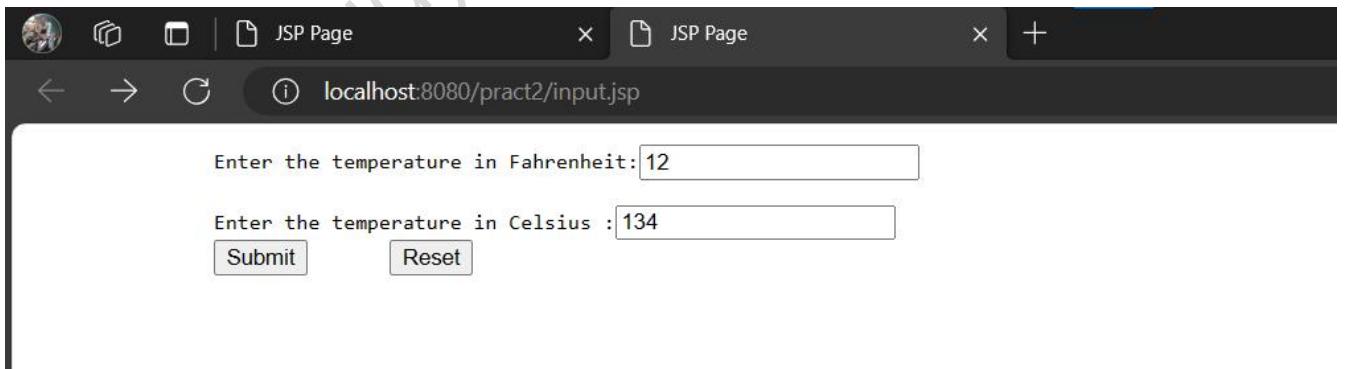
```
<body>
    <form action="output.jsp">
        <pre>
            Enter the temperature in Fahrenheit:<input type="text" name="t1" required>

            Enter the temperature in Celsius :<input type="text" name="t2" required>
            <input type="submit">      <input type="reset">
        </pre>
    </form>
</body>
```

7] create a Web service client and give package name as “Client”. After creating web service client go to web service references and drag and drop the operations in output.jsp in body tag as shown below:

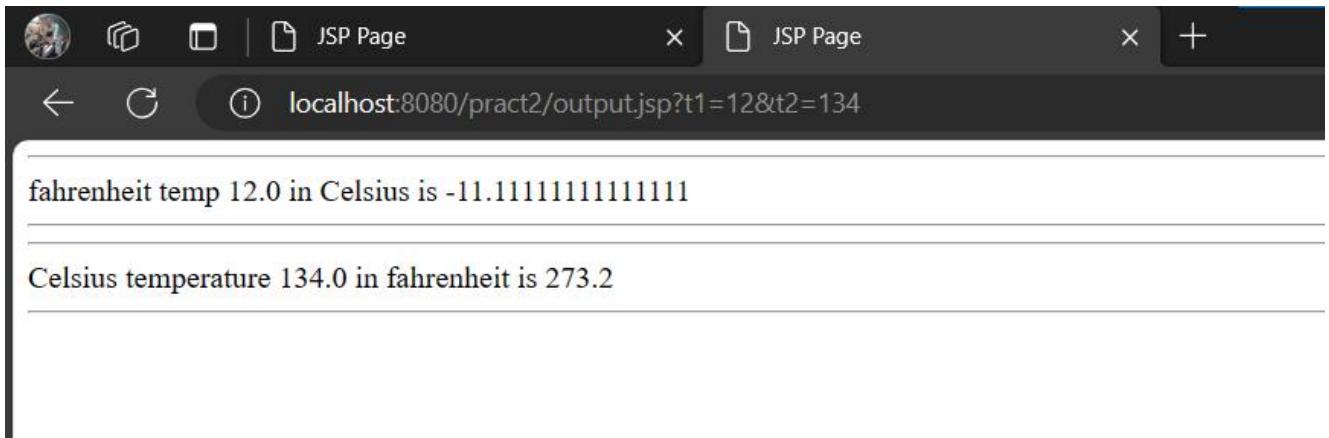
```
<%-- start web service invocation --%><hr/>
<%
try {
    client.Conv_Service service = new client.Conv_Service();
    client.Conv port = service.getConvPort();
    // TODO initialize WS operation arguments here
    double a = Double.parseDouble(request.getParameter("t1"));
    // TODO process result here
    java.lang.String result = port.FtoC(a);
    out.println(result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
%>
<%-- end web service invocation --%><hr/>
```

8] now deploy the model again and run the input.jsp file you will get redirected to the browser page as shown:



After submitting you will get the following output:

TYCS SEM-VI Cloud Computing and Webservices Journal



Python Client

1] Deploy the web service and write the following code :

Code :

```
from zeep import Client
wsdl_url='http://localhost:8080/Practical_2/TempConverter?wsdl'
client=Client(wsdl_url)
#call the method on the webservice
d=float(input("Enter temperaure in Fahrenheit :"))
result = client.service.FtoC(d)
print(result)
```

2] *replace your_wsdl_url with the url which was copied at the time of web service client creation .*

In step no.16 . and replace the operation_name with your method or operation name. and pass parameter to it

3] *final code should look like following:*

TYCS SEM-VI Cloud Computing and Webservices Journal



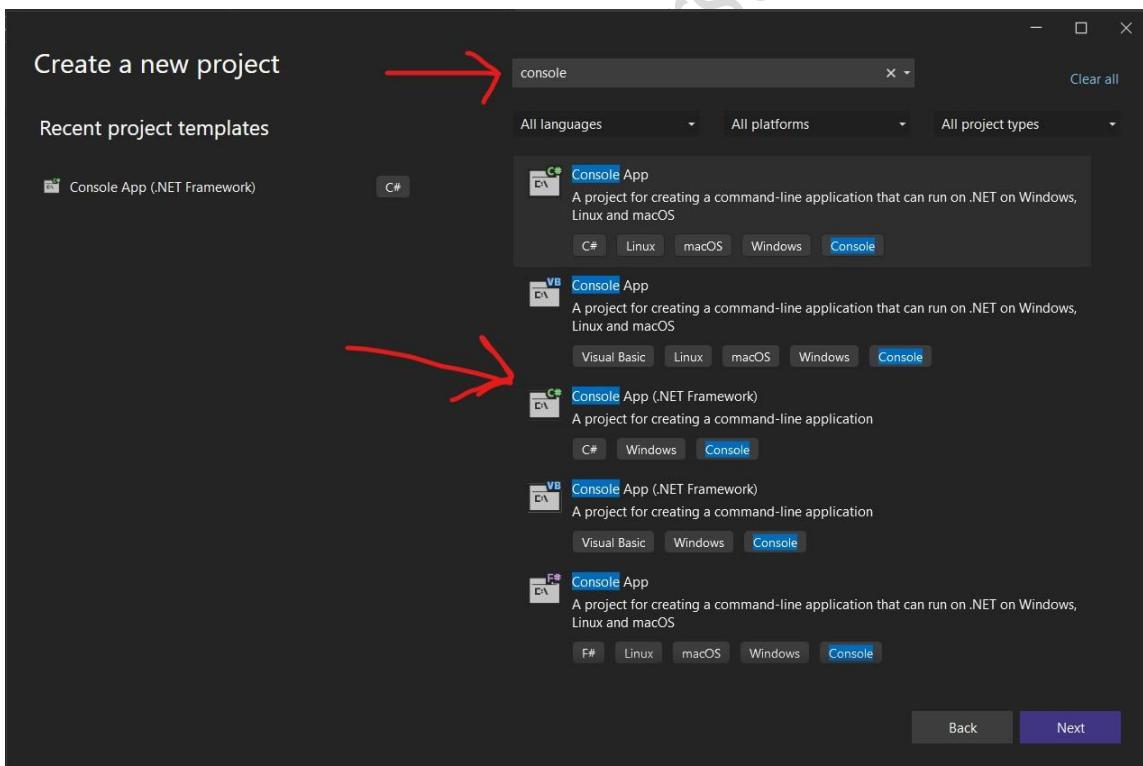
```
*2.py - C:\Users\LENOVO\AppData\Local\Programs\Python\Python311\2.py (3.11.4)*
File Edit Format Run Options Window Help
from zeep import Client
wsdl_url='http://localhost:8080/Practical_2/TempConverter?wsdl'
client=Client(wsdl_url)
#call the method on the webservice
d=float(input("Enter temperaure in Fahrenheits :"))
result = client.service.FtoC(d)
print(result)
```

4] after running the above code you should get the following output:

```
===== RESTART: C:\Users\LENOVO\AppData\Local\Programs\Python\Python311\2.py
Enter temperaure in Fahrenheits :1200
The Fahrenheits Temperature 1200.0 in Celsius is 648.888888888889
```

.Net Client

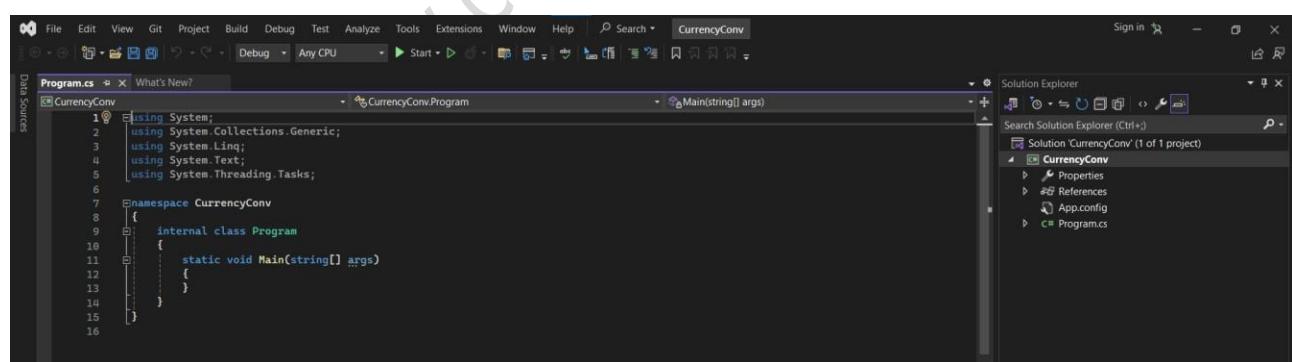
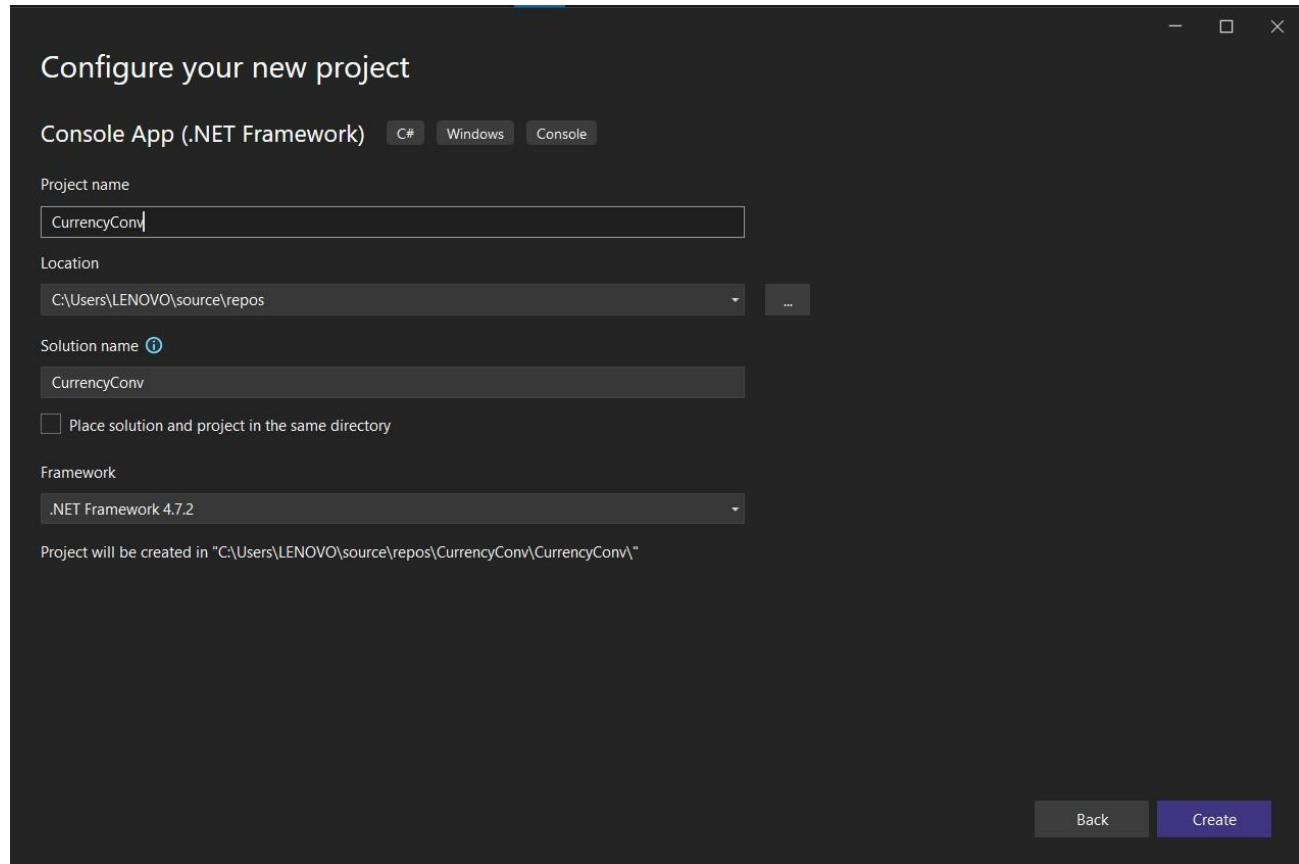
1] open visual studio and select create a new project the following window will appear:



TYCS SEM-VI Cloud Computing and Webservices Journal

Here in search bar type “console c#” and select console app for .Net frame work as shown in above image.

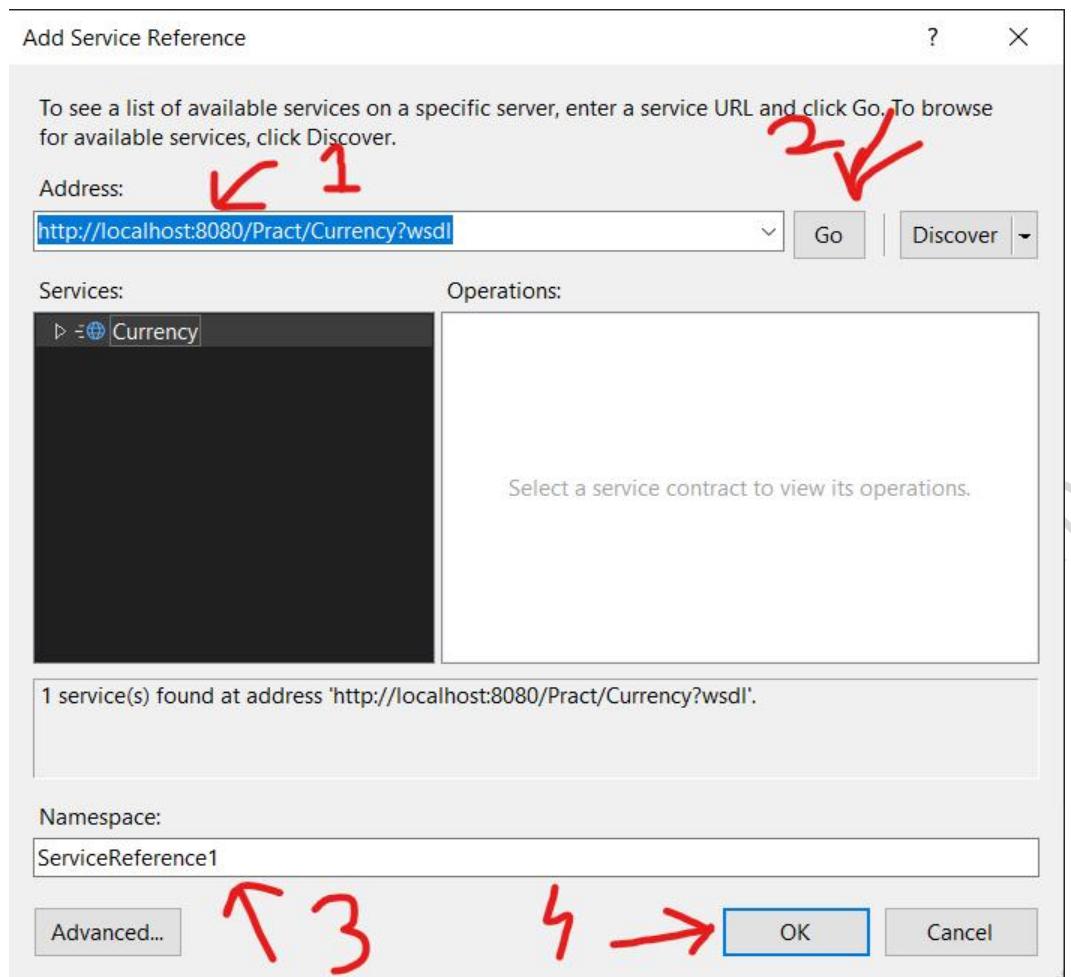
2] Give name to the project and press create button as shown in the image:



This window should appear after clicking create button now In right side there is solution Explorer.

In that right click on your project name and click on add > Service Reference . You will get the following window :

Note: before executing the code make sure that the web service is deployed.

TYCS SEM-VI Cloud Computing and Webservices Journal

Here in address bar paste the wsdl url the same which is used I python code. And press go button . agter set the namespace In my case I am letting is as it is then press ok button

3] now in code write the following line as shown in image:

```
C# CurrencyConv
  1  using System;
  2  using System.Collections.Generic;
  3  using System.Linq;
  4  using System.Text;
  5  using System.Threading.Tasks;
  6  using CurrencyConv.ServiceReference1;
  7
  8  namespace CurrencyConv
  9  {
 10  }
```

Here type:

Using your_projectname . your_serviceReference_name ;

4] now write the code as shown in the following image:

```
static void Main(string[] args)
{
    convClient client = new convClient();
    Console.WriteLine("Enter the Temperature: ");
    double d = double.Parse(Console.ReadLine());
    Console.WriteLine(client.FtoC(d));
    Console.WriteLine("Enter any key to Exit...");
    Console.ReadKey();
    Console.ReadLine();
}
```

Note:

Here my webservice class name is conv and convClient is my webservice client class name and FtoC is method name as shown you can also check your own:

5] when you run the above code you get the following output:

```
C:\Windows\system32\cmd.exe
Enter the Temperature in Fahrenheit :
1200
The Fahrenheit Temperature 1200.0 in Celsius is 648.888888888889
-
```

Practical-03A

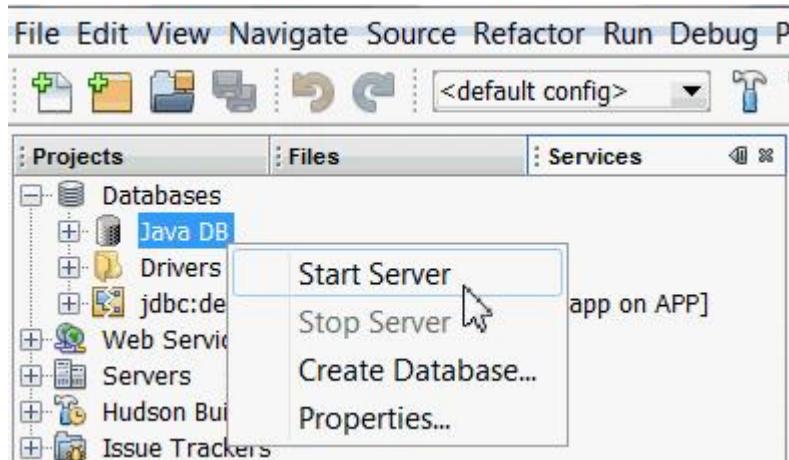
Create a Simple REST Service

- Demonstrate CRUD operations with suitable database using RESTful Web service.

1. To start the Java DB Database from NetBeans, perform the following steps.

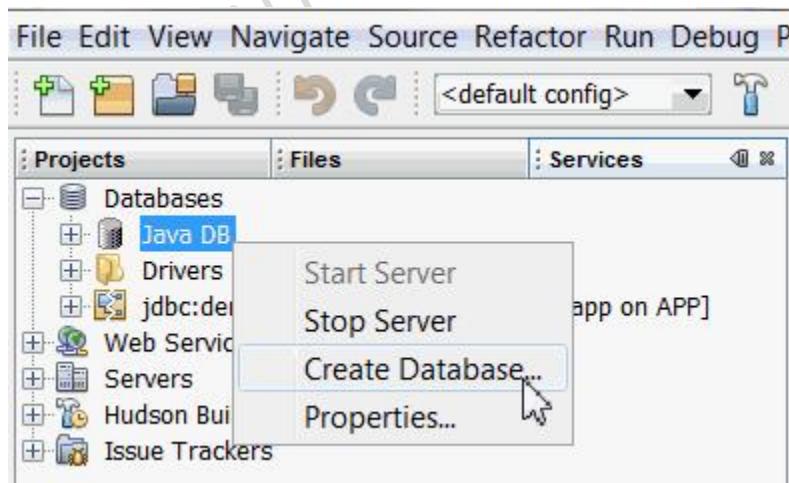
Click Services tab -> Expand Databases node. ->Right-click Java DB icon.

->Select Start Server



2. To create playerDB database, perform the following steps:

a. Right-click Java DB icon, select **Create Database**.



b. Enter the following information for the database:

Database Name: **playerDB**

User Name: **john**

Password: **john**

Confirm Password:**john**

c. Click OK.

3. To connect to the newly created database playerDB, perform the following steps

a. Right-click jdbc:derby://localhost:1527/playerDB connection.

b. Select Connect.

4. Create tables and populate them with data in **playerDB** database.

a. In NetBeans select File > Open File.

b. In the file select playersDB.sql.

c. Click Open. The script automatically opens in the SQL Editor.

5. Examine the contents of the database.

a. In the Services window, expand the jdbc:derby://localhost:1527/playerDB connection under the Databases node.

b. Right-click the connection and select Refresh.

c. Expand the john schema. You see the nodes for Tables, Views, and Procedures. d. Expand the Tables node to see the PLAYER and TEAM tables.

e. Right-click the PLAYER table node and select View Data.

Player server

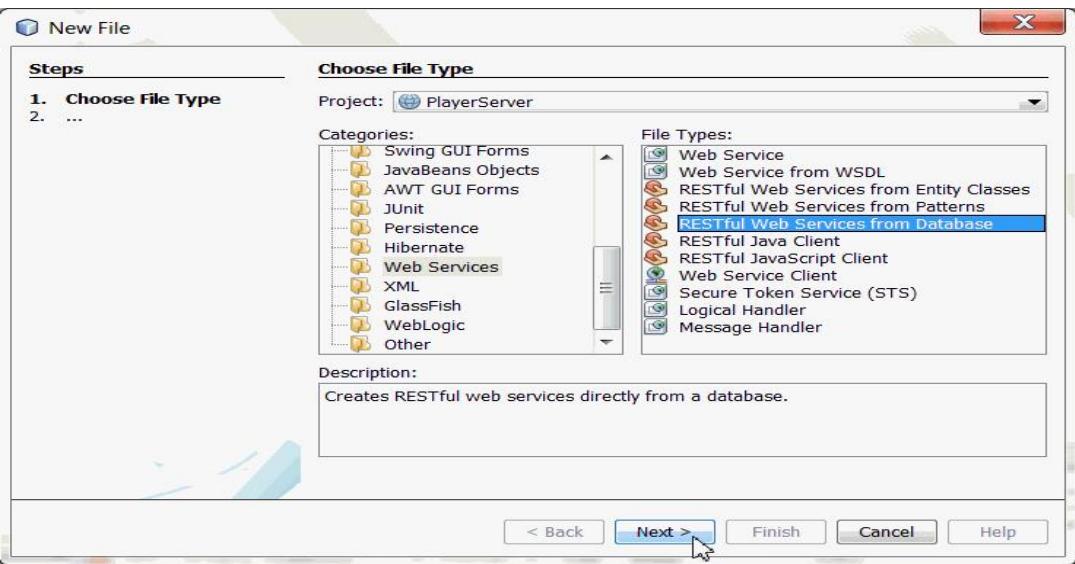
To create RESTful Web Services, you need a Java Web application project. In the below section you will create a demo Java web project, PlayerServer.

To create new Java Web Project, select File -> New Project -> Java web -> Web Application.

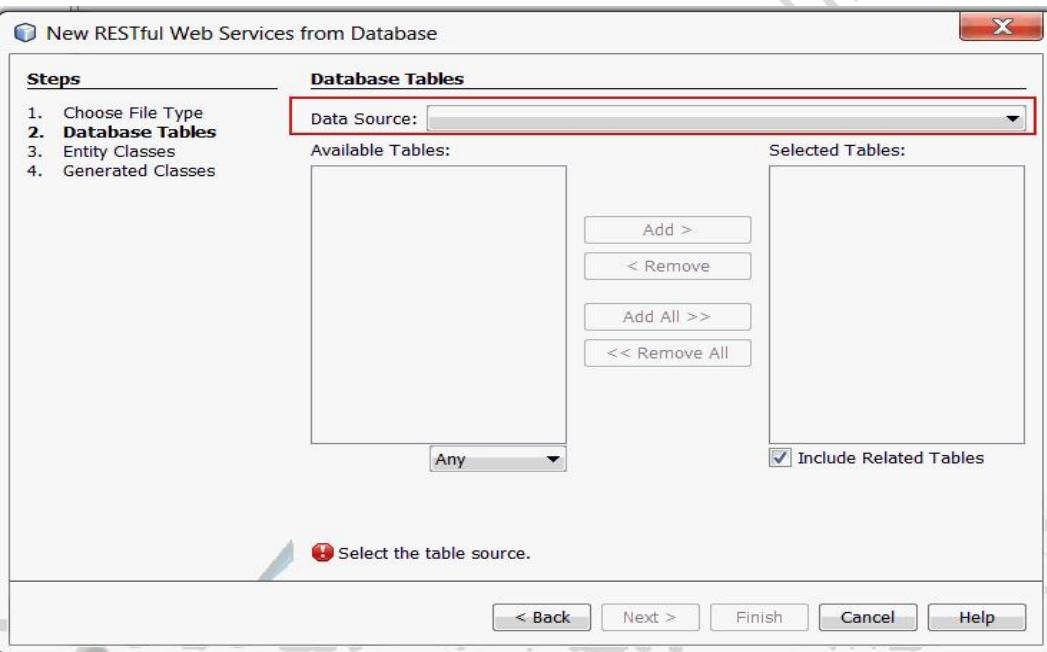
Generating web service

To generate RESTful Web Services do the following:

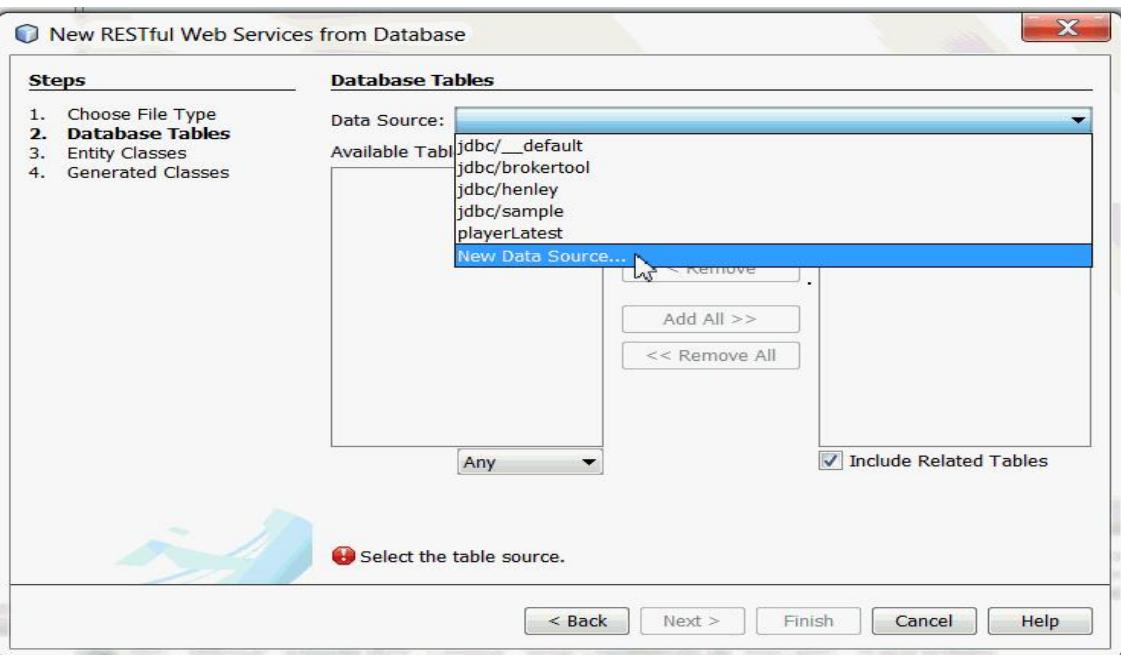
1. Right-click the PlayerServer and choose New > Other > Web Services > RESTful Web Services from Database. The New RESTful Web Service wizard opens on the Database Tables panel.



2. In the Database Tables window, select Data Source.



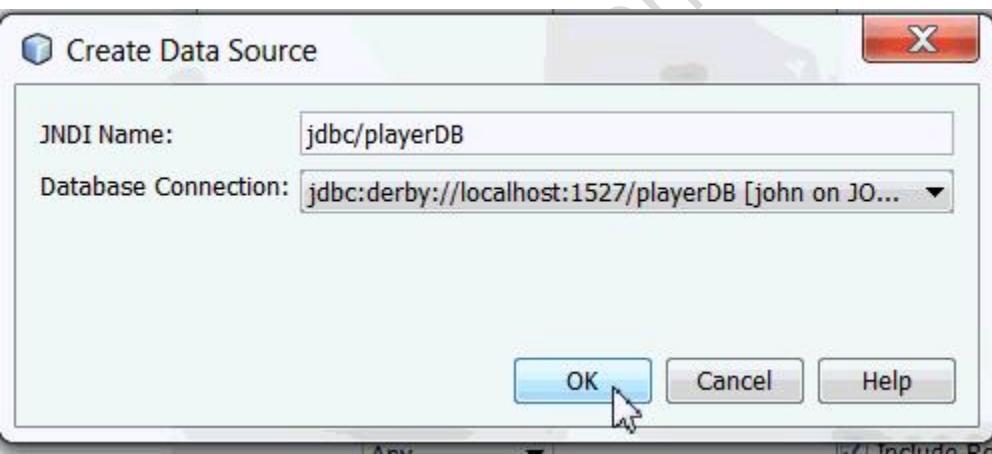
3. Next select "New Data Source" from the Drop-down list.



a. In the Create Data Source Window, enter the following information:

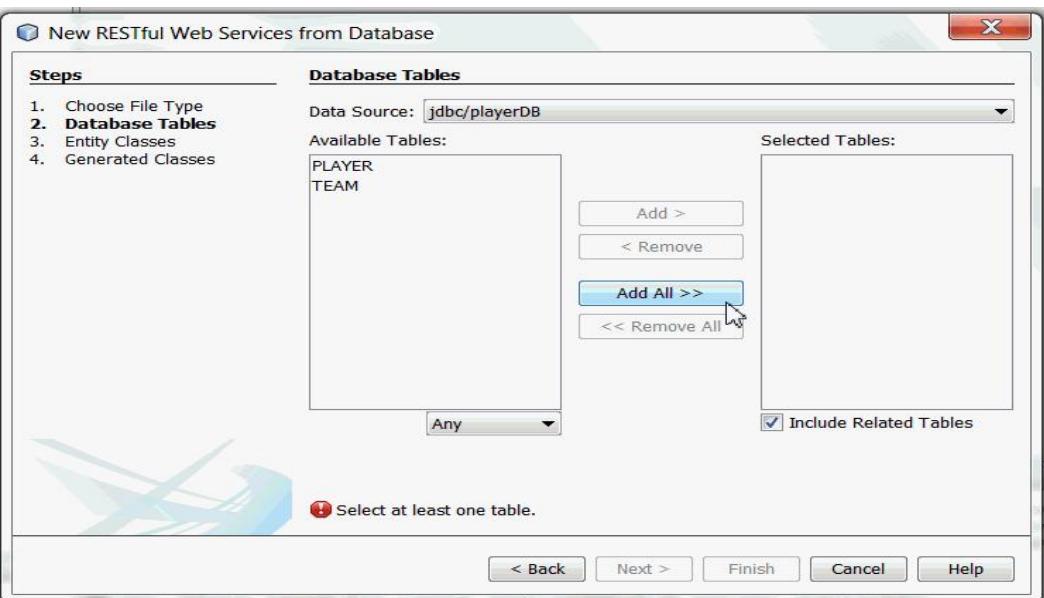
- o **JNDI name:** jdbc/playerDB
- o **Database connection :** select jdbc:derby://localhost:1527/playerDB[john on JOHN]

b. Click OK.



The PLAYER and TEAM tables are displayed under the Available Tables column.

c. Click **Add All**. The PLAYER and TEAM tables are displayed under the Selected Tables column.



d. Click Next.

4. In the **Entity Classes** window, complete the following steps:

- Enter the package name as com.playerentity.
- Click Next.

5. In the **Generated Classes** Window, click Finish with default values.

Test RESTful web Services

- To Generate Web Services Test client, complete the following steps.
 - Select **PlayerServer** project.
 - Right-click and select **Test RESTful Web Services**.
- Select "Web Test Client in project" and click Browse.
- a. In the Select Project dialog box, select **PlayerServer** and click OK.
b. **Configure Rest Test Client window** is displayed, click OK.

WADL: <http://localhost:8080/playerserver/resources/application.wadl>

Test RESTful Web Services

playerserver > pkgserver.player > {id}

Resource: pkgserver.player/{id} (pkgserver.player/{id})

Choose method to test: GET MIME: application/xml Add Parameter Test

id:

Status: 200 (OK)

Response:

Tabular View Raw View Sub-Resource Headers Http Monitor

```
<?xml version="1.0" encoding="UTF-8"?> version="1.0" encoding="UTF-8" standalone="yes"
<players>
  <player>
    <firstname>Jordan</firstname>
    <id>1</id>
    <jerseynumber>30</jerseynumber>
    <lastname>Michael</lastname>
    <lastspokenwords> will be back</lastspokenwords>
  </player>
  <player>
    <firstname>Becks</firstname>
    <id>2</id>
    <jerseynumber>20</jerseynumber>
    <lastname>David</lastname>
    <lastspokenwords> will make it to the team</lastspokenwords>
  </player>
  <player>
    <firstname>Harry</firstname>
    <id>30</id>
    <jerseynumber>60</jerseynumber>
    <lastname>Potter</lastname>
  </player>
</players>
```

Contains commands for working with the selected items.

Player Client

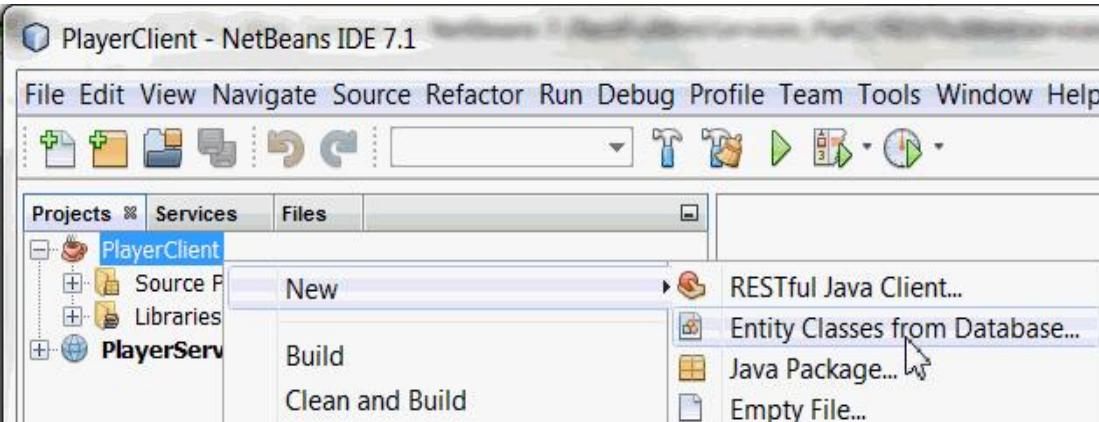
To create new Java Project, select File > New Project > Playerclient

Generating Entity Classes from Database

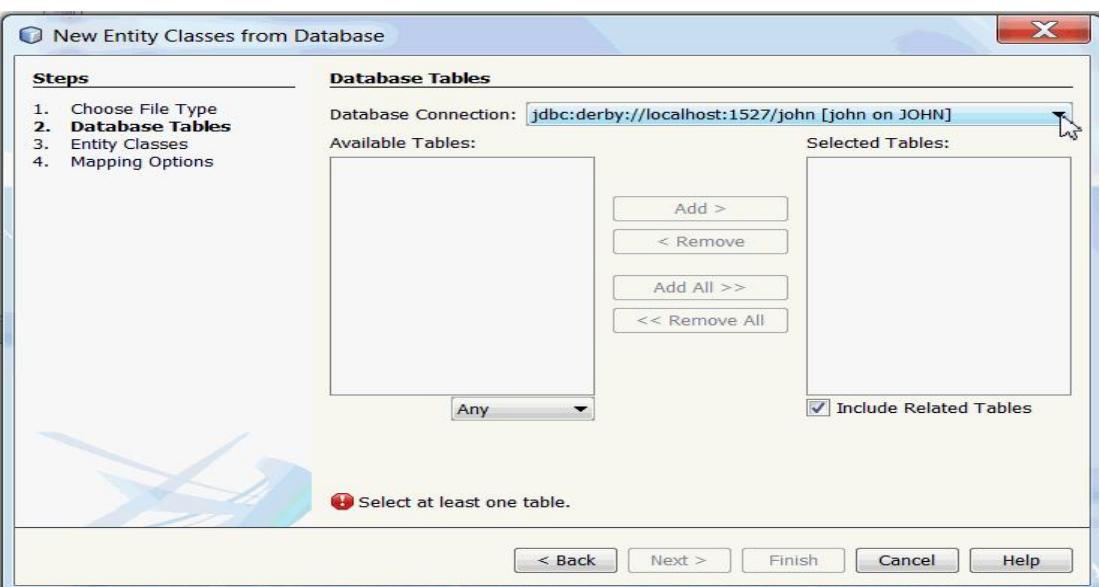
Implementing CRUD operations on the database, requires creation of Entity Classes to communicate with the database. The following section demonstrates how to create Entity

Classes from database.

1. Right-click PlayerClient Project and select **New > Entity Classes From Database**.

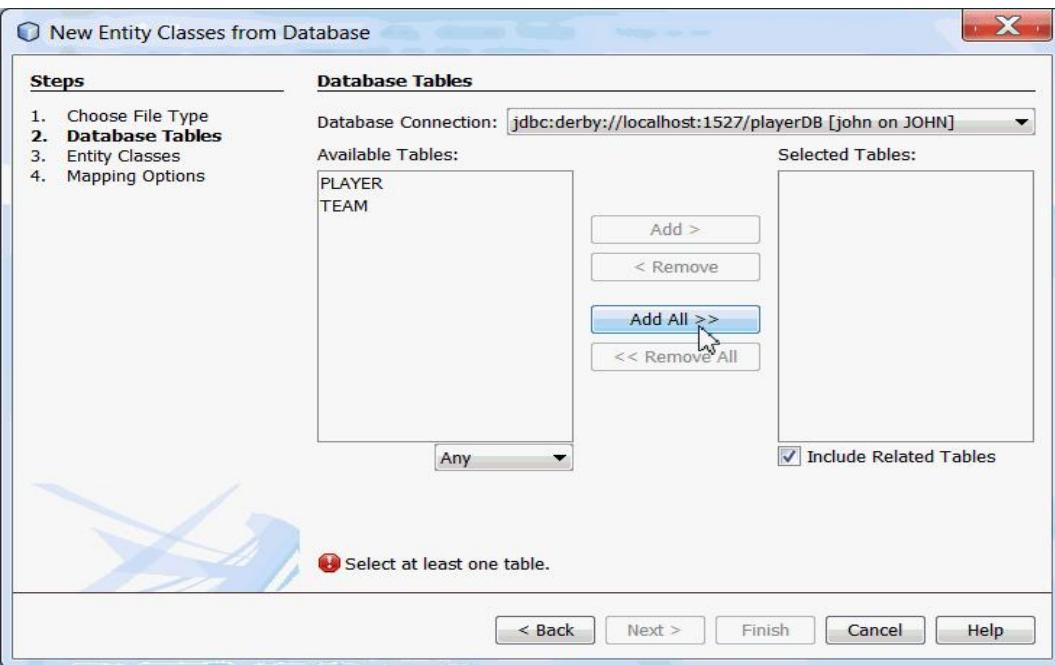


2. a. In the Database Tables window, **Database Connection** field
select jdbc:derby://localhost:1527/playerDB[john on JOHN] from the drop-down list.



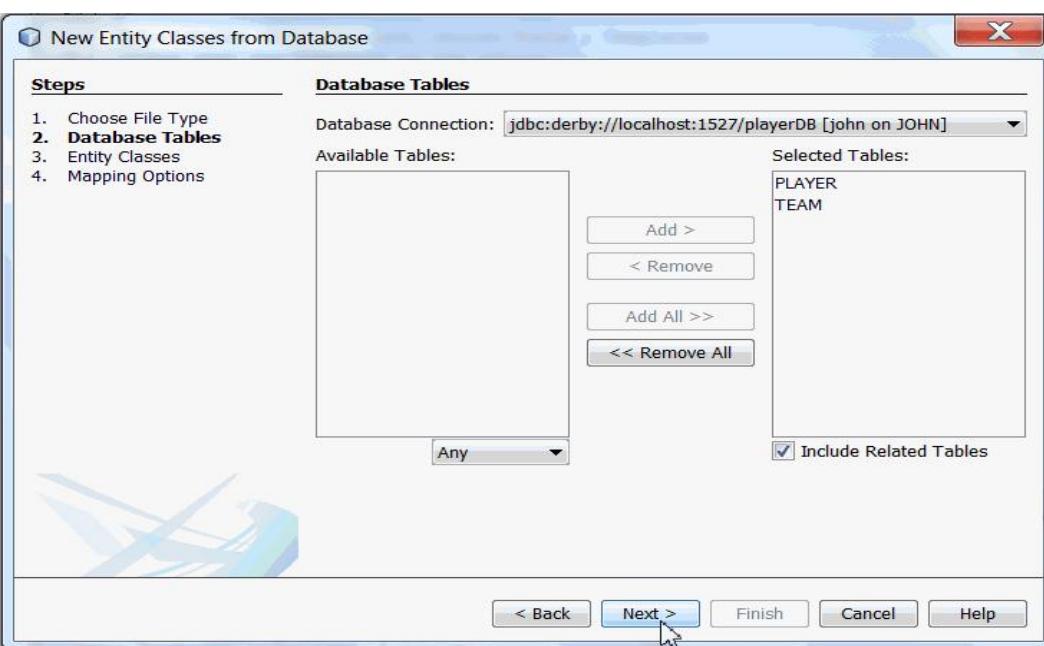
You see PLAYER and TEAM tables in Available Tables category

b. Click Add All

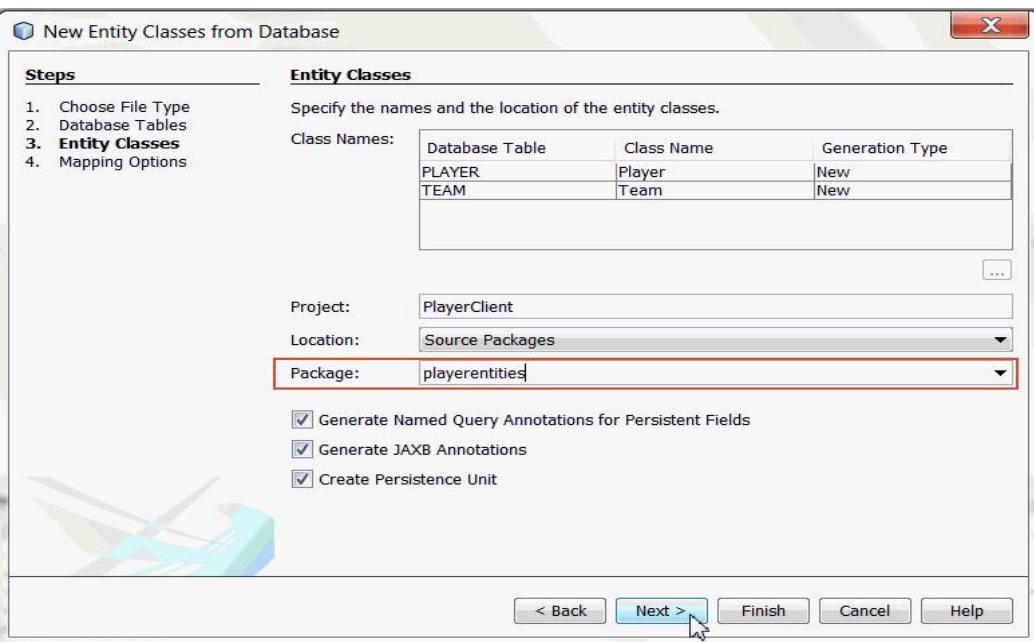


You see both the tables PLAYER and TEAM in Selected Tables Category.

c. Click Next



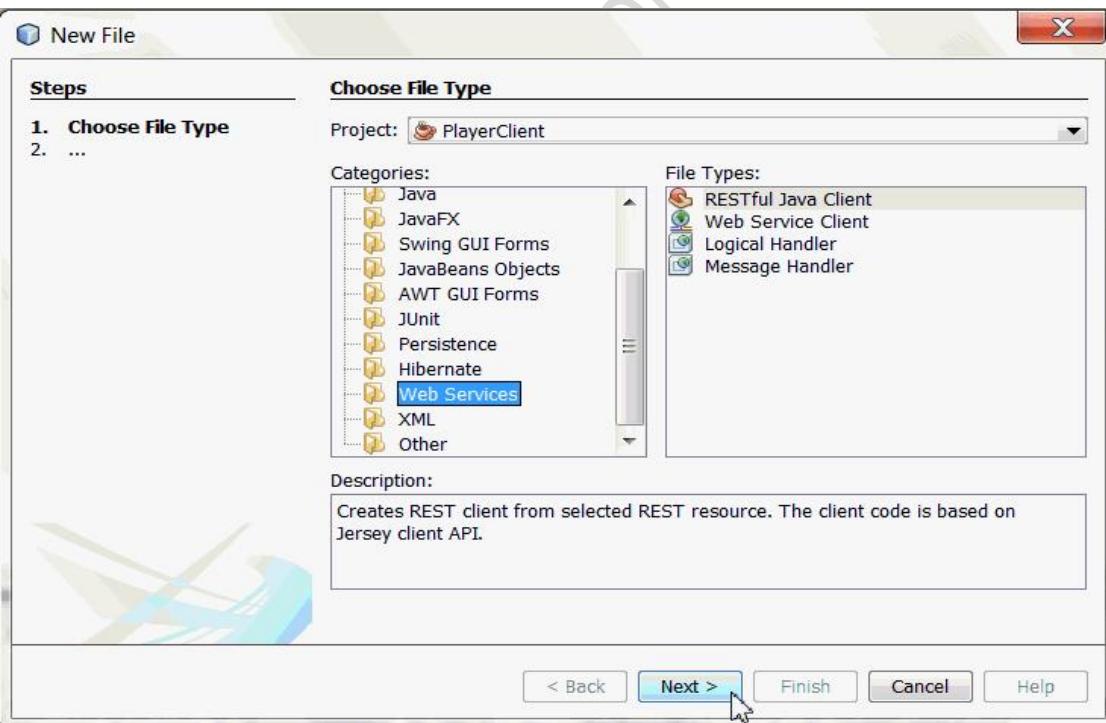
3. In the Entity classes Window, enter the Package Name as **playerentities** and click Next.



Create Client

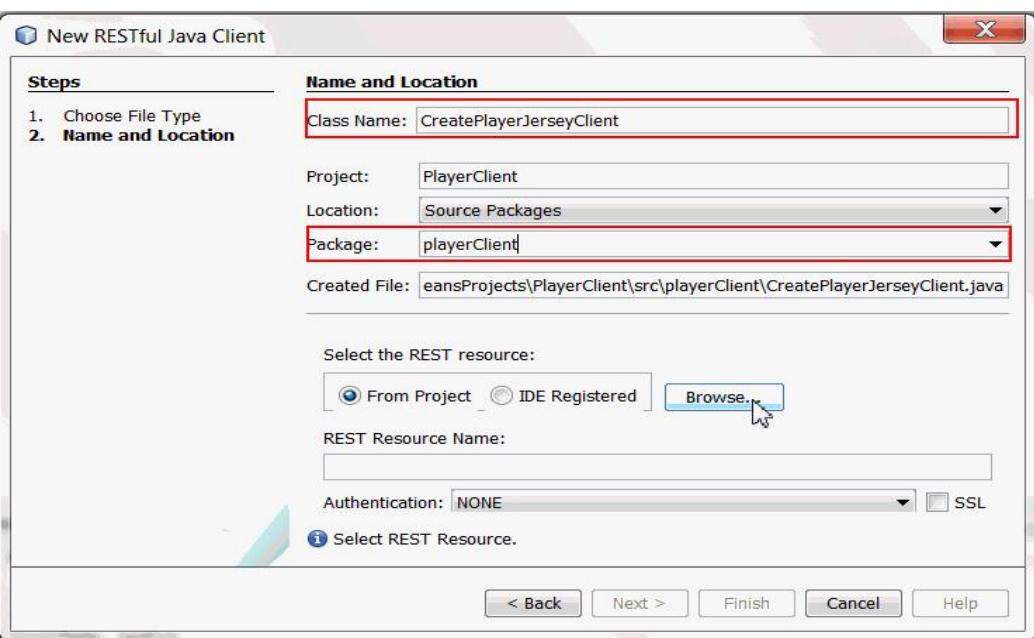
Complete the following steps to create RESTful client in PlayerClient project.

- Right-click the PlayerClient and choose New > Other > Web Services > RESTful Java Client.
- Click Next.

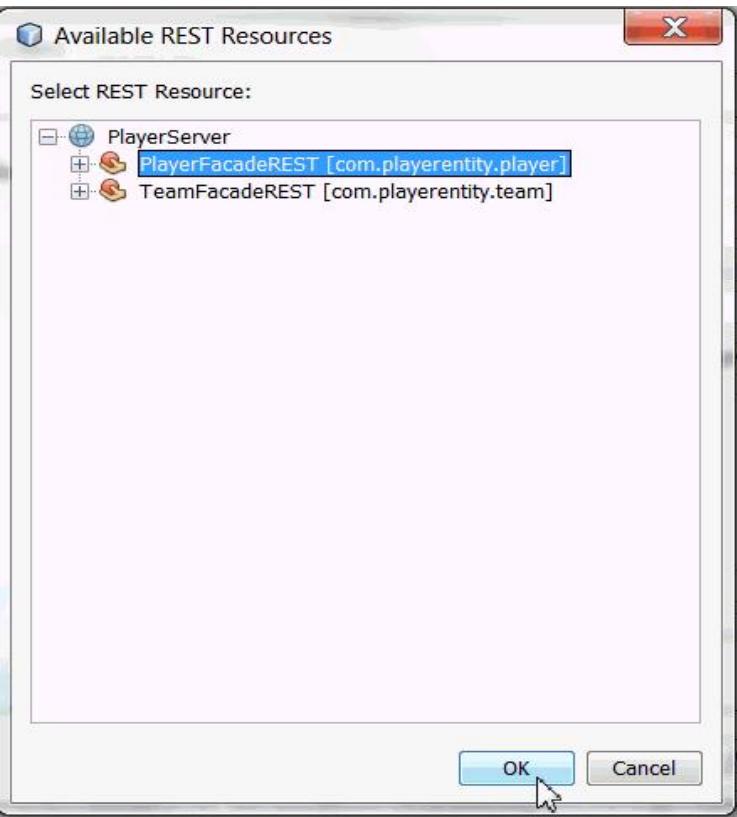


- a. Enter the following details for the fields in the New Restful Java Client window :

- Class Name: CreatePlayerJerseyClient
- Package: playerClient
- Select the REST resource: From Project and Click Browse.



- b. Select PlayerServer.
- c. Expand the PlayerServer application.
- d. Select PlayerFacadeREST[com.playerentity.player].



2.Add the main method.

```
public static void main(String[]args){  
    CreateClient client1 = new CreateClient();  
    ClientResponse response = client1.findAll_XML(ClientResponse.class);  
  
    GenericType<List<Player>> genericType = new GenericType<List<Player>>(){};  
    List<Player> data = new ArrayList<Player>();  
    data = (response.getEntity(genericType));  
  
    Player p = new Player();  
    p.setFirstname("Harry");  
    p.setId(30);  
    p.setJerseynumber(60);  
    p.setLastname("Potter");  
    p.setLastspokenwords("Thanks to my fans");
```

```
client1.create_XML(p);
```

```
}
```

#	ID	LASTNAME	FIRSTNAME	JERSEYNUMBER	LASTSPOKENWORDS
1		1 Michael	Jordan	30	I will be back
2		2 David	Becks	100	I will be retiring soon
3		30 Potter	Harry	60	Thanks to my fans

Read Client

```
public static void main(String[]args){  
    ReadClient client1 = new ReadClient();  
    ClientResponse response = client1.findAll_XML(ClientResponse.class);  
    GenericType<List<Player>> genericType = new GenericType<List<Player>>(){};  
    List<Player> data = new ArrayList<Player>();  
    data = (response.getEntity(genericType));  
    System.out.println("Retreiving and Displaying Player Details");  
    for(Player players:data){  
        System.out.println("FirstName:"+players.getFirstname());  
        System.out.println("PlayerID:"+players.getId());  
        System.out.println("JerseyNo:"+players.getJerseynumber());  
    }  
    System.out.println("Data Retreived!!");  
}
```

Output

Java DB Database Process x GlassFish Server 3.1.1 x SQL Command 1 execution x playerclient (run) x

```
run:
Retreiving and Displaying Player Details
FirstName:Jordan
PlayerID:1
JerseyNo:30
FirstName:Becks
PlayerID:2
JerseyNo:100
FirstName:Harry
PlayerID:30
JerseyNo:60
Data Retrieved!!
BUILD SUCCESSFUL (total time: 1 second)
```

Delete Client

```
public static void main(String[]args){
    DeleteClient client1 = new DeleteClient();
    client1.remove("1");
}
```

select * from MSC17.PLAYE... x

The screenshot shows a database query results window. At the top, there is a search bar with the text "select * from MSC17.PLAYE... x". Below the search bar, there are navigation icons for a table (grid), a refresh symbol, and arrows for sorting and filtering. To the right of these icons, it says "Page Size: 20", "Total Rows: 2", "Page: 1 of 1", and "Matching Rows: 2". The main area displays a table with the following data:

#	ID	LASTNAME	FIRSTNAME	JERSEYNUMBER	LASTSPOKENWORDS
1	2	David	Becks	100	I will be retiring soon
2	30	Potter	Harry	60	Thanks to my fans

Update Client

```
public static void main(String[]args){
    UpdateClient client1 = new UpdateClient();
```

```
ClientResponse response = client1.find_XML(ClientResponse.class, "2");
GenericType<Player>genericType = new GenericType<Player>() {};
Player player = response.getEntity(genericType);
System.out.println("First Name: "+player.getFirstname());
System.out.println("PlayerId: "+player.getId());
System.out.println("JerseyNo: "+player.getJerseynumber());
System.out.println("Last Name: "+player.getLastname());
System.out.println("Last Spoken Words: "+player.getLastspokenwords());
player.setJerseynumber(100);
player.setLastspokenwords("I will be retiring soon");
client1.edit_XML(player);
```

}

#	ID	LASTNAME	FIRSTNAME	JERSEYNUMBER	LASTSPOKENWORDS
1	1	Michael	Jordan	30	I will be back
2	2	David	Becks	100	I will be retiring soon
3	30	Potter	Harry	60	Thanks to my fans

Practical No. 3B

- Develop Micro-blogger application (like Twitter) using RESTful Web services.

Blogger Server

The screenshot shows the 'Test RESTful Web Services' interface. On the left, a tree view shows a project named 'BloggerServer' containing a package 'pkgserver.bloggerdb' with resources '{id}', '{from}/{to}', and 'count'. The main panel displays a REST endpoint: 'BloggerServer > pkgserver.bloggerdb > {id}'. A 'Resource' section shows the URL 'pkgserver.bloggerdb/{id}' with a note '(pkgserver.bloggerdb/{id})'. Below it, 'Choose method to test:' dropdown is set to 'GET', 'MIME:' is 'application/xml', and there are 'Add Parameter' and 'Test' buttons. An input field 'id:' contains the value '1'. At the bottom, the status is 'Status: 200 (OK)' and the response content is displayed in a tabular view, showing the XML structure: <?xml version='1.0' encoding='UTF-8'?> version='1.0' encoding='UTF-8' standalone='yes' <bloggerdb><content>Ron the chess Champion</content>

The screenshot shows the 'Test RESTful Web Services' interface again. The left pane is empty. The main panel displays a REST endpoint: 'BloggerServer > pkgserver.bloggerdb'. A 'Resource' section shows the URL 'pkgserver.bloggerdb'. Below it, 'Choose method to test:' dropdown is set to 'GET', 'MIME:' is 'application/xml', and there are 'Add Parameter' and 'Test' buttons. At the bottom, the status is 'Status: 200 (OK)' and the response content is displayed in a tabular view, showing the XML structure: <?xml version='1.0' encoding='UTF-8'?> version='1.0' encoding='UTF-8' standalone='yes' <bloggerdb><content>Ron the chess Champion</content>...</bloggerdb>

Blogger Client Create Client

```
public static void main(String[]args){  
    CreateClient client1 = new CreateClient();  
    ClientResponse response = client1.findAll_XML(ClientResponse.class);  
    GenericType<List<Bloggerdb>> genericType = new GenericType<List<Bloggerdb>>(){};  
    List<Bloggerdb> data = new ArrayList<Bloggerdb>();  
    data = (response.getEntity(genericType));  
    Bloggerdb b = new Bloggerdb();  
    b.setMessagelId(4);  
    b.setContent("Draco the Slytherine Champion");  
    b.setUsername("Draco");  
    b.setPassword("Malfoy");  
    client1.create_XML(b);  
}
```

#	MESSAGE_ID	USERNAME	CONTENT	PASSWORD
1	1	Ronald	Ron the chess Champion	fleur
2	2	Harry	Harry the Quidditch Champion	Ginny
3	3	Hermione	Hermione the scholar	Viktor
4	4	Draco	Draco the Slytherine Champion	Malfoy

Update Client

```
public static void main(String[]args){  
    UpdateClient client1 = new UpdateClient();  
    ClientResponse response = client1.find_XML(ClientResponse.class, "1");  
    GenericType<Bloggerdb>genericType = new GenericType<Bloggerdb>(){};  
    Bloggerdb b = response.getEntity(genericType);  
    System.out.println("Message id:"+b.getMessagelId());  
    System.out.println("Content:"+b.getContent());  
    System.out.println("Username:"+b.getUsername());  
    System.out.println("Password:"+b.getPassword());  
    b.setContent("I love food");  
    client1.edit_XML(b);  
}
```

#	MESSAGE_ID	USERNAME	CONTENT	PASSWORD
1		1Ronald	I love food	fleur
2		2Harry	Harry the Quidditch Champion	Ginny
3		3Hermione	Hermione the scholar	Viktor
4		4Draco	Draco the Slytherine Champion	Malfoy

Delete Client

```
public static void main(String[]args){  
    DeleteClient client1 = new DeleteClient();  
    client1.remove("4");  
    System.out.println("Successfully Deleted!!");  
}
```

#	MESSAGE_ID	USERNAME	CONTENT	PASSWORD
1		1Ronald	I love food	fleur
2		2Harry	Harry the Quidditch Champion	Ginny
3		3Hermione	Hermione the scholar	Viktor

Read Client

```
public static void main(String[]args){  
    ReadClient client1 = new ReadClient();  
    ClientResponse response = client1.findAll_XML(ClientResponse.class);  
    GenericType<List<Bloggerdb>> genericType = new GenericType<List<Bloggerdb>>(){};  
    List<Bloggerdb> data = new ArrayList<Bloggerdb>();  
    data = (response.getEntity(genericType));  
    System.out.println("Retreiving and Displaying Player Details");  
    for(Bloggerdb b:data){  
        System.out.println("Content: "+b.getContent());  
        System.out.println("Username: "+b.getUsername());  
        System.out.println("Password:"+b.getPassword());  
    }  
    System.out.println("Data Retreived!!");
```

The screenshot shows the NetBeans IDE interface with the Output window selected. The window displays the following text:

```
run:
Retreiving and Displaying Player Details
Content: I love food
Username: Ronald
Password:fleaur
Content: Harry the Quidditch Champion
Username: Harry
Password:Ginny
Content: Hermione the scholar
Username: Hermione
Password:Viktor
Data Retreived!!
BUILD SUCCESSFUL (total time: 1 second)
```

Practical-4

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

https://study.cscornersunitarai.com/

- Develop application to consume Google's search / Google's Map RESTful Web service.

1. First of all we need to create an Java Web Application with any name, let it be GoogleMap here using Netbeans IDE.

2. The code inside the input.jsp will be similar to this input.jsp 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>
```

3. 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).

The screenshot shows the Google APIs Dashboard. At the top, there are several tabs: 'https://mahad...', 'New announce...', 'M. Sc CS 2018...', 'syllabus_msc...', and 'Java Web S...'. Below the tabs, the URL is https://console.developers.google.com/apis/dashboard?project=eminent-kit-237101. The main header says 'Google APIs' and 'GoogleMap'. A search bar is on the right. The left sidebar has sections for 'APIs & Services' and 'ENABLE APIS AND SERVICES'. Below this are icons for 'Cloud Functions', 'Cloud Pub/Sub', and 'Cloud Storage'. The main content area has two graphs: 'Traffic' and 'Errors'. The 'Traffic' graph shows 0.001/s. The 'Errors' graph shows 0 errors per second.

- Create a new API Project.

The screenshot shows a 'Select a project' dialog box. At the top, it says 'Select a project' and 'Search projects and folders'. There is a 'NEW PROJECT' button with a plus sign icon. Below is a search bar with a magnifying glass icon. The 'RECENT' tab is selected, showing three projects: 'GoogleMap' (selected), 'Practical6', and 'My Project'. The 'ALL' tab is also present. At the bottom are 'CANCEL' and 'OPEN' buttons.

Name	ID
GoogleMap	eminent-kit-237101
Practical6	practical6-235906
My Project	lateral-shore-180303

- Enter the name to your project.

The screenshot shows the 'New Project' creation interface. At the top, there's a warning message: 'You have 7 projects remaining in your quota. Request an increase or delete projects.' with a 'Learn more' link and a 'MANAGE QUOTAS' button. Below this, the 'Project name *' field is filled with 'My Project 81907'. A note below says 'Project ID: zeta-environs-237517. It cannot be changed later.' with an 'EDIT' link. Under 'Location *', it shows 'No organization' with a 'BROWSE' button. There's also a 'Parent organization or folder' input field. At the bottom are 'CREATE' and 'CANCEL' buttons.

- Enable the Google Maps API V3.

The screenshot shows the 'API Library' section. It lists the 'Maps JavaScript API' by Google. The entry includes a circular icon with 'JS' and 'Maps' text, the API name, developer information ('Google'), and status ('API enabled'). Below this, there's a detailed card with sections for 'Type' (APIs & services), 'Last updated' (1/10/19, 2:02 AM), 'Category' (Maps), 'Service name' (maps-backend.googleapis.com), and 'Overview'. The 'Overview' section describes the API's purpose of adding maps to websites. Other links include 'Tutorials and documentation'.

4. Create another file index.jsp

Index.jsp

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

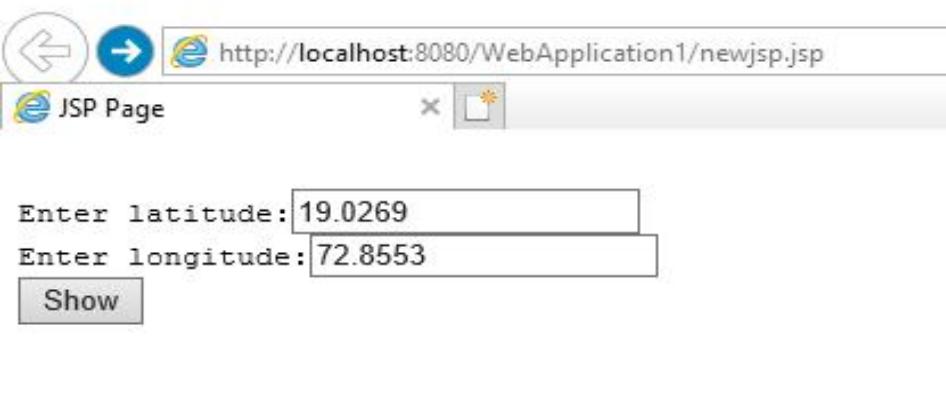
```
#map {  
    height: 400px;  
    width: 100%;  
}  
</style>  
</head>  
<body>  
<%  
    double lati=Double.parseDouble(request.getParameter("t1"));  
    double longi=Double.parseDouble(request.getParameter("t2"));  
%>  
<h3> Google Maps </h3>  
<div id="map"></div>  
<script lang="javascript">  
    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="put your key here">
```

<https://study.cscornersunitarai.com/>

<https://www.youtube.com/@cscornersunitarai>

```
</script>
</body>
</html>
```

Output :-



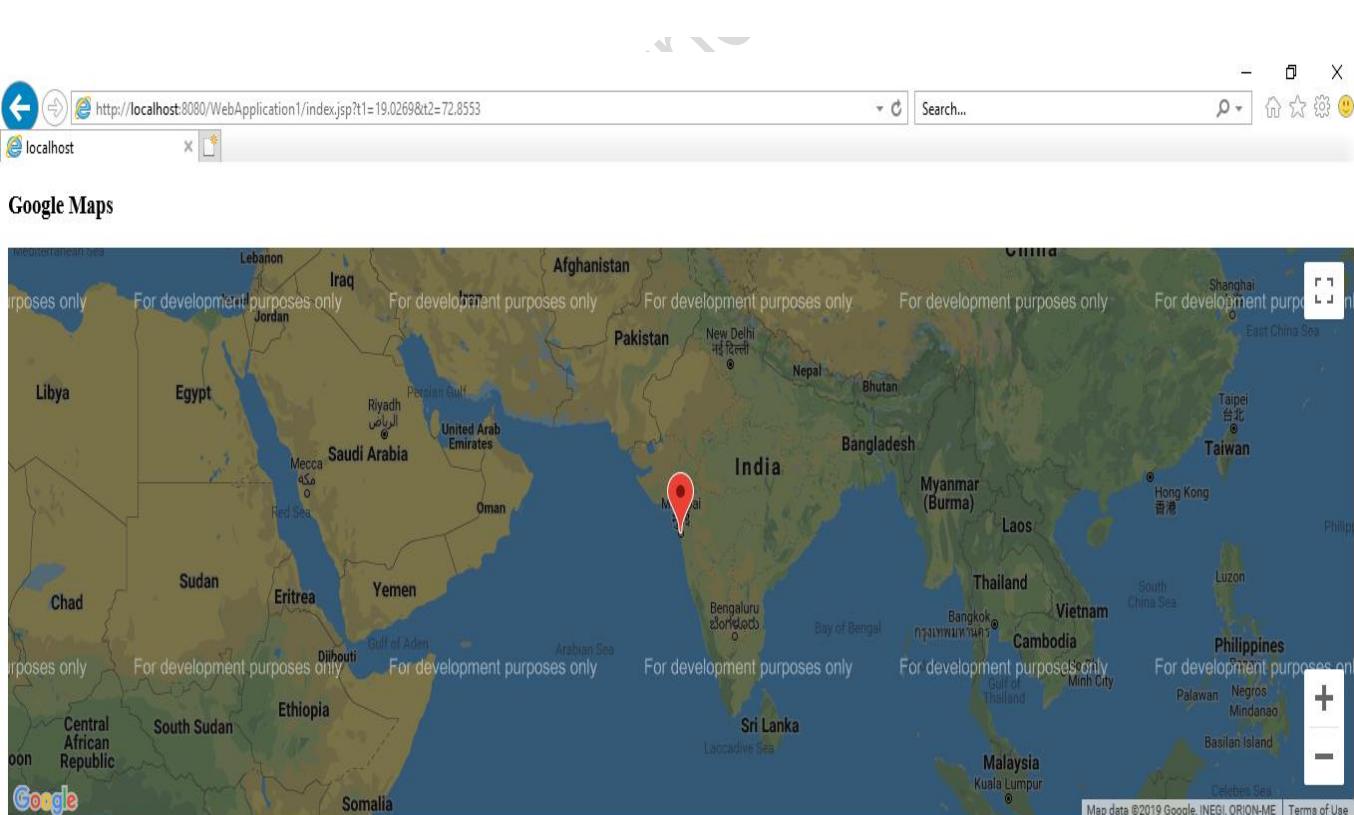
http://localhost:8080/WebApplication1/newjsp.jsp

JSP Page

Enter latitude:

Enter longitude:

Show



Google Maps

Map data ©2019 Google, INEGI, ORION-ME Terms of Use

Practical-5

Installation and Configuration of virtualization using KVM

Practical-6

Develop application to download image/video from server or upload image/video to server using MTOM techniques

MTOMClient

Index.jsp

```
<%@page import="java.io.BufferedOutputStream">
<%@page import="java.io.FileOutputStream">
<%@page import="java.io.FileInputStream">
<%@page import="java.io.BufferedReaderInputStream">
<%@page import="javax.imageio.stream.FileImageInputStream">
<%@page import="java.io.File">
<%@page import="javax.xml.ws.soap.MTOMFeature">
<%@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>
```

```
<%-- start web service invocation --><hr/>
<%
try {
pkg.ImageWS_Service service = new pkg.ImageWS_Service();
pkg.ImageWS port = service.getImageWSPort(new MTOMFeature(60000));
// TODO initialize WS operation arguments here
String filePath="c:/msc/ABC.jpg";
File file=new File(filePath);
FileInputStream fis=new FileInputStream(file);
BufferedInputStream bis=new BufferedInputStream(fis);
java.lang.String filename = file.getName();
byte[]imageBytes=new byte[(int)file.length()];
bis.read(imageBytes);
port.upload(filename, imageBytes);
bis.close();
out.println("File uploaded :" + filePath);
} catch (Exception ex) {
// TODO handle custom exceptions here
}
>
<%-- end web service invocation --><hr/>

<%-- start web service invocation --><hr/>
<%
try {

pkg.ImageWS_Service service = new pkg.ImageWS_Service();
pkg.ImageWS port = service.getImageWSPort();
// TODO initialize WS operation arguments here
java.lang.String filename = "ABC.jpg";
String filePath="c:/msc/download/" + filename;
// TODO process result here
byte[] fileBytes = port.download(filename);
FileOutputStream fos=new FileOutputStream(filePath);
BufferedOutputStream bos=new BufferedOutputStream(fos);
bos.write(fileBytes);
bos.close();
out.println("File downloaded" + filePath);
} catch (Exception ex) {
// TODO handle custom exceptions here
}
>
<%-- end web service invocation --><hr/>
</body>
</html>
MTOMServer
ImageWS.java
package mypkg;
import java.io.*;
import javax.jws.Oneway;
```

```
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.xml.ws.soap.MTOM;
@MTOM(enabled=true,threshold=60000)
@WebService(serviceName = "ImageWS")
public class ImageWS {
    @WebMethod(operationName = "upload")
    @Oneway
    public void upload(@WebParam(name = "Filename") String Filename, @WebParam(name =
    "ImageBytes") byte[] ImageBytes){
        String filePath="C:/MSC/upload/"+Filename;
        try
        {
            FileOutputStream fos=new FileOutputStream(filePath);
            BufferedOutputStream bos=new BufferedOutputStream(fos);
            bos.write(ImageBytes);
            bos.close();
            System.out.println("Recieved file :"+filePath);
        }
        catch(Exception ex)

        {
            System.out.println(ex);
        }
    }
    @WebMethod(operationName = "download")
    public byte[] download(@WebParam(name = "Filename") String Filename) {
        String filePath="C:/MSC/upload/"+Filename;
        System.out.println("Sending file :"+filePath);
        try
        {
            File file=new File(filePath);
            FileInputStream fis=new FileInputStream(file);
            BufferedInputStream bis=new BufferedInputStream(fis);
            byte[] fileBytes=new byte[(int)file.length()];
            bis.read(fileBytes);
            bis.close();
            return fileBytes;
        }
        catch(Exception ex)
        {
            System.out.println(ex);
        }
        return null;
    }
}
Output:
```

<https://study.cscornersunitarai.com/>

<https://www.youtube.com/@cscornersunitarai>

Practical-07

<https://www.youtube.com/@codewithsunita>

Implement FOSS-Cloud Functionality VSI (Virtual Server Infrastructure)
Infrastructure as a Service (IaaS), Creating Virtual Machine or Storage

FOSS Cloud Installation

Steps:

1. Choose your keymap.
2. Confirm that you want to start: yes
3. Choose Demo-System: 1
4. Choose a Block-Device: sda
5. Confirm that you want to continue: yes
6. Confirm that you want to continue: yes
7. Choose the network interface: eth0
8. Choose if you want to use automatic network configuration: Yes
9. Reboot your system: yes
10. Login as root and run "fc-node-configuration -n demo-system --password admin"

Note :- Above mentioned steps aren't mandatory you can start with the installation process of virt-viewer as shown below.

1. Install virt-viewer.
2. Install spice-client.0.6.3.
3. Open Browser.
 - Once you open browser then on the search type, **192.168.21.36**

The screenshot shows the FOSS-Cloud homepage. On the left, there's a sidebar with navigation links: Virtual Machine, VM Pool, Storage Pool, Node, Network, User, Configuration, Diagnostics, and Assigned VMs. The main content area has a heading "Welcome to the FOSS-Cloud". It includes a brief description of what FOSS-Cloud is, a note about donations, and a "Donate" button. Below that, there's a "Thank you for using FOSS-Cloud" message and credit to "The FOSS-Cloud Team". On the right, there's a "Links" section with links to Documentation and Splice-Client (with protocol handler) download. At the bottom, there's a copyright notice: "Version 1.3.2 on server localhost. Copyright © 2024 by FOSS-Group. All Rights Reserved." A "Logout (admin)" link is at the top right.

•Uploading ISO files to Foss cloud

Go to virtual machine→Profile→Upload ISO file→file name

The screenshot shows the "Upload ISO File" page. The sidebar on the left has a "Virtual Machine" section with "Upload ISO File" highlighted. The main form has a title "Upload ISO File" and a note "Fields with * are required." It includes a link "Alternative upload method", a "Iso File" field with a "Choose file" button and a "No file chosen" message, a "File Name" input field, and an "Upload" button. The rest of the sidebar and header are similar to the previous screenshot.

Now choose the **ubuntu-16.04.5-desktop-i386 iso-file** from the FOSS cloud folder. Then, give your desired file name to it and click on the upload button.

The screenshot shows the FOSS Cloud web interface. In the top left, the logo 'FOSS Cloud' is visible. The top right features a language selection dropdown ('EN') and a 'Logout (admin)' link. The main navigation menu on the left includes 'Home', 'About', 'Contact', and a expanded 'Virtual Machine' section which contains 'Persistent VMs', 'Dynamic VMs', 'VM Templates', 'Create', 'Profiles', 'Create', and 'Upload ISO File'. Below this is a list of system components: 'VM Pool', 'Storage Pool', 'Node', 'Network', 'User', 'Configuration', 'Diagnostics', and 'Assigned VMs'. The central area is titled 'Upload ISO File' with a note that fields marked with an asterisk (*) are required. It includes an 'Alternative upload method' link and a 'File Name' input field containing 'cloucomputing'. A file selection button 'Choose file' has 'ubuntu-16.0...' selected. A progress bar indicates 'Upload finished (1.55 GB)'. A modal window titled 'Finished' displays the message 'Upload finished!' and notes 'took 15 seconds'. The background of the page has a watermark-like diagonal text: 'https://study.cscornersunitarai'.

Once done with the upload process go to the create section in profiles.

Creating a Profile

The profile creates the relationship between the ISO file and the FOSS-Cloud.

1. Open Virtual Machines
2. Choose VM-Profiles
3. Choose the right Base Profile
4. Choose the right architecture (Windows only x86_64)
5. Choose the language (it is a information - not keyboard relevant)
6. Choose the ISO file (which you gave in file name tab)
7. Fill out name and description
8. Choose the amount of memory and volume capacity
9. Choose amount of CPU
10. Choose clock offset (normally Windows is "localtime" and Linux is "utc")

Virtual Machine

Persistent VMs

Dynamic VMs

VM Templates

Create

Profiles

Create

Upload ISO File

VM Pool

Storage Pool

Node

Network

User

Configuration

Diagnostics

Assigned VMs

Create VM Profile

Fields with * are required.

Step I
Please select a profile first

Step II
Overwrite the default values if necessary!

BaseProfile

- linux
- windows
 - default
 - nasir34
 - vip
- TYCS2024
 - x86_64
 - en-US
 - foss_server
 - ubuntu_19
 - priyanka
 - kayapan
 - ky pn
 - Ubuntu VM
 - VM414
 - khushboo
 - ubuntusk
 - Roman
 - virtual
 - R_UBUNTU
 - VM441

Isofile

- ubuntu.iso
- Students.iso
- ubuntu-16.04.5-desktop-i386.iso
- CC.iso

Name *

TYCS2024

Description *

Welcome To The World Of Programming

Memory *

256 MB 128 GB 2 GB

Volume Capacity *

10 GB 2048 GB 10 GB

CPU *

1

Clock Offset *

localtime

Create

Activate \ Go to Settings

Now, we will create a VM template.

Creating Template

1. Choose the profile you have prepared before.
2. Add the VM-pool and one or more nodes, where you will run this VM (when the chosen VM-pool has only one node assigned, you don't have a choice)
3. You can change all the other information you have entered before

Click on "create" and the template is ready for installing the guest operating system.

The screenshot shows the 'VM Pool' configuration interface. On the left, a sidebar lists various management options like VM Pool, Storage Pool, Node, Network, User, Configuration, Diagnostics, and Assigned VMs. The main area displays a tree view of existing VMs under 'nasir34'. A new template is being created, with the 'en-US' language selected. The configuration form includes fields for Name (TYCS2024), Description (Welcome To The World Of Programming), Memory (4.13 GB), Volume Capacity (105 GB), CPU (1), Clock Offset (localtime), and Number of displays (1). A 'Create' button is at the bottom.

Once you create your virtual machine template you will be able to see your **VM template**.

The screenshot shows the 'Manage VM Templates' page in the Foss Cloud interface. The left sidebar has 'Virtual Machine' selected, with sub-options for Persistent VMs, Dynamic VMs, VM Templates (selected), Create, Profiles, Create, and Upload ISO File. The main content area shows a table titled 'Manage VM Templates' with a dropdown menu 'Vm Pool' set to 'vm-template-virtual-machine-pool-01'. The table lists nine entries, each with a status icon (red for stopped, green for running), a display name, status (e.g., stopped or running), run action icons, memory usage (e.g., 2 GB / 2 GB), node (foss-cloud-01.foss-cloud.org), and action icons. The last row, 'TYCS2024', has a red status icon and is listed as stopped.

No.	DisplayName	Status	Run Action	Memory	Node	Action
+	1 ubuntu_19	stopped	→ ↓ ✘ ↻	...	foss-cloud-01.foss-cloud.org	
+	2 priyanka	stopped	→ ↓ ✘ ↻	...	foss-cloud-01.foss-cloud.org	
+	3 413	stopped	→ ↓ ✘ ↻	...	foss-cloud-01.foss-cloud.org	
+	4 VM414	stopped	→ ↓ ✘ ↻	...	foss-cloud-01.foss-cloud.org	
+	5 Roman	stopped	→ ↓ ✘ ↻	...	foss-cloud-01.foss-cloud.org	
+	6 virtual	running	→ ↓ ✘ ↻	2 GB / 2 GB	foss-cloud-01.foss-cloud.org	
+	7 R_UBUNTU	stopped	→ ↓ ✘ ↻	...	foss-cloud-01.foss-cloud.org	
+	8 R_UBUNTU	running	→ ↓ ✘ ↻	2.63 GB / 2.63 GB	foss-cloud-01.foss-cloud.org	
+	9 TYCS2024	stopped	→ ↓ ✘ ↻	...	foss-cloud-01.foss-cloud.org	

As you can see the last row of Display column our template is created.

To update your template status, click on the green arrow.

FOG Cloud

Home About Contact Logout (admin)

Virtual Machine

Persistent VMs Dynamic VMs VM Templates Create Profiles Create Upload ISO File

VM Pool Storage Pool Node Network User Configuration Diagnostics Assigned VMs

Manage VM Templates

Vm Pool vm-template-virtual-machine-pool-01

No.	DisplayName	Status	Run Action	Memory	Node	Action			
1	ubuntu_19	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
2	privanka	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
3	413	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
4	VM414	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
5	Roman	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
6	virtual	running	➔ ⏪ ✘ 🚧	2 GB / 2 GB	foss-cloud-01.foss-cloud.org				
7	R_UNBUNTU	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
8	R_UNBUNTU	running	➔ ⏪ ✘ 🚧	2.63 GB / 2.63 GB	foss-cloud-01.foss-cloud.org				
9	TYCS2024	running	➔ ⏪ ✘ 🚧	4 GB / 4 GB	foss-cloud-01.foss-cloud.org				

Now click on Use Template in Action column of your corresponding template.

FOG Cloud

Home About Contact Logout (admin)

Virtual Machine

Persistent VMs Dynamic VMs VM Templates Create Profiles Create Upload ISO File

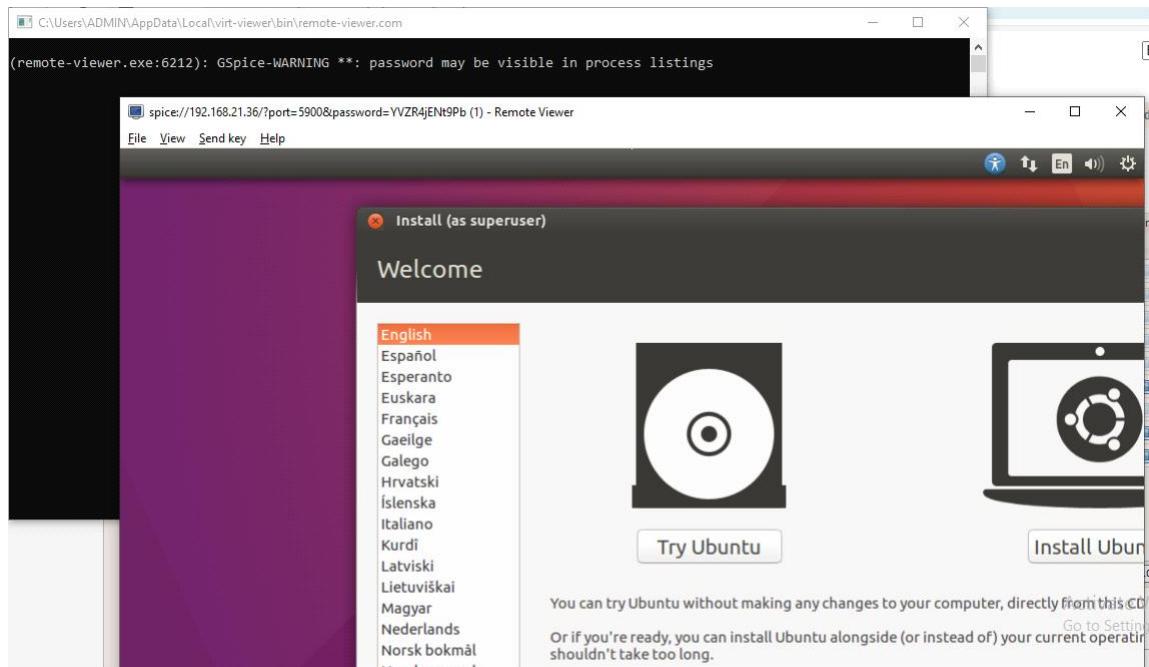
VM Pool Storage Pool Node Network User Configuration Diagnostics Assigned VMs

Manage VM Templates

Vm Pool vm-template-virtual-machine-pool-01

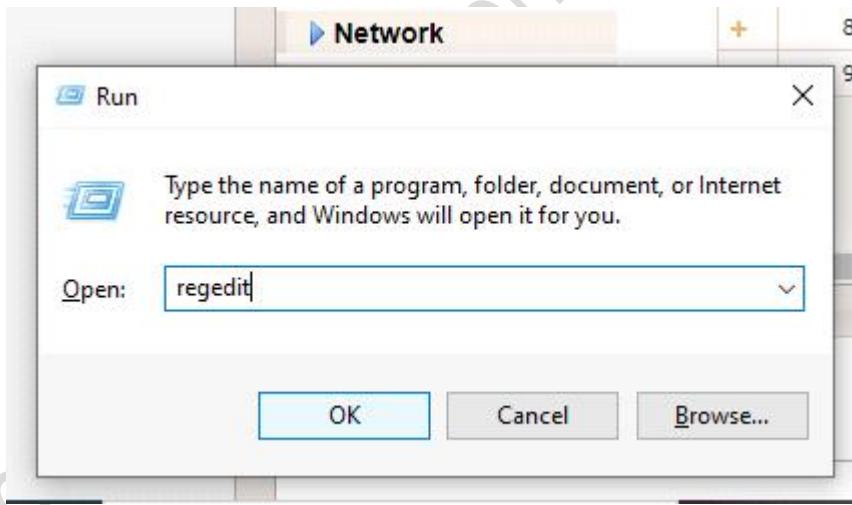
No.	DisplayName	Status	Run Action	Memory	Node	Action			
1	ubuntu_19	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
2	privanka	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
3	413	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
4	VM414	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
5	Roman	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
6	virtual	running	➔ ⏪ ✘ 🚧	2 GB / 2 GB	foss-cloud-01.foss-cloud.org				
7	R_UNBUNTU	stopped	➔ ⏪ ✘ 🚧	---	foss-cloud-01.foss-cloud.org				
8	R_UNBUNTU	running	➔ ⏪ ✘ 🚧	2.63 GB / 2.63 GB	foss-cloud-01.foss-cloud.org				
9	TYCS2024	running	➔ ⏪ ✘ 🚧	4 GB / 4 GB	foss-cloud-01.foss-cloud.org				

Just after clicking on it a pop up would appear to Open remote-viewer? click on Open-remote-viewer button, you will see the following screen.



Note :- If you don't see the above screen follow the below given steps.

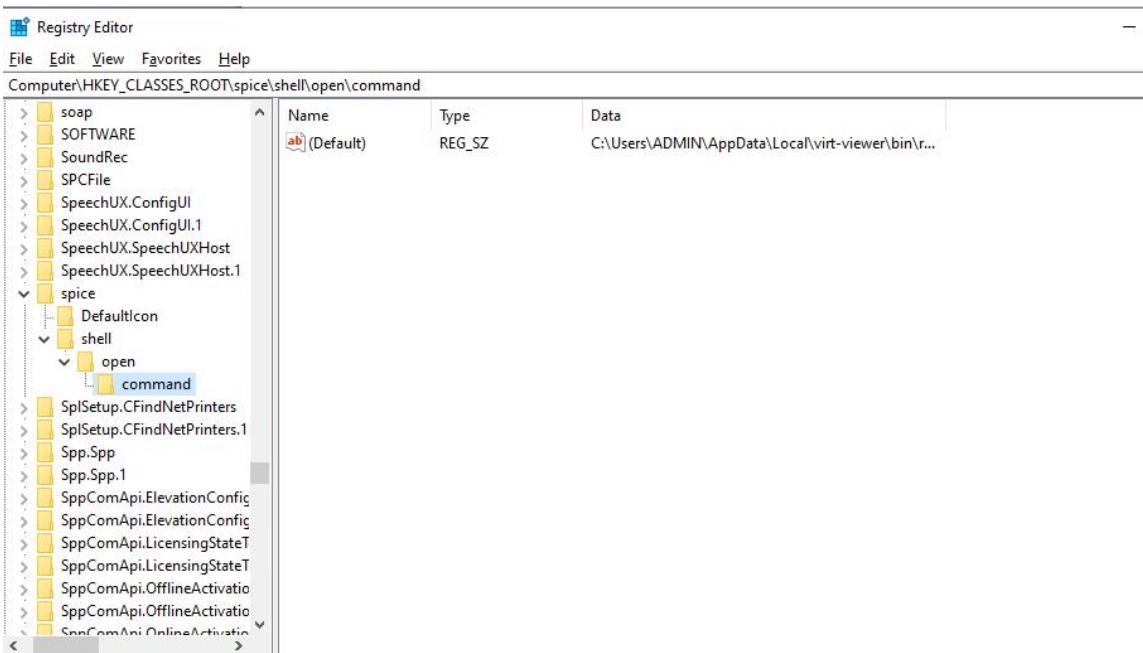
1. First press window + r, then type regedit and click ok.



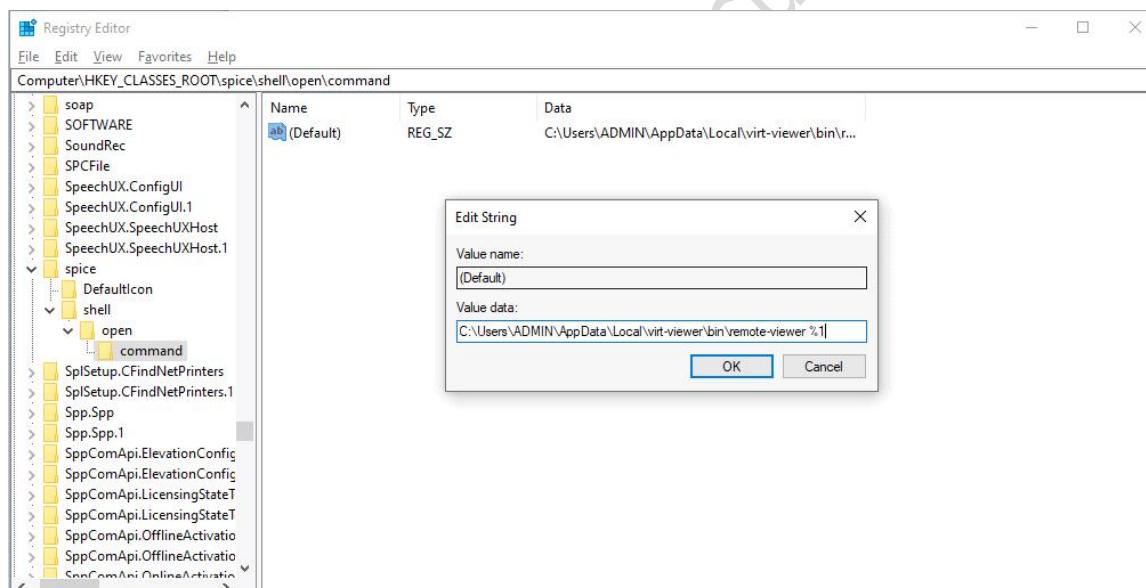
2. Further permissions tab will open then click yes, and a regedit editor will get open.

3. In the Registry Editor a list of directories would appear, in those lists of directories search for the spice directory.

4. Keep on expanding the spice directory until you reach the command section.

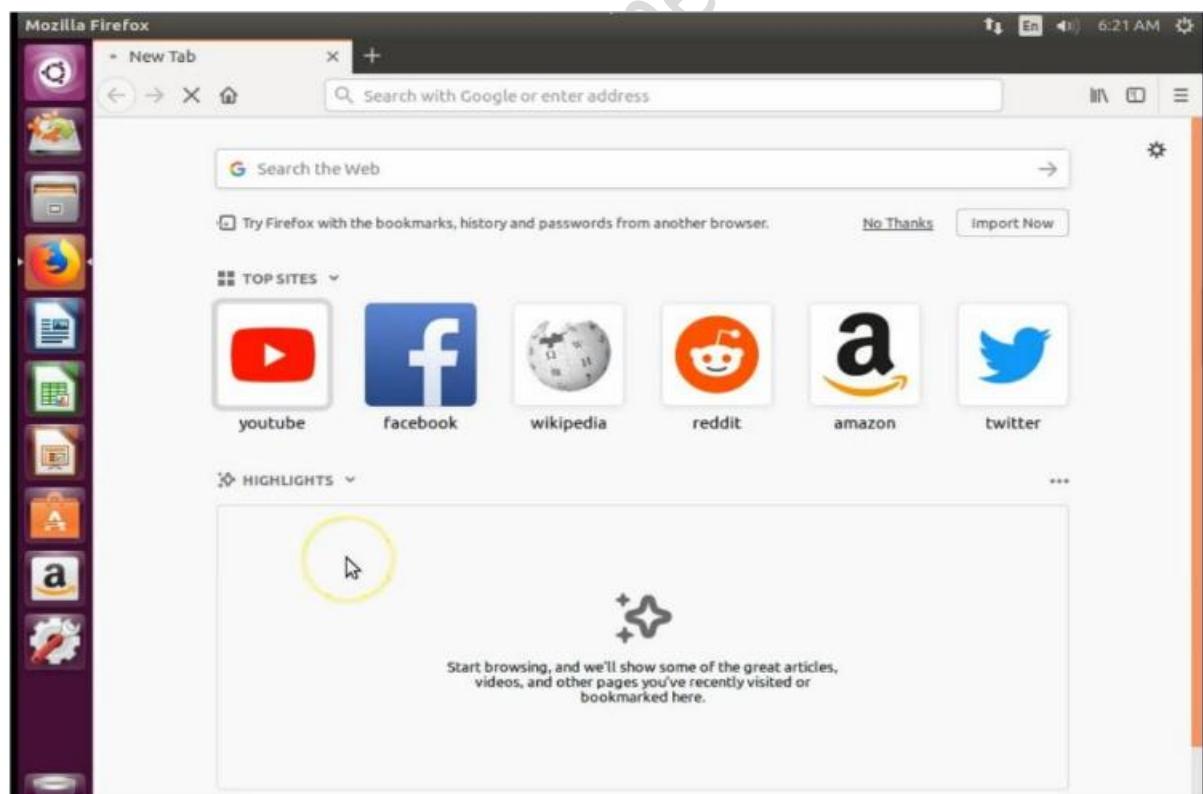
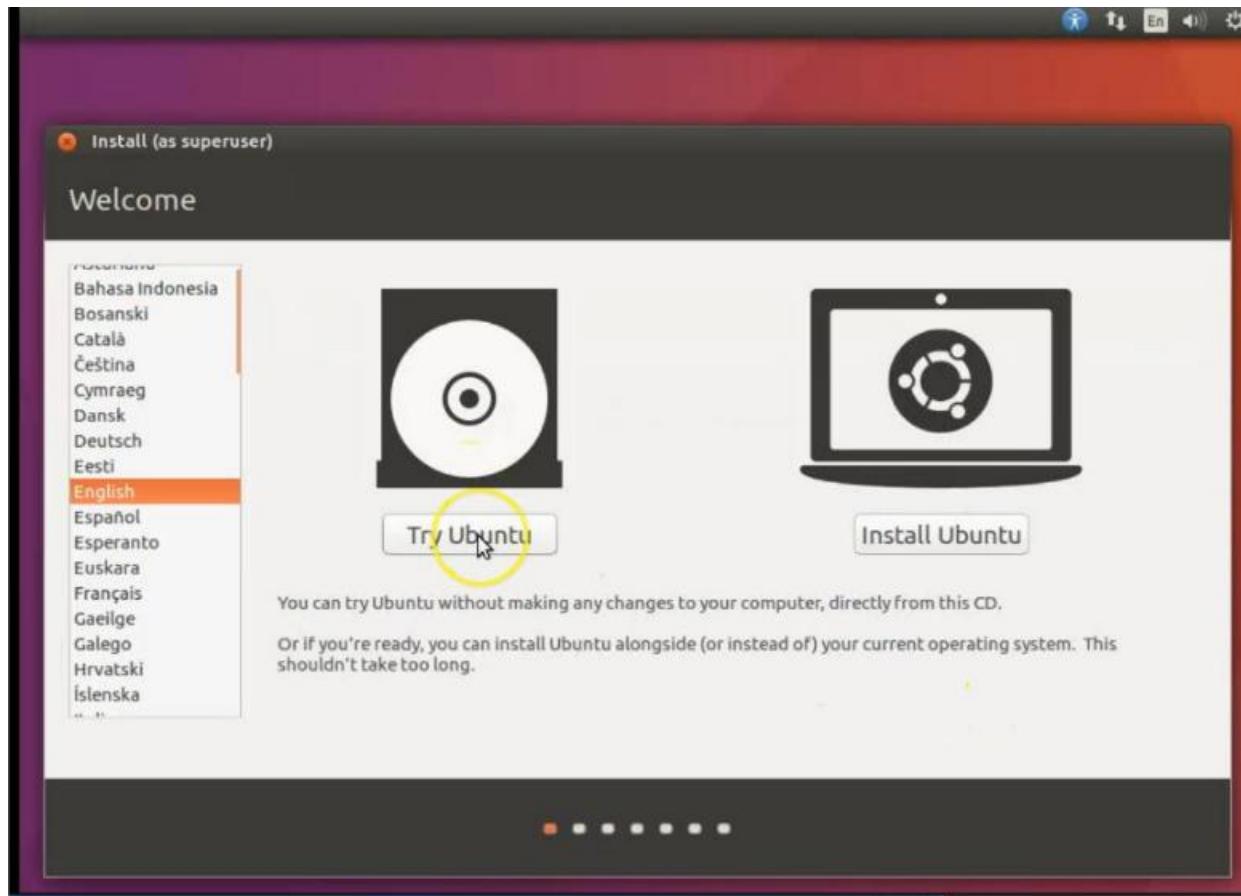


5. Then click on the default file and a pop up will appear, then you have to paste the path **C:\Users\ADMIN\AppData\Local\virt-viewer\bin** in the value data and **also type %1 in last**. Once done the above screen will appear.



Finally, after clicking on remote viewer button a command line come where a graphical control loading and getting connected will be displayed and then finally the ubuntu will get open.

On opening of ubuntu click on try ubuntu. Now, you can use ubuntu on your virtual machine.

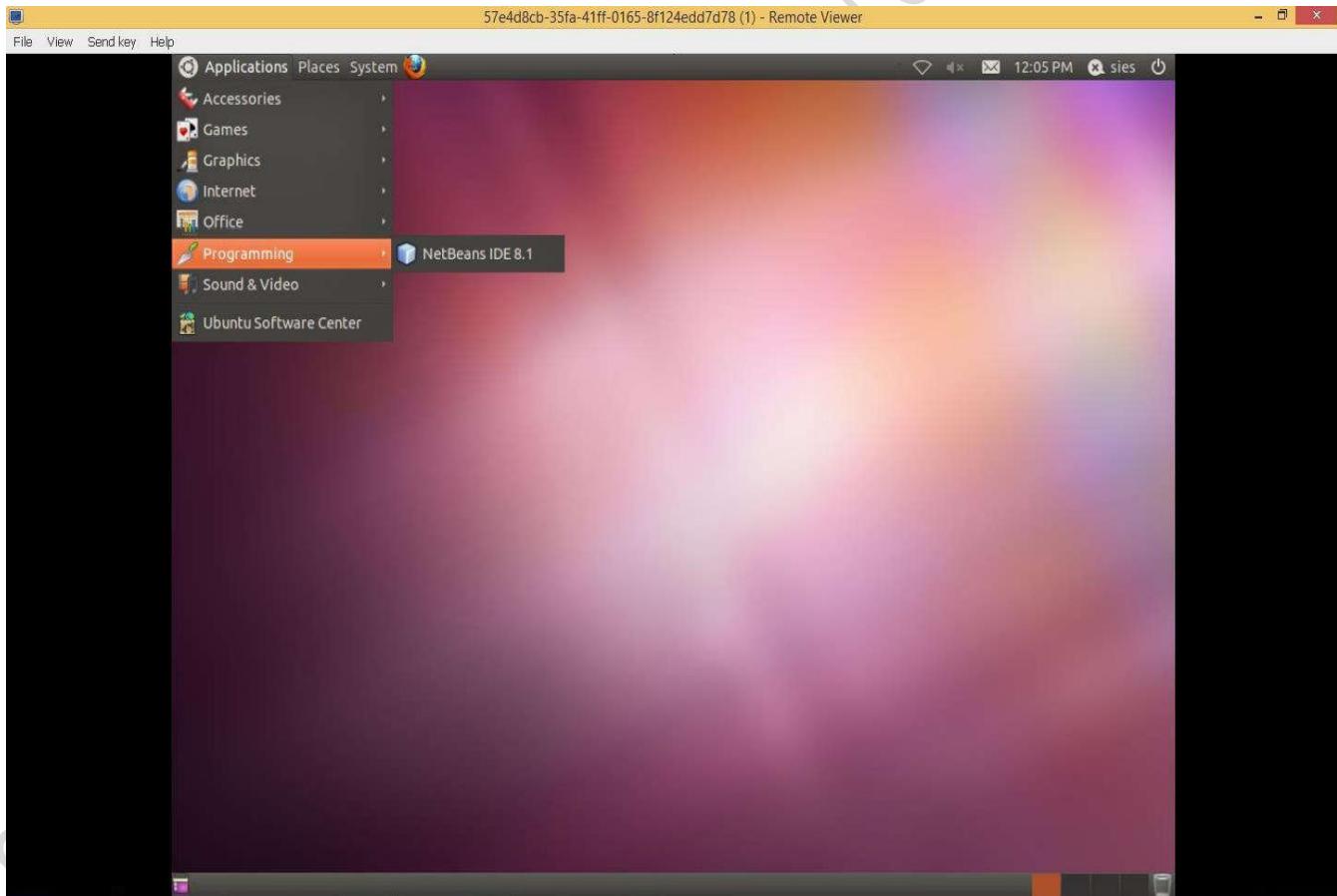


Practical-08

Implement FOSS-Cloud Functionality - VSI Platform as a Service (PaaS)

Software development Kits can be made available in the Virtual Machines that can be implemented as Platform as a Service.

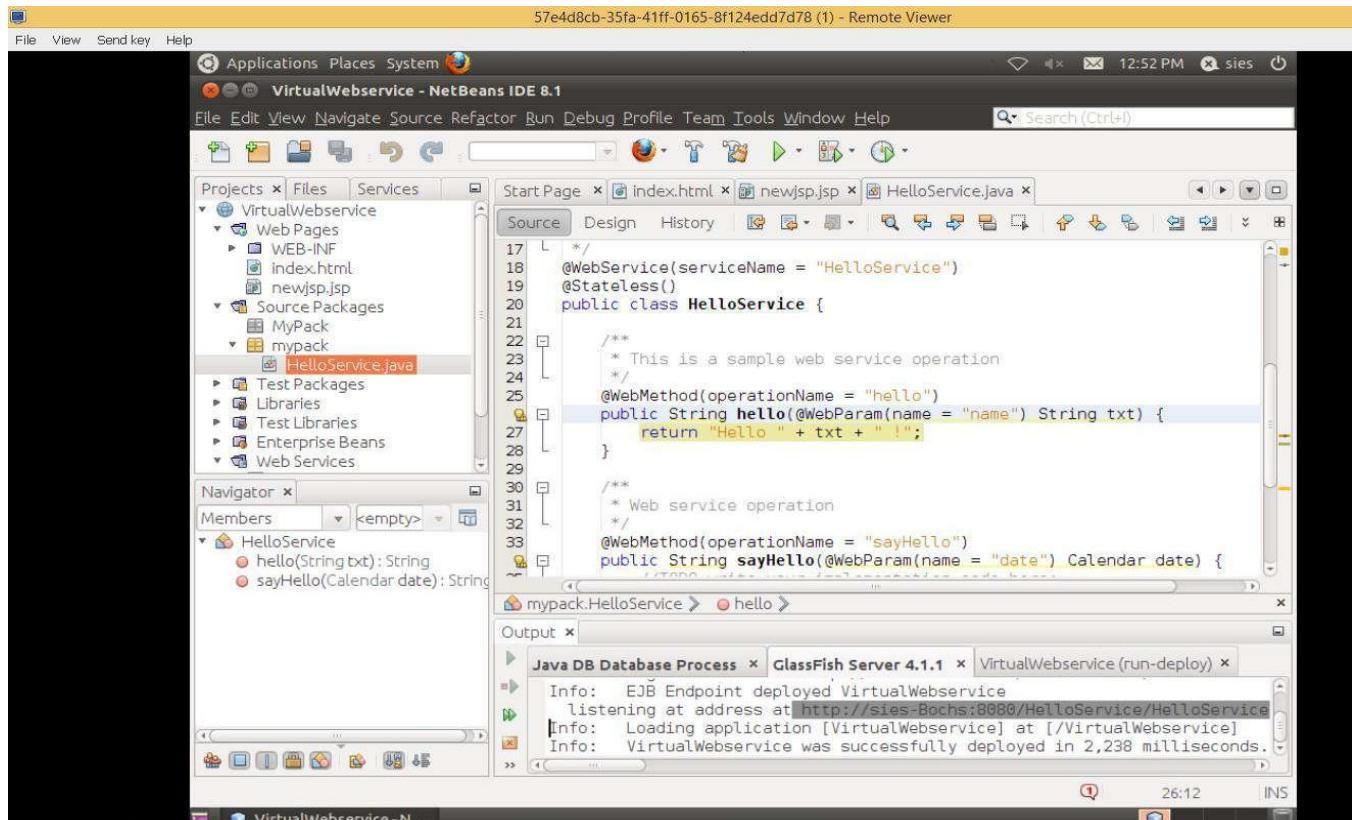
Installation of Netbeans, Eclipse, Visual Studio and DBMS can be done in the appropriate Virtual Machines.

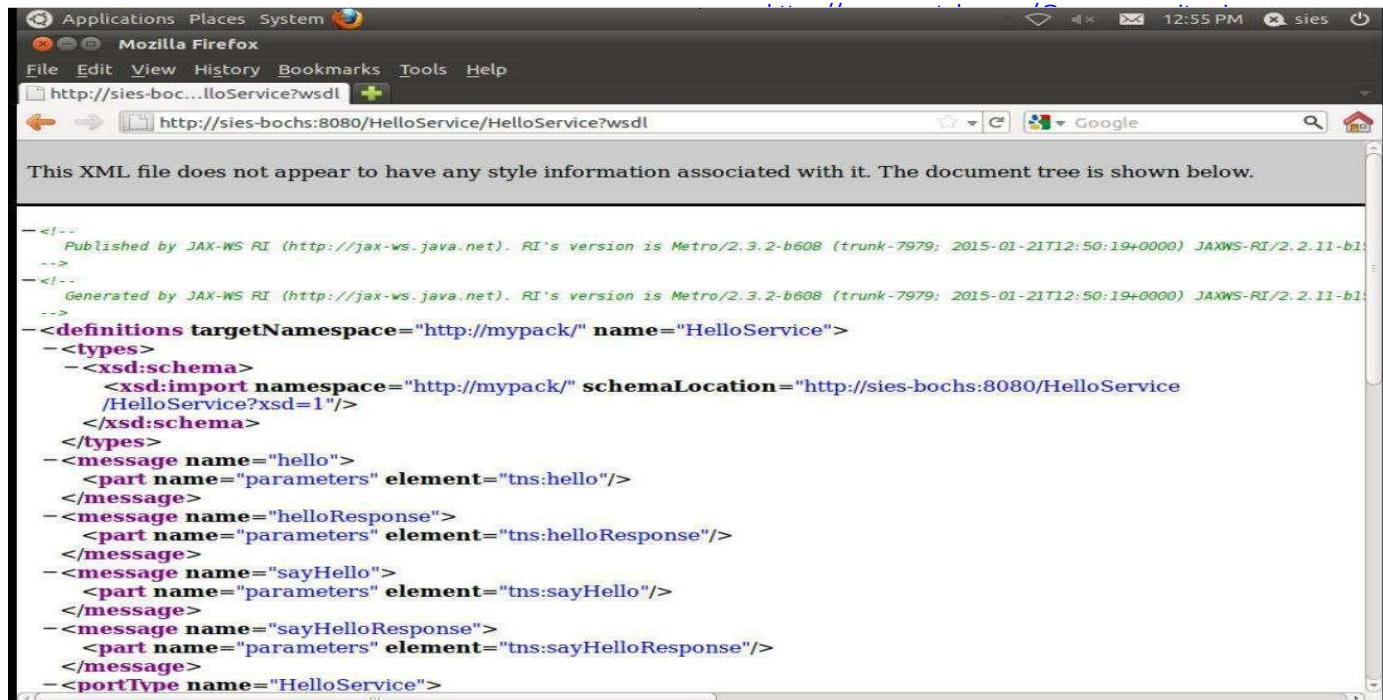


Aim: Implement FOSS-Cloud Functionality VSI Software as a Service (SaaS)

Applications created and deployed in the virtual machines can be accessed by outside world.

This can be implemented as Software as Service.





This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b1 -->
<!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b1 -->
<definitions targetNamespace="http://mypack/" name="HelloService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://mypack/" schemaLocation="http://sies-bochs:8080/HelloService/HelloService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="hello">
    <part name="parameters" element="tns:hello"/>
  </message>
  <message name="helloResponse">
    <part name="parameters" element="tns:helloResponse"/>
  </message>
  <message name="sayHello">
    <part name="parameters" element="tns:sayHello"/>
  </message>
  <message name="sayHelloResponse">
    <part name="parameters" element="tns:sayHelloResponse"/>
  </message>
  <portType name="HelloService">
```