SA PRATICALS GUIDE:

Practical No. 1:

Modelling using XADL

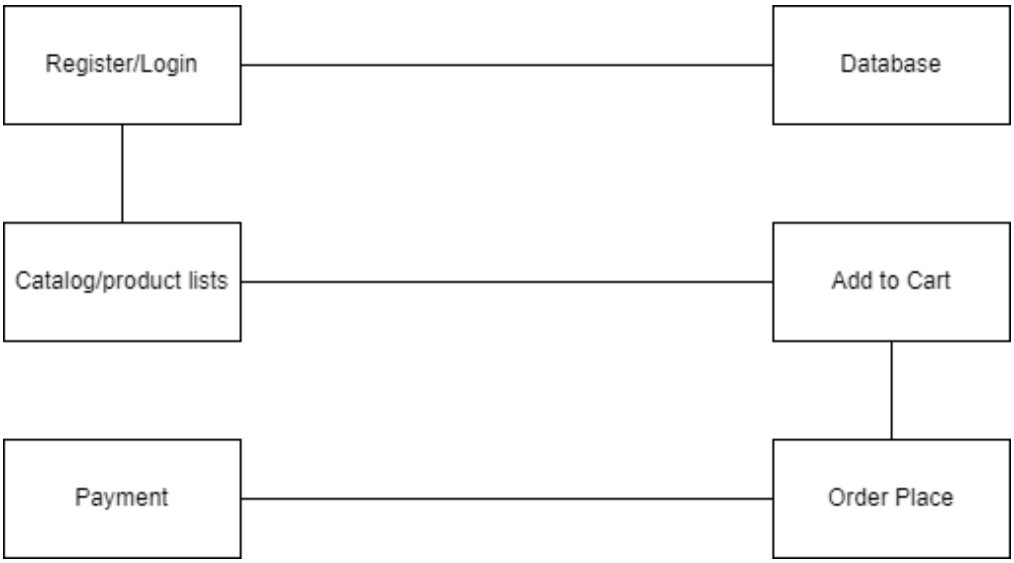
Problem Statement: E-Commerce Platform – (Clothing's Website)

Phase - I: FDL (Formal Description Language)

1. The website should start with a Login/Signup page.
2. The login page should be different for Users and Admins.
3. After logging in/ signing up the website should display the home page, which must include-
   - Search tab
   - Profile button
   - Cart button
   - Wish list button
   - An AI based Chabot
4. The E-commerce website should have shopping sections like Men, Women, Kids
5. The product description should have at least 2 images, name of the product, a small description, price, retailer's description, size chart, and customer reviews and ratings.
6. The user should be provided with three options for purchase like:
   - Buy Now (Direct purchase)
   - Add to cart (Later Purchase)
   - Add to Wish list (Wish list for later purchase)
7. Purchase options and steps should be available to the user when they decide to buy the product in the "Buy Now"/" Cart" section.
8. Revert of the successful or failed payment should be notified to the user through the registered e-mail of the customer.

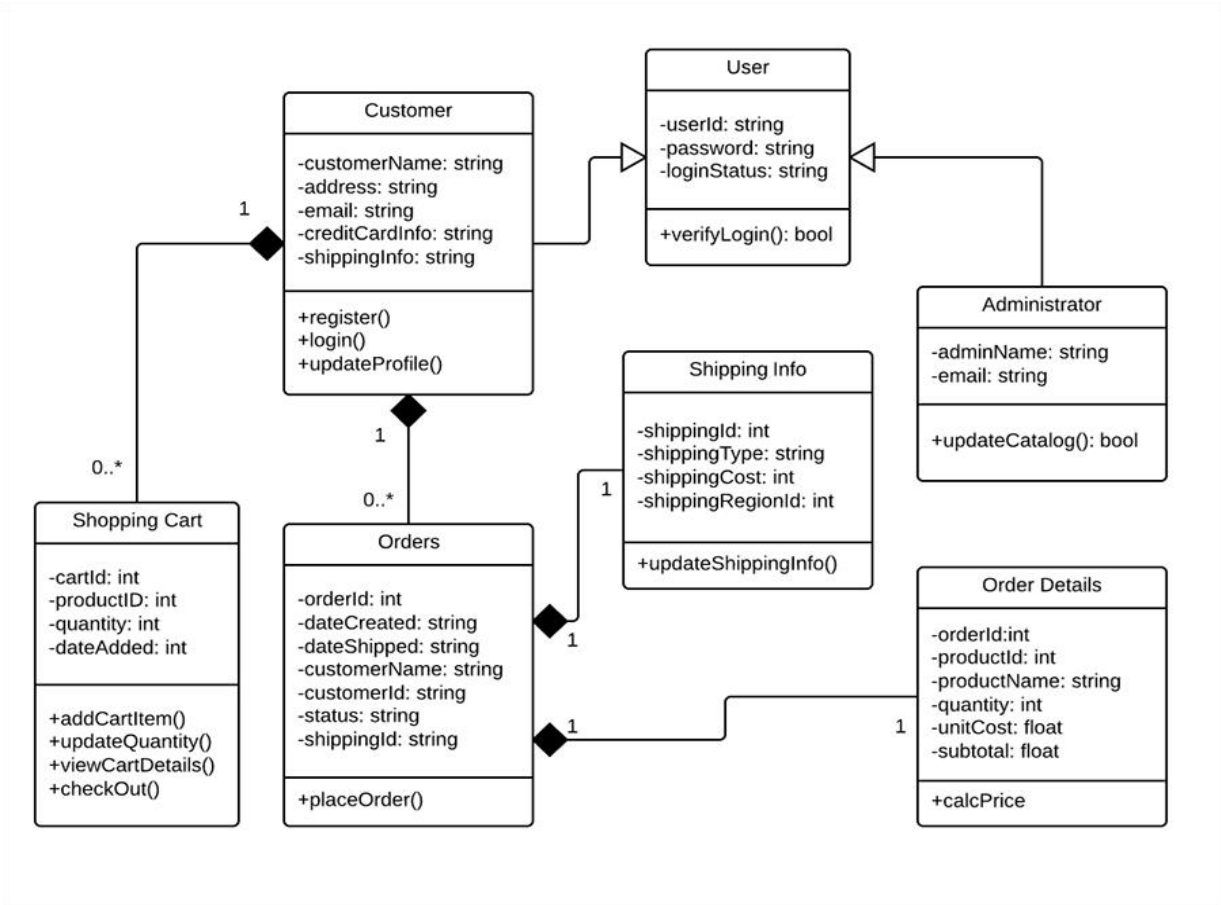
(Description for viva:)

Phase – II: Box & Line



Phase – III: UML

- Structural Diagram – (Class Diagram)



- Behavioral Diagram – (Use Case Diagram)

Practical No. 2:

Visualization using XADL 2.0

- **Component Diagram**

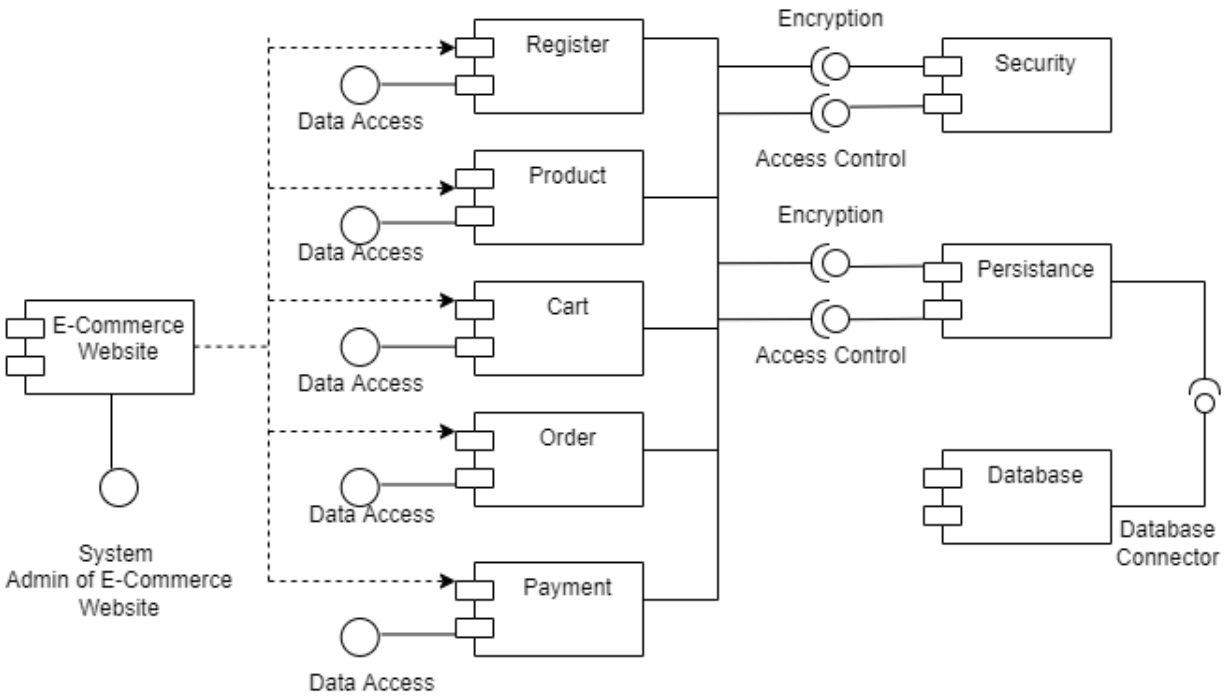


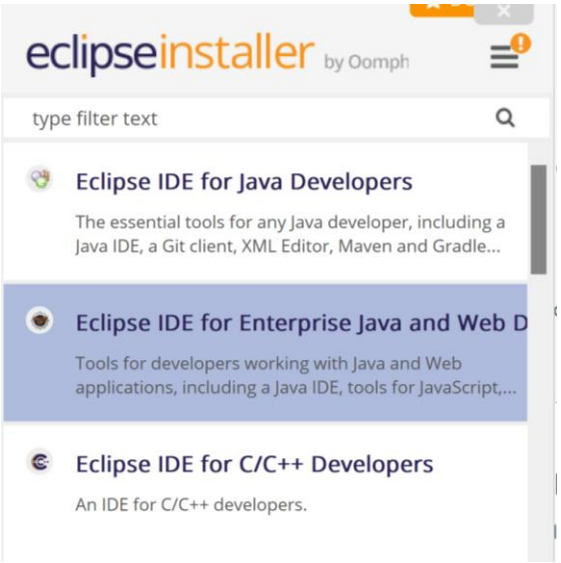
Fig 1: Component Diagram of E-Commerce Website

Practical No. 3:

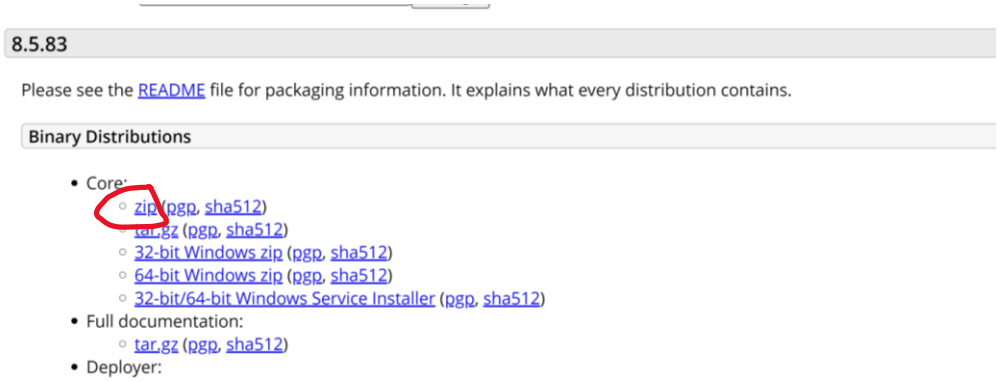
Creating Web services

Steps:

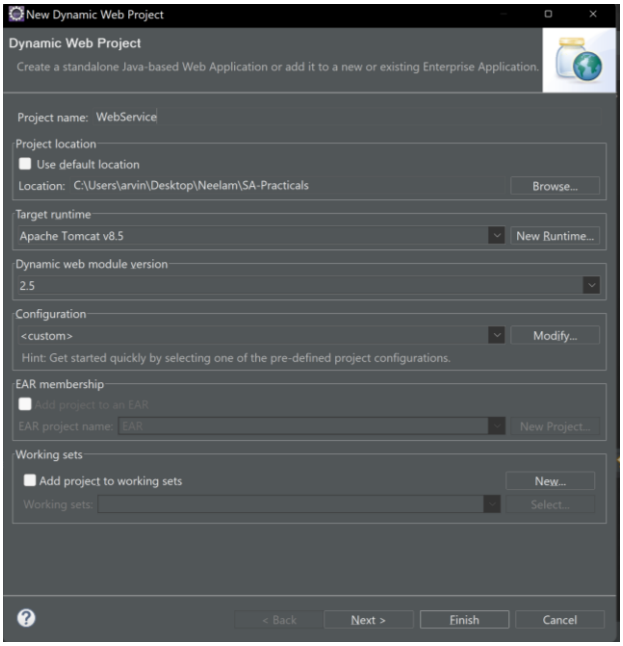
1. **Install Eclipse Java EE- Java**
2. **Choose Eclipse java for web developers**



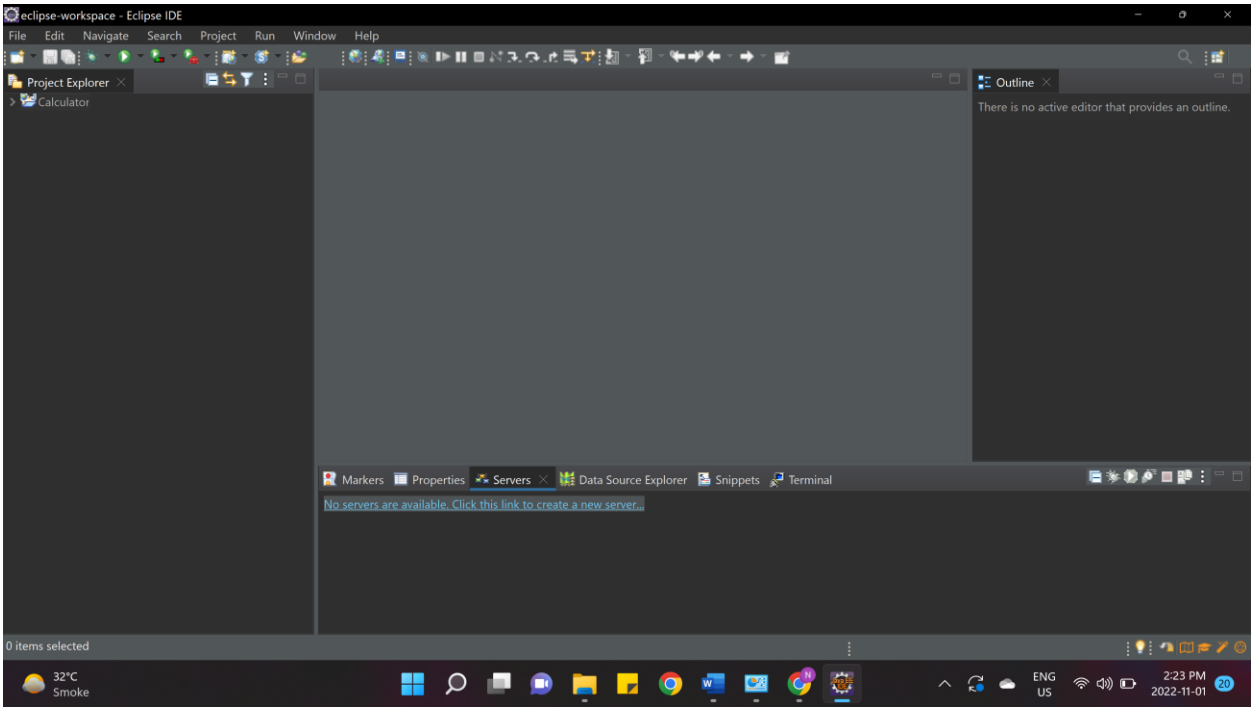
3. **Click on install**
4. **Download apache tomcat 8.5 download.**



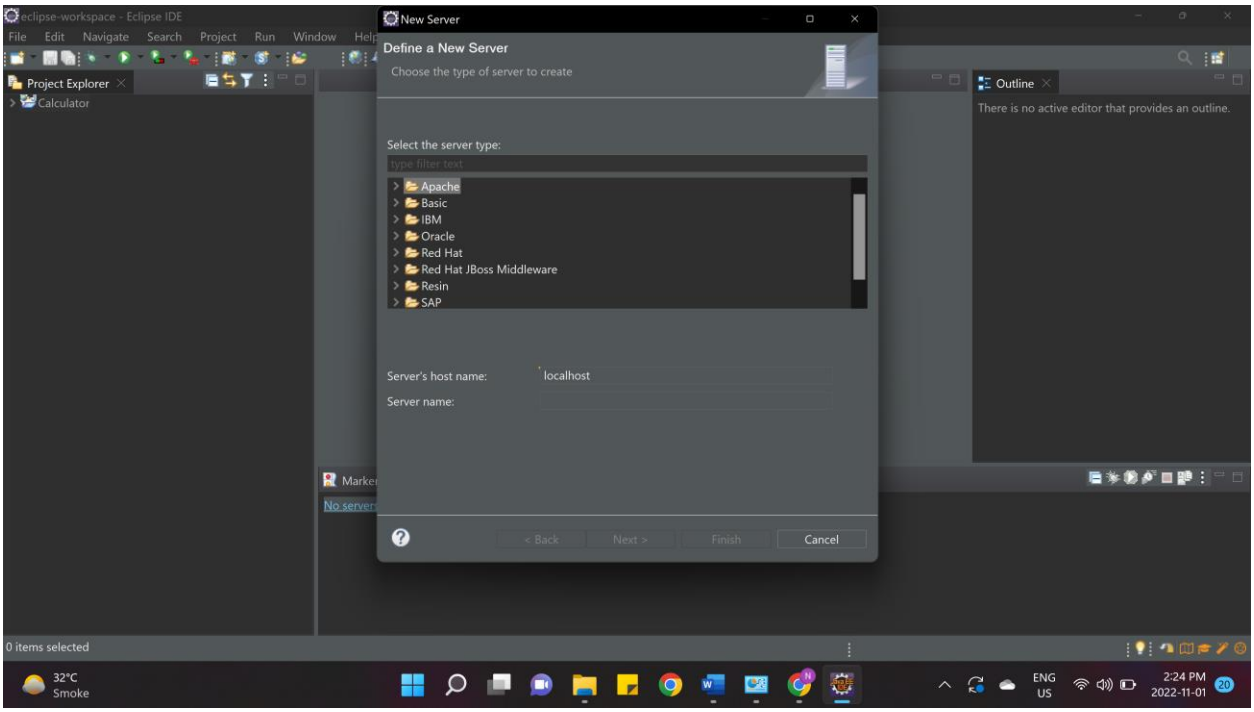
5.
6. **Extract the zip folder and then copy the path where you have extracted this zip folder.**
7. **C:\Users\arvin\Downloads\apache-tomcat-8.5.83**
8. **Go to file -> new-> dynamic web project.**

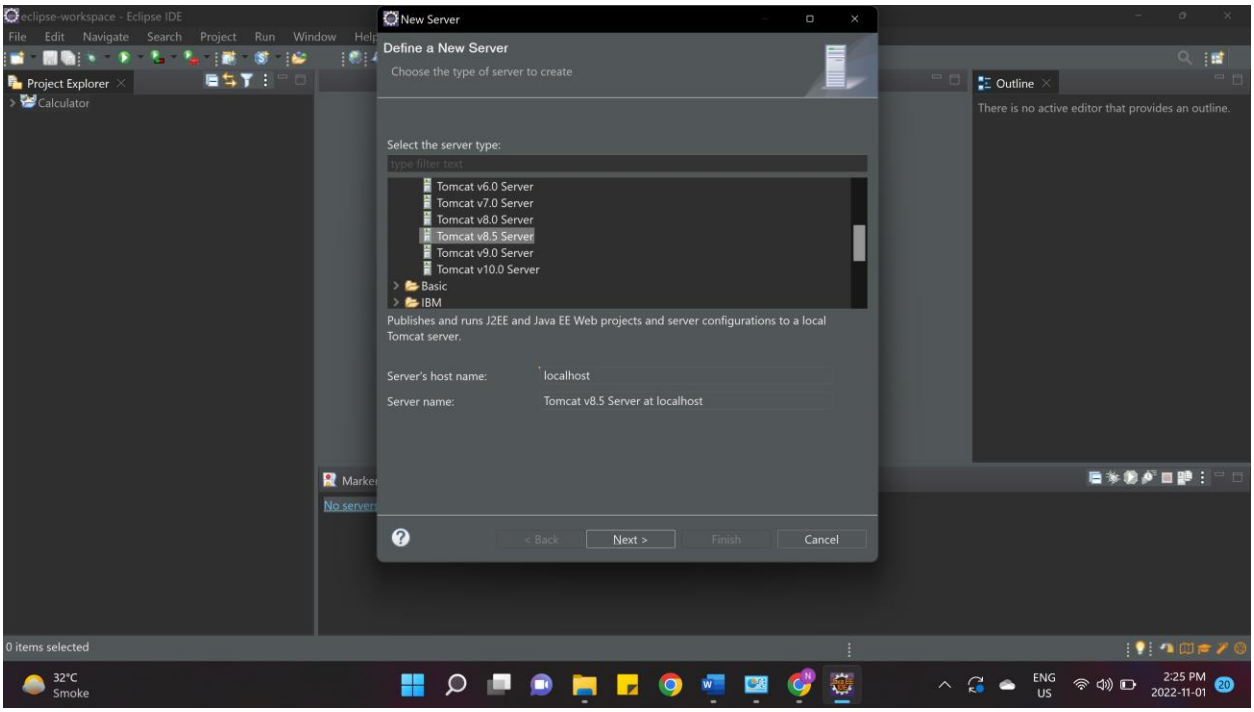
9. 

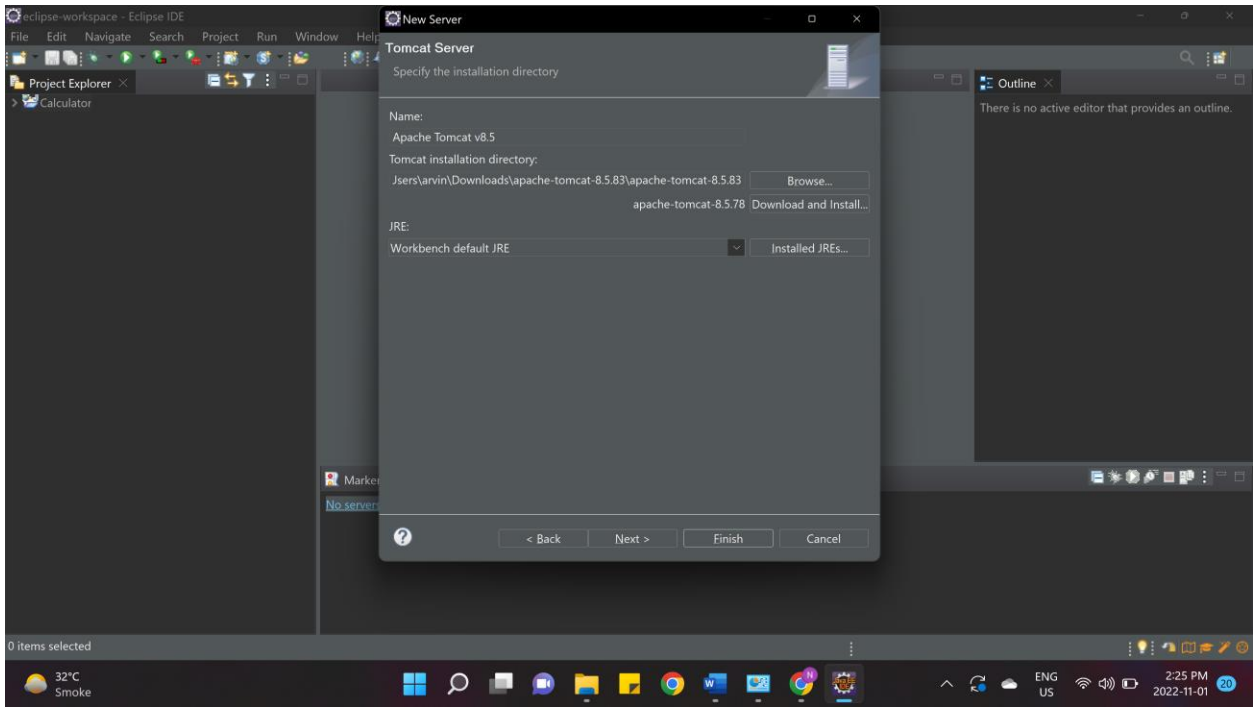
10. **Select Dynamic web module version 2.5**
11. **Apache 8.5**
12. **Select EAR Membership checkbox**
13. **Click on next and then finish**
14. **Click on the server option down**



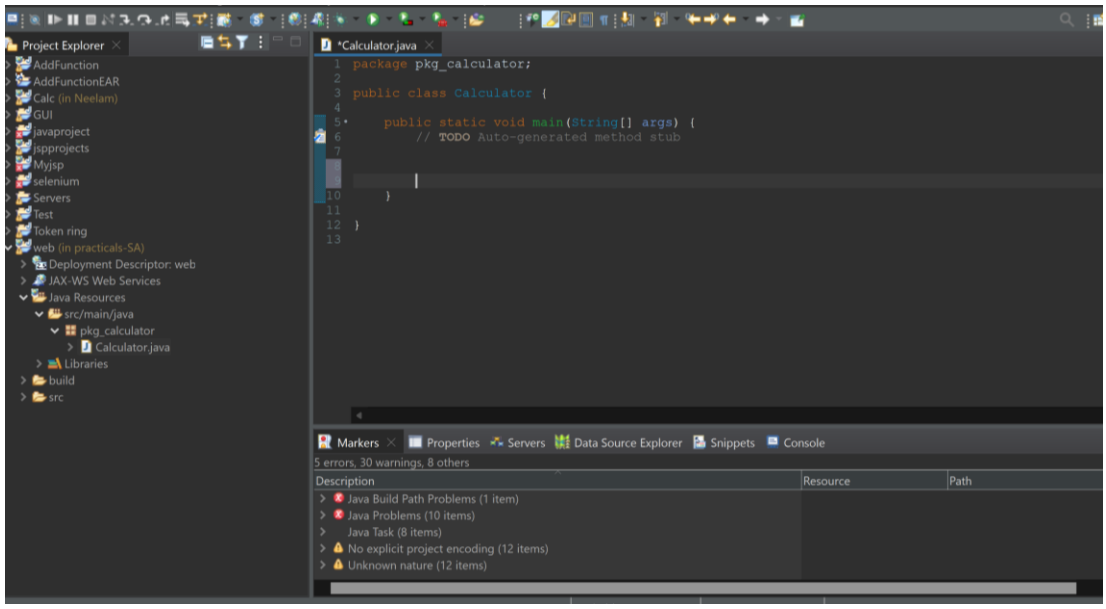
15.Select apache folder and then copy the path of apache 8.5 in the directory option given







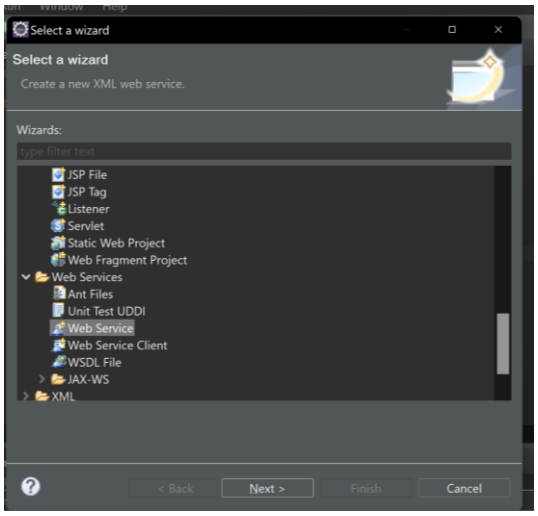
16. **Select next**
17. **Click on your project name**
18. **Click on add**
19. **Finish**
20. **Now create a java file by right clicking on project folder -> new-> class->create package (start the name with pkg_cal) and check radio button public static void main then right click on server and start**



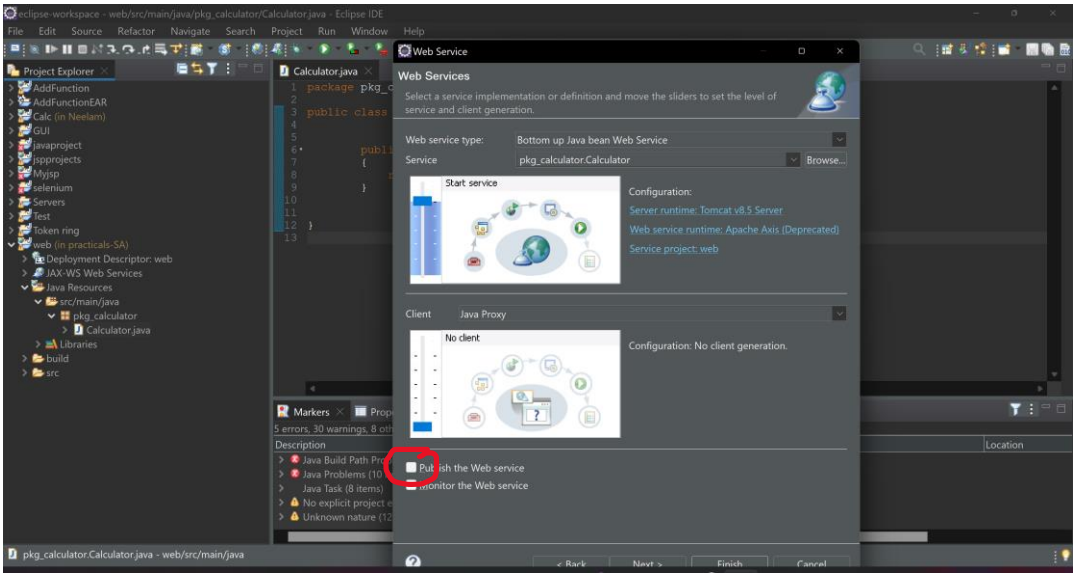
21. **(If you are not able to see project folder then click on window and then show view-> project explorer)**
22. **If you get an error 8080 problem then click right-> open-> change http code -> 8081 -> save it.**
23. **After successfully creating the java class-> right click on java class-> new-> other -> web services -> next.**

Code :

```
package pkg_calculator;

public class addition {

        public int add(int a, int b)

        {
                return a + b;
        }

}
```

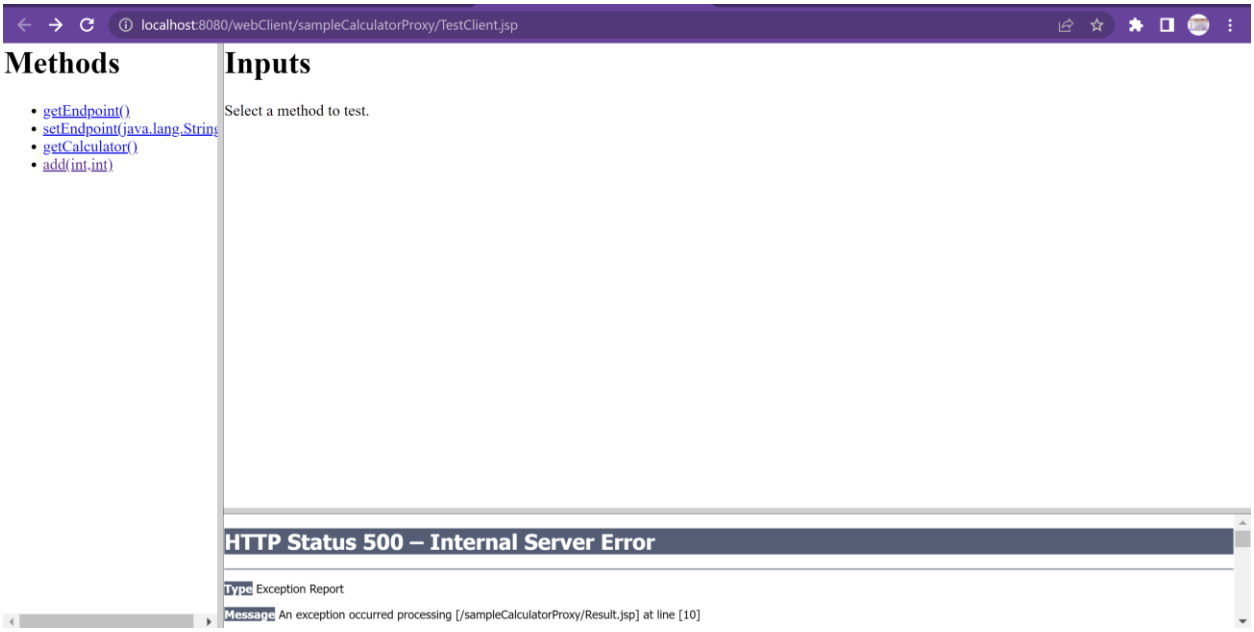


24. **Select second row bar to full-> click on PUBLISH THE WEB SERVICE check box given down-> click on next and then -> finish.**



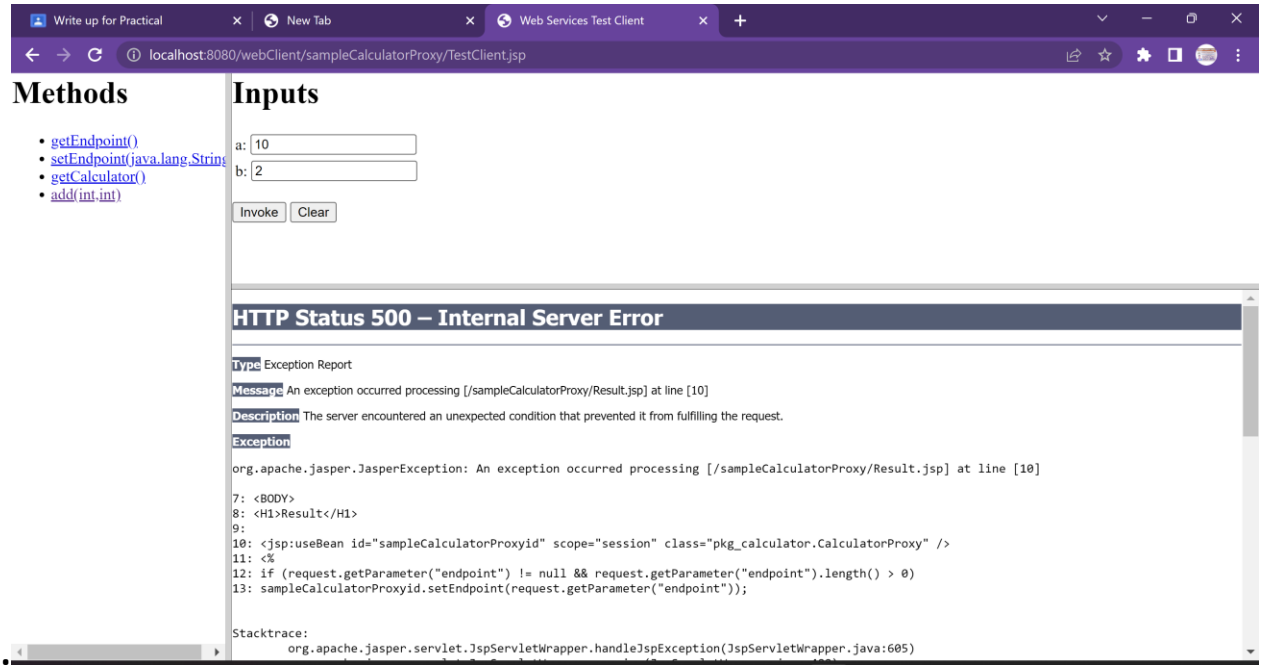
25.Click on next

26.Finally the screen will be visible



27.Click on your function you created at the left side its shown

28.


Write up for Practical
New Tab
Web Services Test Client
localhost:8080/webClient/sampleCalculatorProxy/TestClient.jsp
Methods
getEndpoint()
setEndpoint(java.lang.String
getCalculator()
add(int,int)
Inputs
a: 10
b: 2
Invoke
Clear
HTTP Status 500 – Internal Server Error
Type Exception Report
Message An exception occurred processing [/sampleCalculatorProxy/Result.jsp] at line [10]
Description The server encountered an unexpected condition that prevented it from fulfilling the request.
Exception
org.apache.jasper.JasperException: An exception occurred processing [/sampleCalculatorProxy/Result.jsp] at line [10]
7: <BODY>
8: <H1>Result</H1>
9:
10: <jsp:useBean id="sampleCalculatorProxyid" scope="session" class="pkg_calculator.CalculatorProxy" />
11: <%
12: if (request.getParameter("endpoint") != null && request.getParameter("endpoint").length() > 0)
13: sampleCalculatorProxyid.setEndpoint(request.getParameter("endpoint"));
Stacktrace:
org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:605)

Practical No. 4:

Integrate Software Components using middleware

Steps

1. **Create a new web dynamic project**
2. **Create two classes namely client(ClientClass.java), server(ServerClass.java).**
3. **Create an interface class by going into -> file-> new-> interface-> write name and click on enter.**

4. **Code:**

InterfaceCalci.java (interface class code)

```
package pkg_integrate;

public interface InterfaceCalci {


        public int add(int a , int b);
        public int sub(int a , int b);
        public int mul(int a , int b);
        public int div(int a , int b);

}
```

ServerClass.java

```
package pkg_integrate;
```

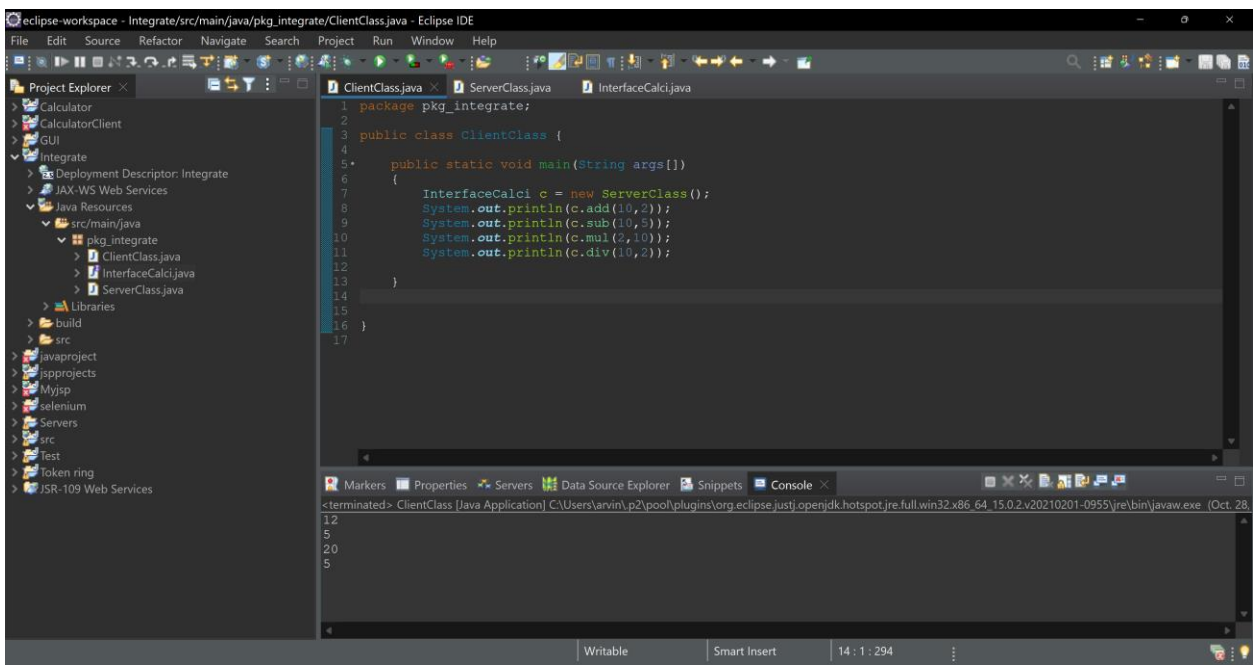
```
public class ServerClass implements InterfaceCalci {

        public int add(int a , int b)
        {
                return a+b;
        }
        public int sub(int a , int b)
        {
                return a-b;
        }
        public int mul(int a , int b)
        {
                return a*b;
        }
        public int div(int a , int b)
        {
                return a/b;
        }


}
```

ClientClass.java

```
package pkg_integrate;

public class ClientClass {

        public static void main(String args[])
        {
                InterfaceCalci c = new ServerClass();
                System.out.println(c.add(10,2));
                System.out.println(c.sub(10,5));
```

```
            System.out.println(c.mul(2,10));
            System.out.println(c.div(10,2));

        }


    }
```

5. **Now go to client class file and click on the above green run button-> run as-> java application , and then output will be printed on the console.**
6. **Output:**



Practical No. 5:

Use middleware to implement connectors

1. Make sure you have Java version 1.8 with rmi registry.exe in bin folder, check for the same.

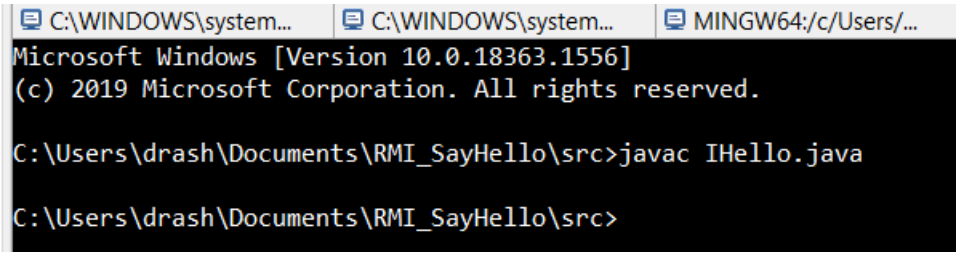
2. Create a new java project --> RMI_Demo

3. Right click on project and add new Interface --> IHello (Write interface code)

4. Paste below code in IHello.java

```
import java.rmi.*;

public interface IHello extends Remote{

	public String message() throws RemoteException;

}
```

5. Right click on IHello.java --> Show in Local terminal --> Terminal and when console opens compile your code as follows:



6. Add new class --> HelloImpl.java (write implementation code)

7. Paste below code in HelloImpl.java:

```
import java.rmi.*;
```

```
import java.rmi.server.*;



public class HelloImpl extends UnicastRemoteObject
        implements IHello{

    public HelloImpl() throws RemoteException {
        //There is no action need       in this moment.
    }

    public String message() throws RemoteException {
        return ("Hello there, student.");
    }
}
```

8. Compile the HelloImpl.java as follows:

```
C:\Users\drash\Documents\RMI_SayHello\src>javac HelloImpl.java

C:\Users\drash\Documents\RMI_SayHello\src>
```

9. Add class HelloServer and paste below code:

```
import java.rmi.*;

public class HelloServer {
```

```
    private static final String host = "localhost";

    public static void main(String[] args) throws Exception {
        //** Step 1
        //** Declare a reference for the object that will be implemented
        HelloImpl temp = new HelloImpl();

        //** Step 2
        //** Declare a string variable for holding the URL of the object's name
        String rmiObjectName = "rmi://" + host + "/Hello";

        //Step 3
        //Binding the object reference to the object name.
        Naming.rebind(rmiObjectName, temp);

        //Step 4
        //Tell to the user that the process is completed.
        System.out.println("Binding complete...\n");
    }
}
```

10. Compile server class.

11. Add class HelloClient and paste below code:

```
import java.rmi.ConnectException;
import java.rmi.Naming;


public class HelloClient
{
    private static final String host = "localhost";
```

```
    public static void main(String[] args)
    {
        try
        {
            //We obtain a reference to the object from the registry and next,
            //it will be typecasted into the most appropiate type.
            IHello greeting_message = (IHello) Naming.lookup("rmi://"
                    + host + "/Hello");

            //Next, we will use the above reference to invoke the remote
            //object method.
            System.out.println("Message received: " +
                    greeting_message.getGreetingMessage());
        }
        catch (ConnectException conEx)
        {
            System.out.println("Unable to connect to server!");
            System.exit(1);
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
            System.exit(1);
        }
    }
}
```

12. Compile client class.

13. Start the rmi registry as follows:

```
C:\Users\drash\Documents\RMI_SayHello\src>start rmiregistry

C:\Users\drash\Documents\RMI_SayHello\src>
```

It starts another console which gets displayed as follows:

```
C:/Users/drash/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.3.v20220515-1416/jre/bin\rmiregistry.exe
WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release
```

14. Now run the Client code as follows:

```
C:\Users\drash\Documents\RMI_Hello_new\src>java HelloClient.java

java.rmi.NotBoundException: Hello
```

Practical No. 6:

Wrapper to connect two applications with different architectures.

Steps:

1. **Write DSender code**
2. **Write DReceiver code**
3. **On command line write java DSender enter**
4. **It will show "Enter your Message"**
5. **Write Message and click on Enter**
6. **Message will be then send to the receiver**

Code:

DSender.java

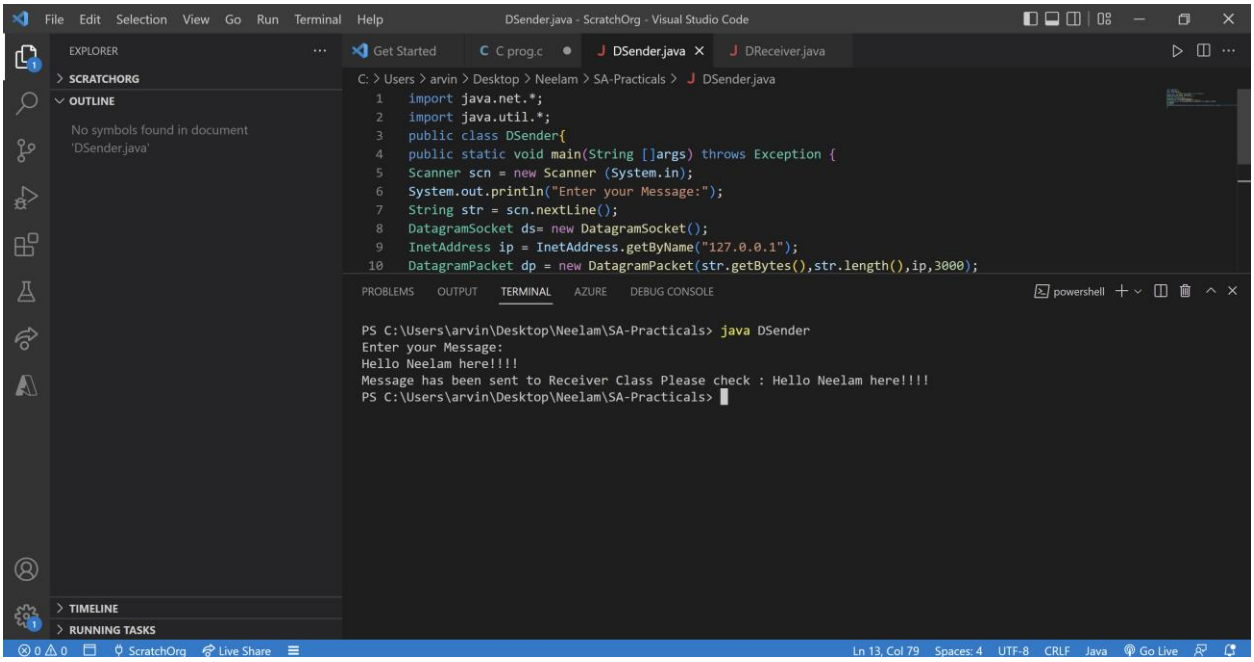
```
import java.net.*;
import java.util.*;
public class DSender{
public static void main(String []args) throws Exception {
Scanner scn = new Scanner (System.in);
System.out.println("Enter your Message:");
String str = scn.nextLine();
DatagramSocket ds= new DatagramSocket();
InetAddress ip = InetAddress.getByName("127.0.0.1");
DatagramPacket dp = new DatagramPacket(str.getBytes(),str.length(),ip,3000);
ds.send(dp);
ds.close();
System.out.println("Message has been sent to Receiver Class Please check : "
+str);
}
}
```

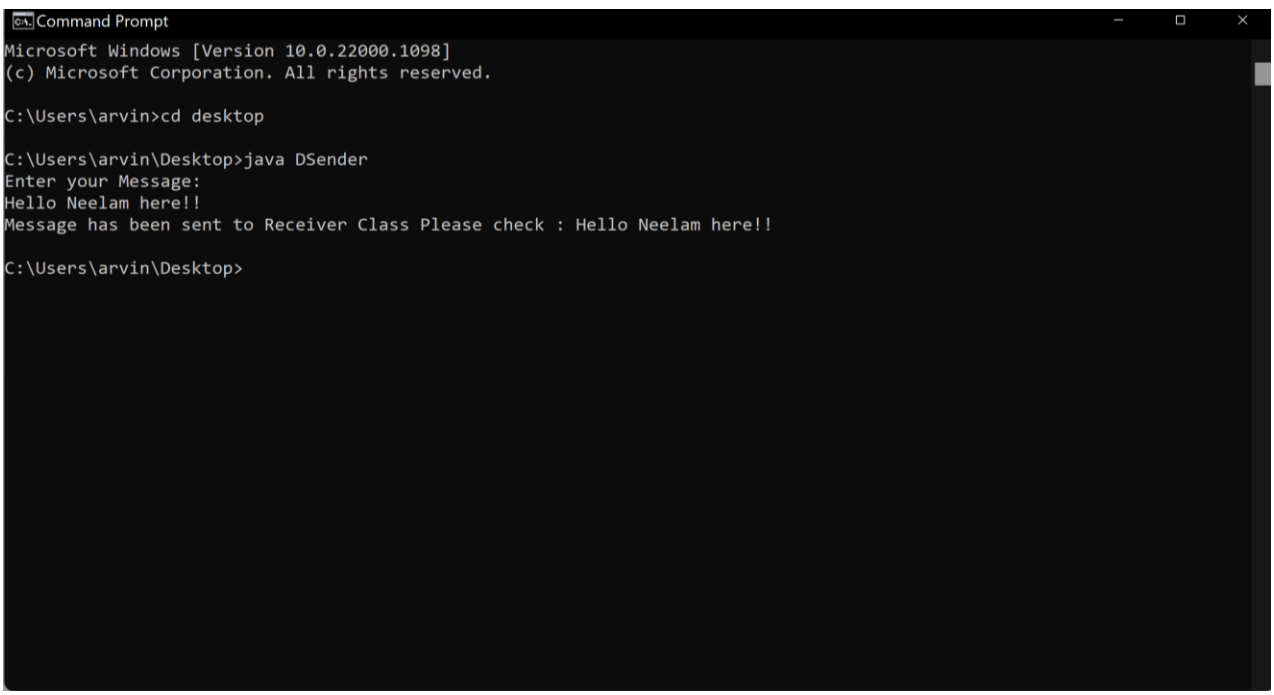
DReceiver.java

```
import java.net.*;
public class DReceiver {
public static void main (String [] args) throws Exception {
System.out.println("Waiting for sender to send the message ");
DatagramSocket ds = new DatagramSocket(3000);
byte[] buf = new byte[1024];
DatagramPacket dp = new DatagramPacket (buf,1024);
ds.receive(dp);
String str = new String (dp.getData(), 0, dp.getLength());
System.out.println(str);
ds.close() ;
System.out.println(" Message received Successfully ");
}
}
```

Output:

59_BECOMP_C_Neelam Yadav

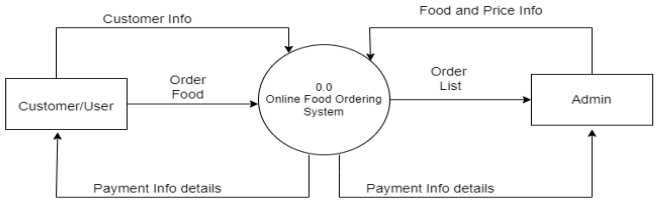




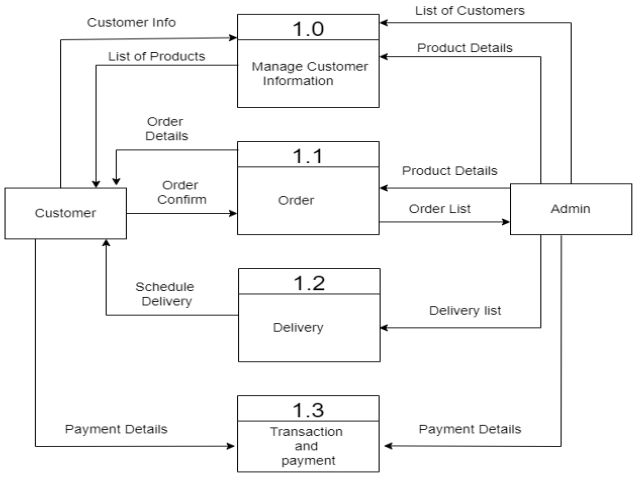
Practical No. 7:

Identifying Design requirements for an Architecture for any specific domain.

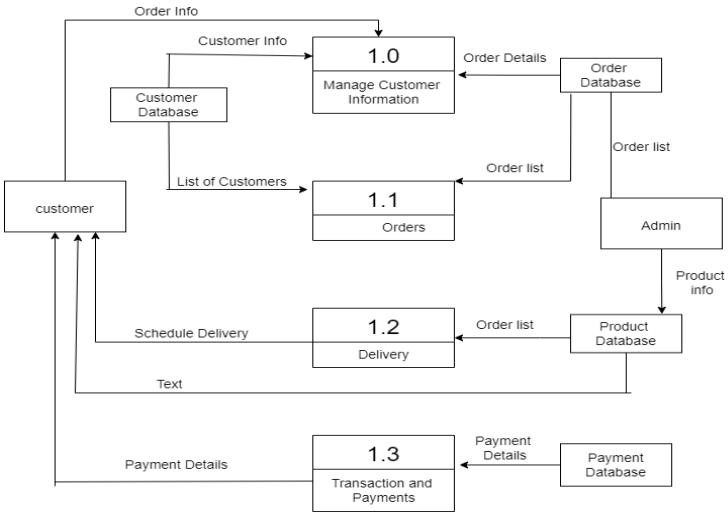
Zero Level DFD



1-Level DFD



2 Level DFD



Practical No. 8:

Identifying System requirements for an Architecture for any specific domain.

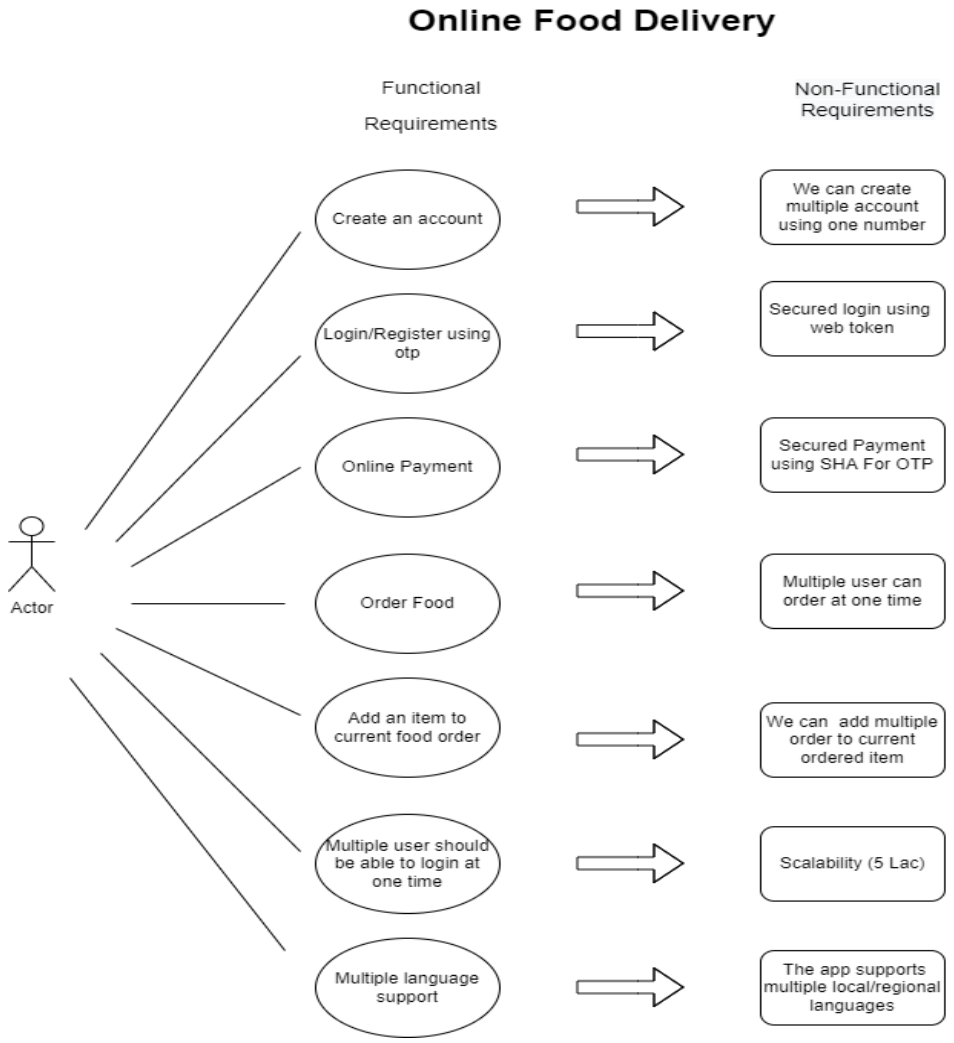
(SRS)

Practical No. 9:

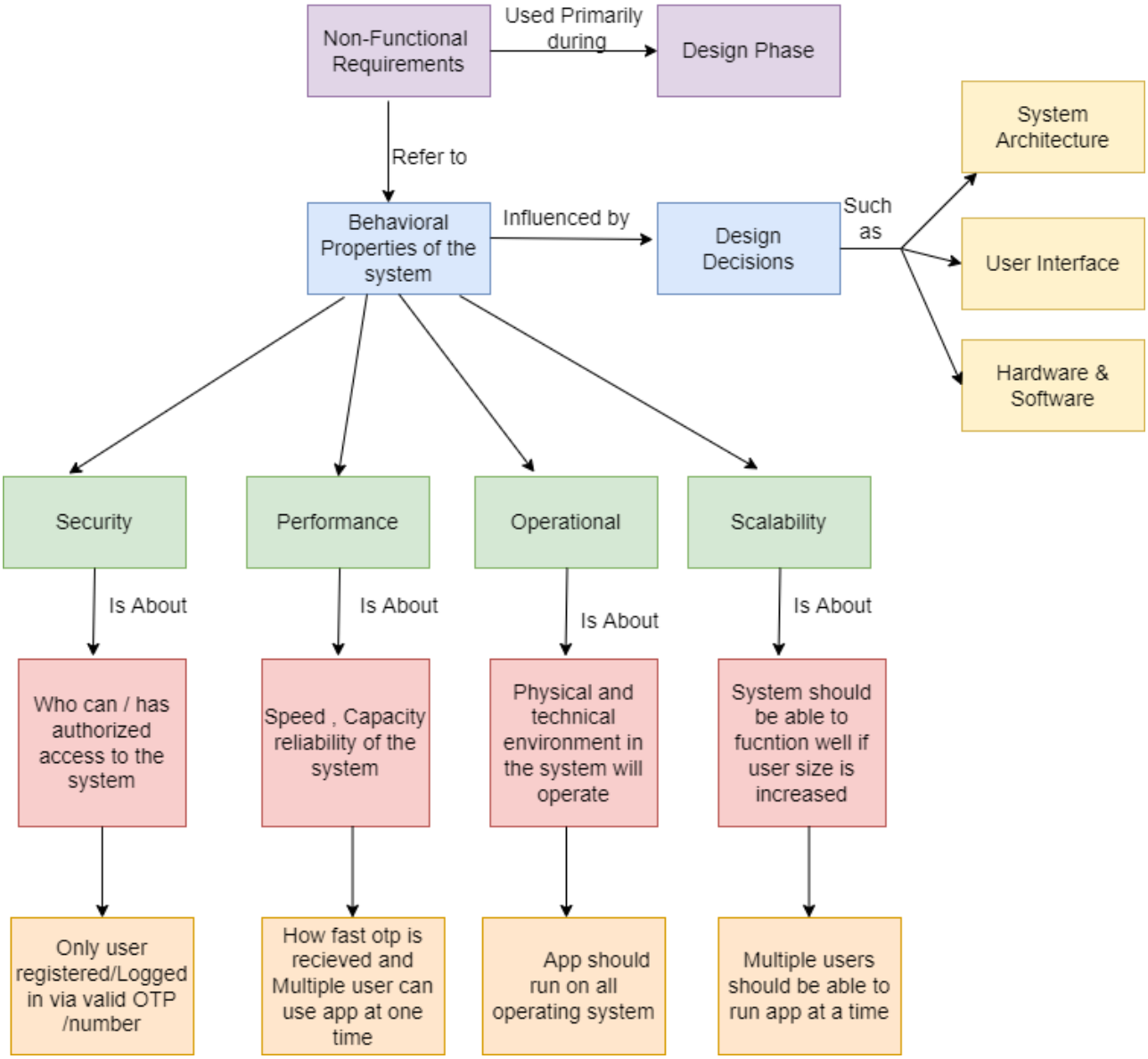
Mapping of Non-Functional requirements

Output:

NFR TO FR



NFR


Non-Functional Requirements
Used Primarily during
Design Phase
Refer to
Behavioral Properties of the system
Influenced by
Design Decisions
Such as
System Architecture
User Interface
Hardware & Software
Security
Performance
Operational
Scalability
Is About
Is About
Is About
Is About
Who can / has authorized access to the system
Speed , Capacity reliability of the system
Physical and technical environment in the system will operate
System should be able to fucntion well if user size is increased
Only user registered/Logged in via valid OTP /number
How fast otp is recieved and Multiple user can use app at one time
App should run on all operating system
Multiple users should be able to run app at a time

59_BECOMP_C_Neelam Yadav