```
    }
}
```

SUPPLIER:

```
Public class supplier
{
    Public integer supply;
    Public void available product()
    }
```

RESULT:

Thus thethe diagrams [Usecase, Activity, Sequence, Collaboration, Class, Statechart, Component, Deployment, package] for foreign trading system has been designed, executed and output is verified.

Ex.no 13	BPO MANAGEMENT SYSTEM
Date:	

AIM:

To draw the diagrams [Use case , Class, Activity, Sequence, Collaboration, State Chart, Component, Deployment, package] for the BPO Management System .

OOAD LAB

REGISTER NO:

SOFTWARE REQUIREMENTS SPECIFICATION

	SOFTWARE REQUIREMENTS SPECIFICATION
1.0	Hardware Requirements
1.1	Software Requirements
1.2	Problem Analysis and Project Plan
1.3	Project description
1.4	Reference

1.0 HARDWARE REQUIREMENTS:

Intel Pentium Processor I3/I5

1.1 SOFTWARE REQUIREMENTS:

Rational rose /Argo UML

1.3 PROJECT DESCRIPTION:

This software is designed to know about the process that were taking place in the BPO office. This system holds the details of the customer who and all approaches to it.It is managed by the central systems.

1.4 REFERENCES:

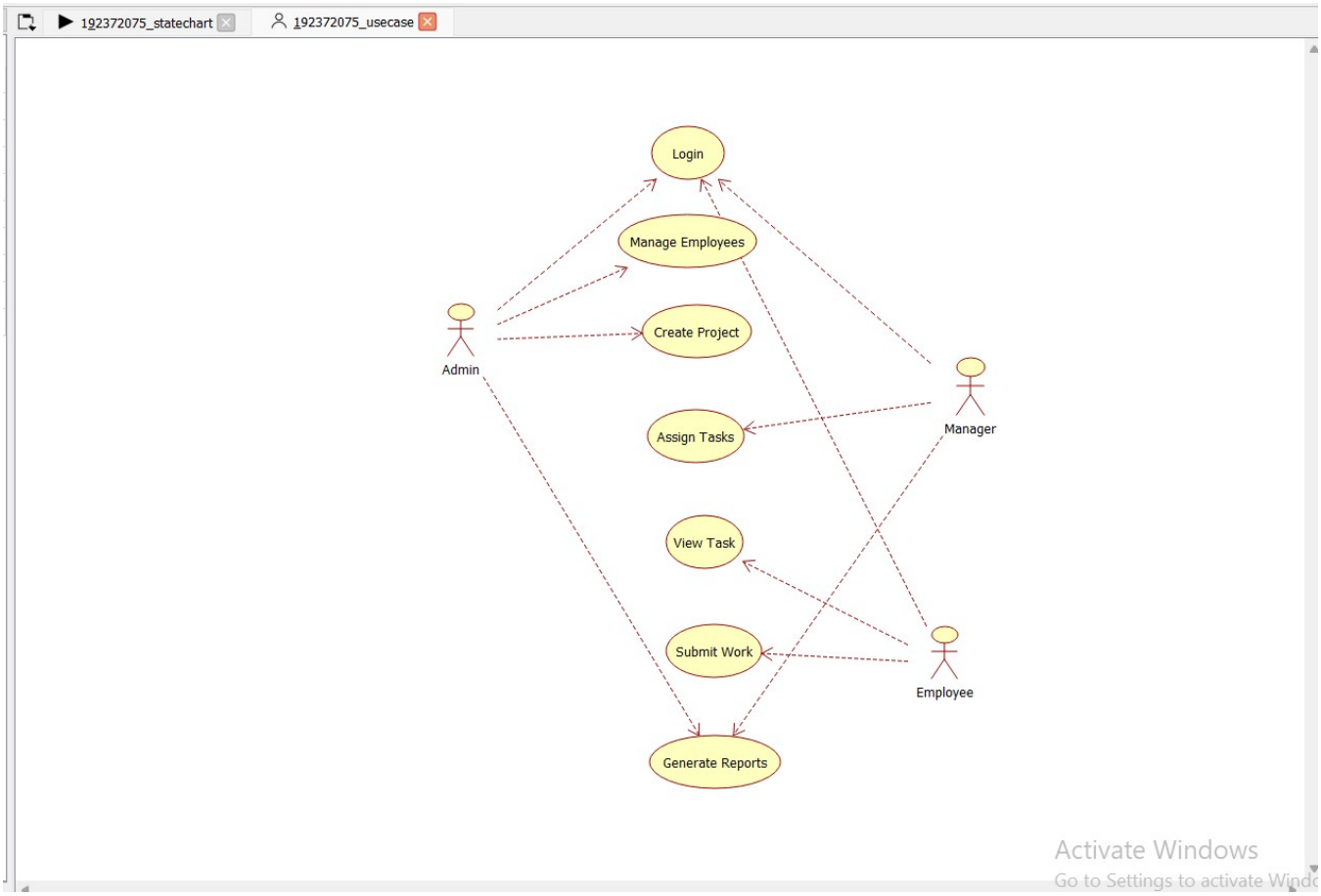
IEEE Software Requirement Specification format.

USECASE DIAGRAM:

This diagram will contain the attributes as start point, end point, decision box as given below

ACTORS: Purchase product, Server, Central system

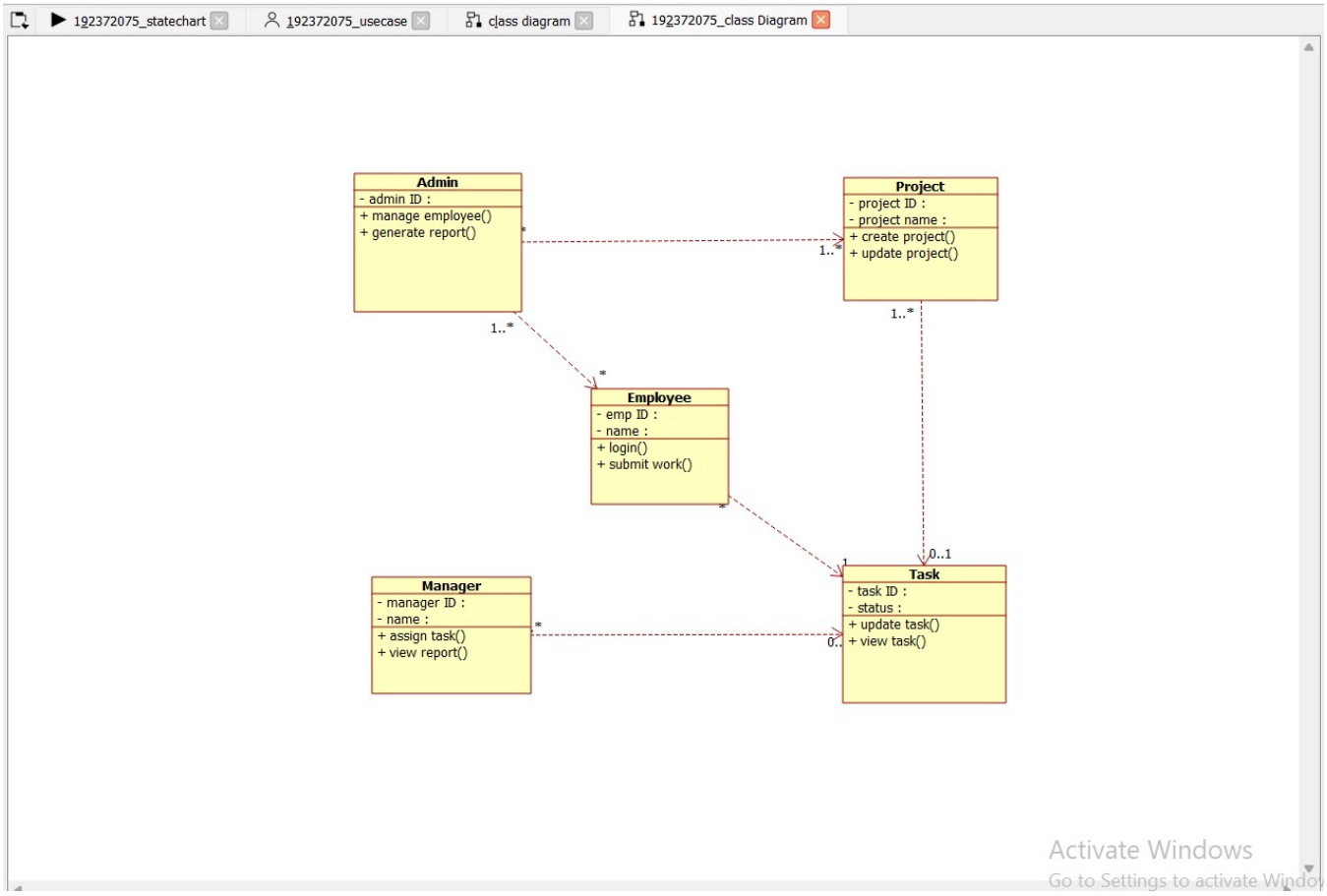
USECASE: Product, Voice, Non-Voice, Indian office, Employee, Feedback.



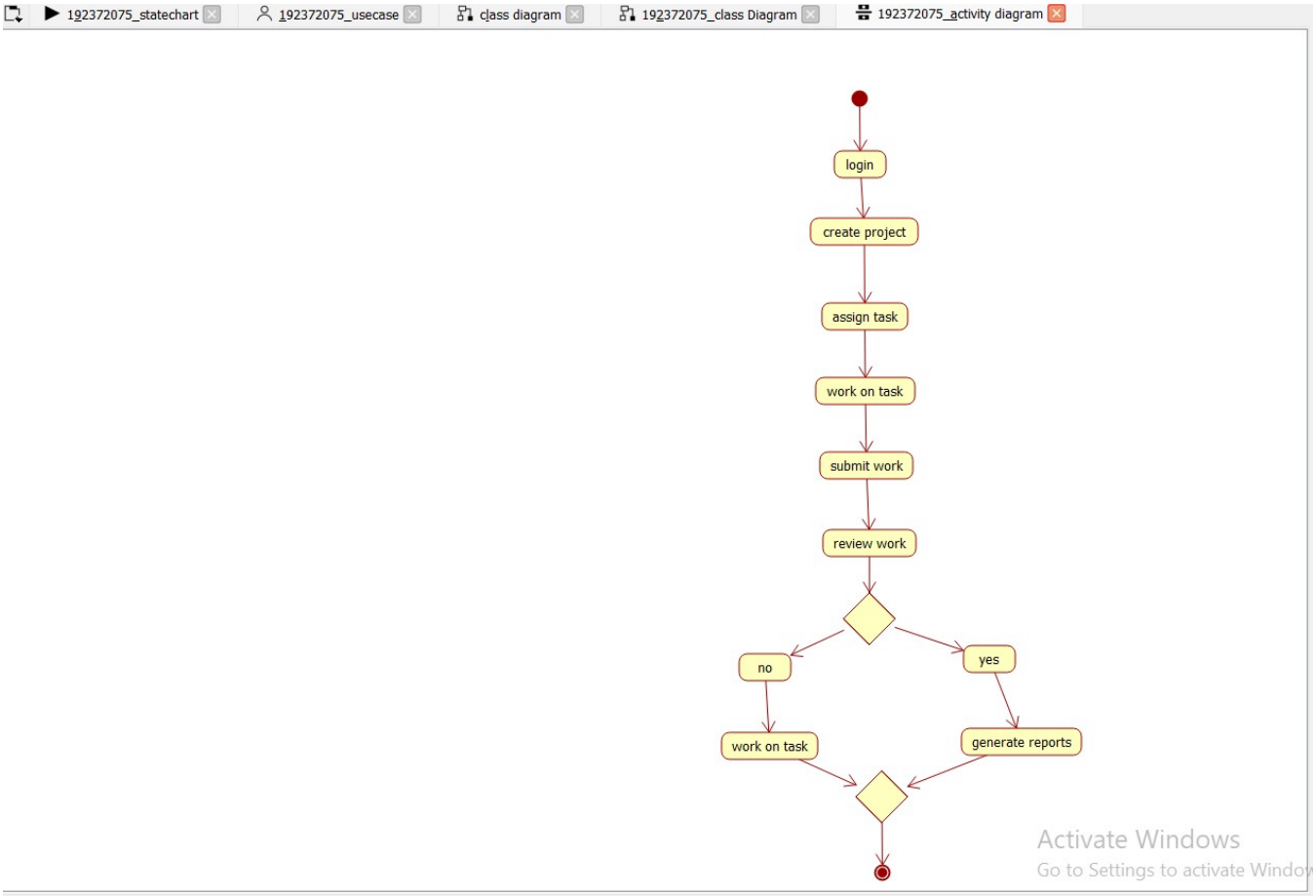
CLASS DIAGRAM:

This Diagram consists of the following classes, attributes and their operations.

CLASSES	ATTRIBUTES	OPERATIONS
Central System	Store, update	Storing(), updating()
Dealer	Employee name	Delivery()
Customer	Details	Feedback()



ACTIVITY DIAGRAM:



This diagram will contain the activities as start point, end point, decision boxes as given below

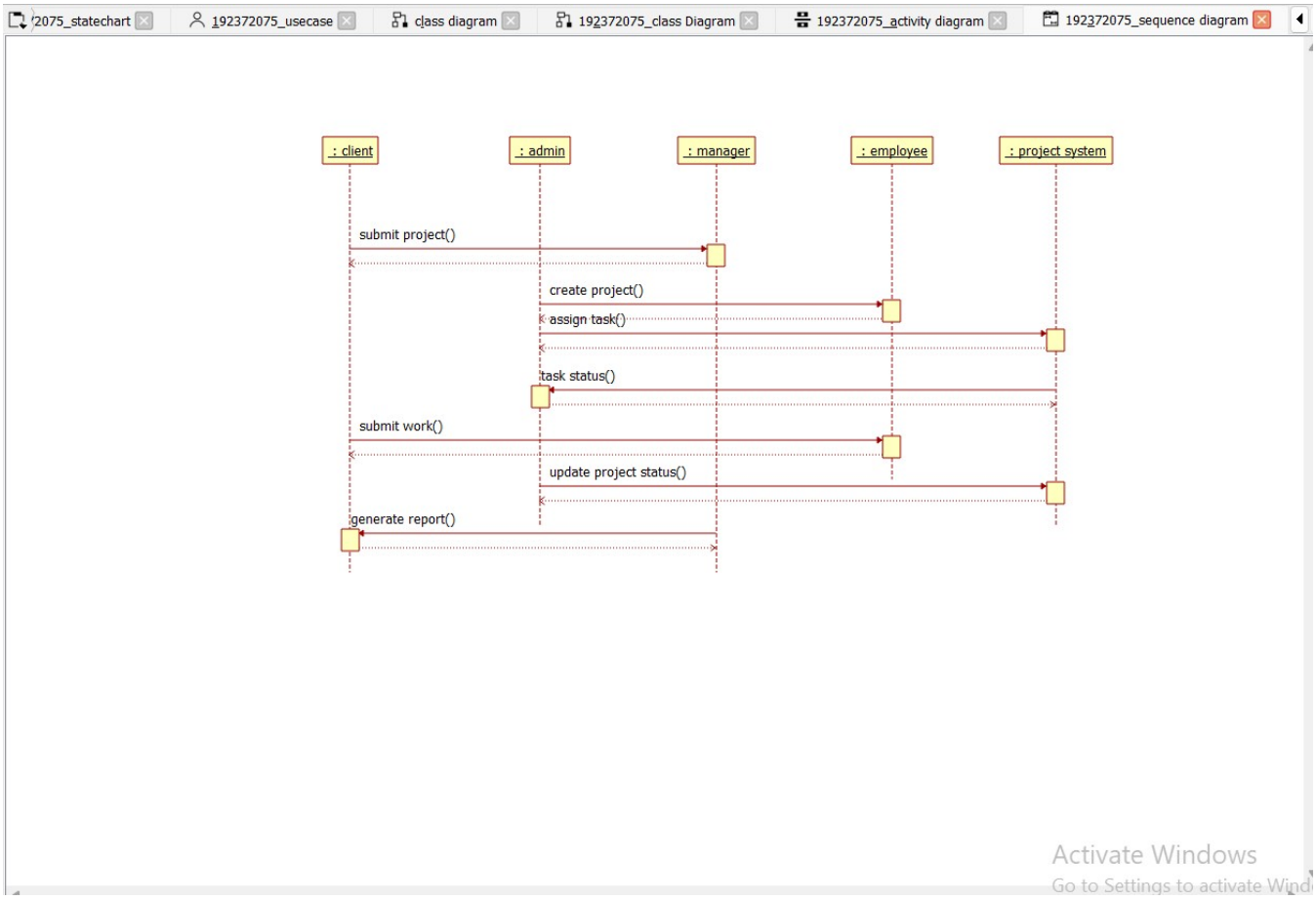
ACTIVITIES: Purchase Product, On call, On chat

DECISION BOX: Option to check

SEQUENCE DIAGRAM:

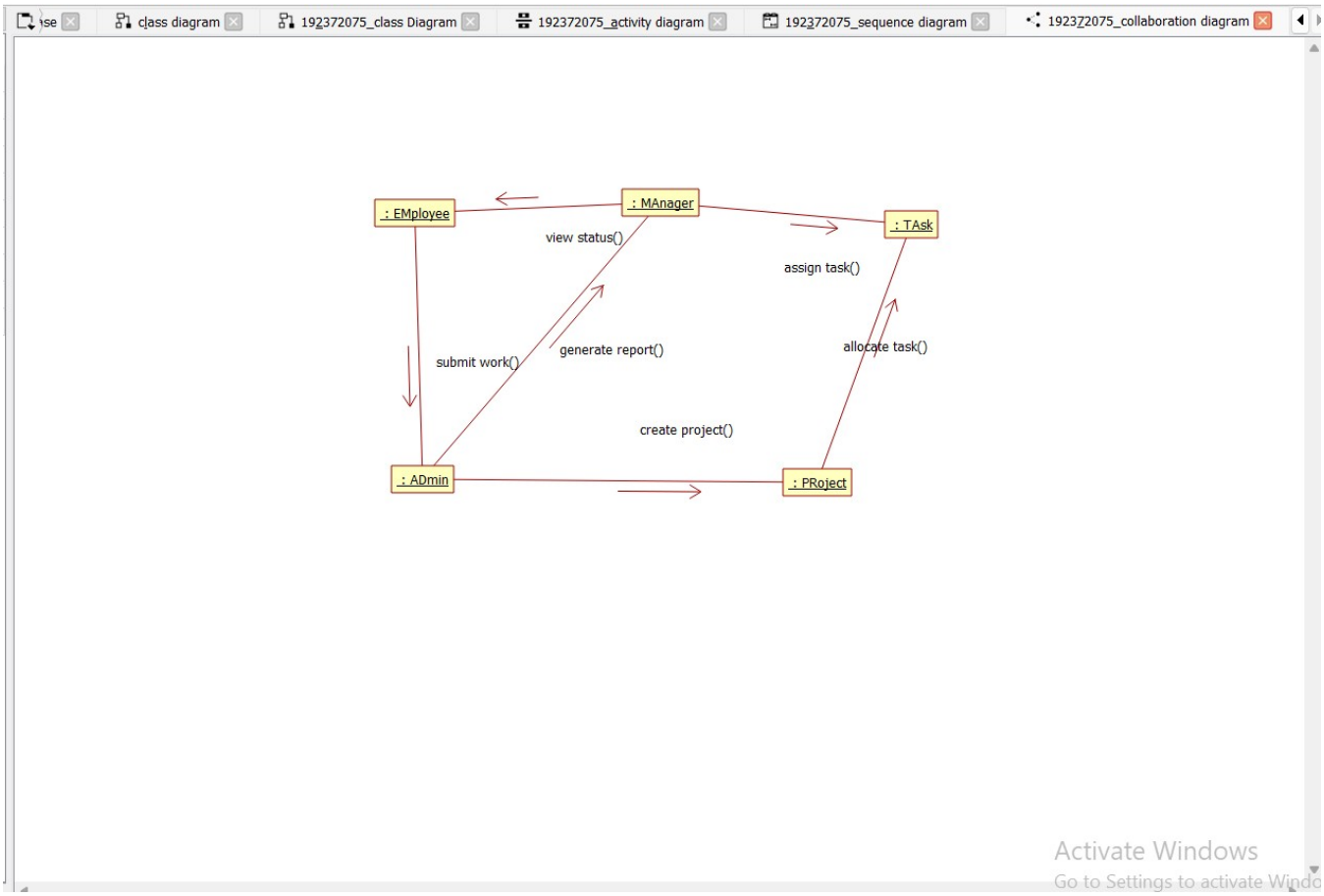
This diagram consists of the objects, messages and return messages

Object: Customer, Dealer, Central System



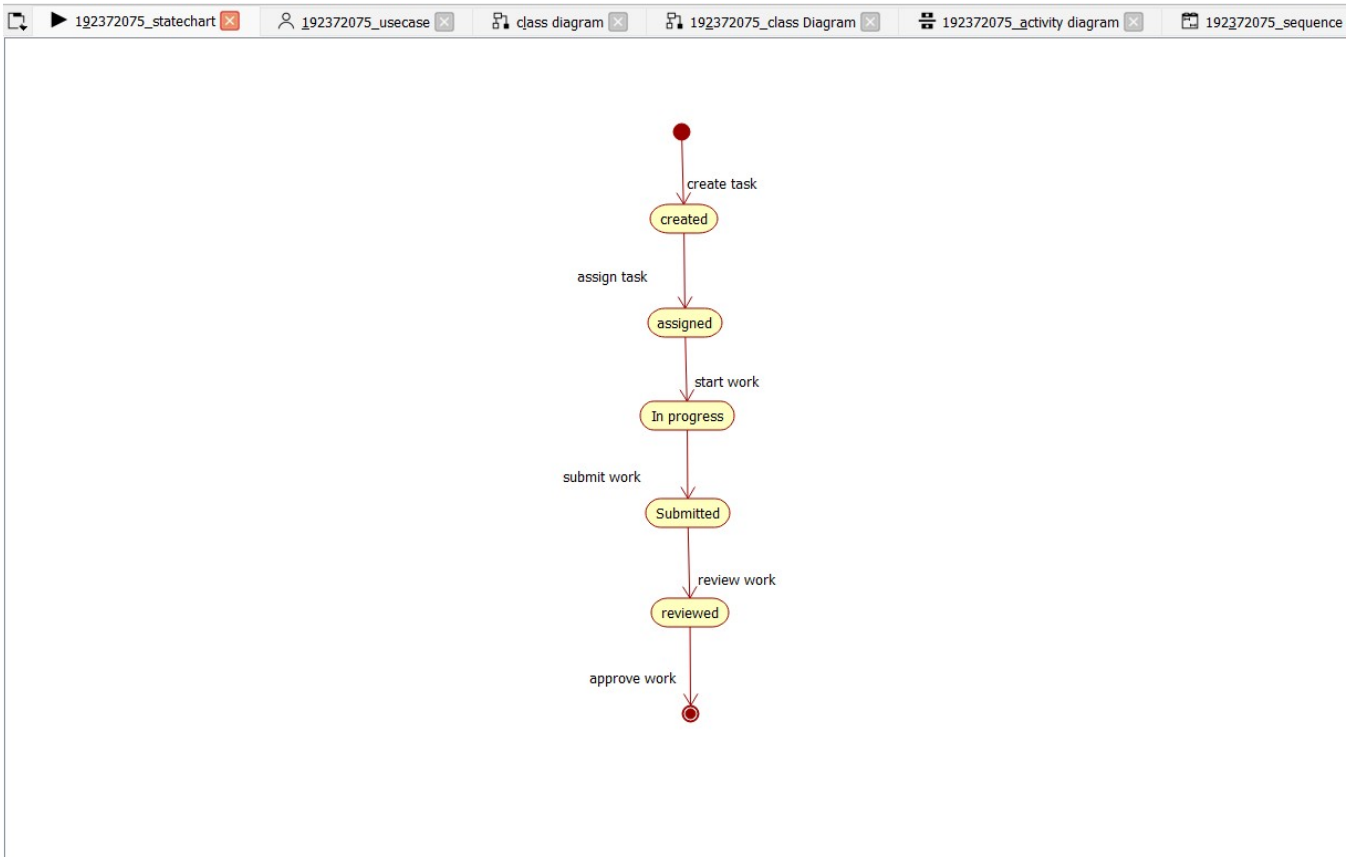
COLLABORATION DIAGRAM:

This diagram contains the objects and actors. This will be obtained by the completion of the sequence diagram and pressing F5 key



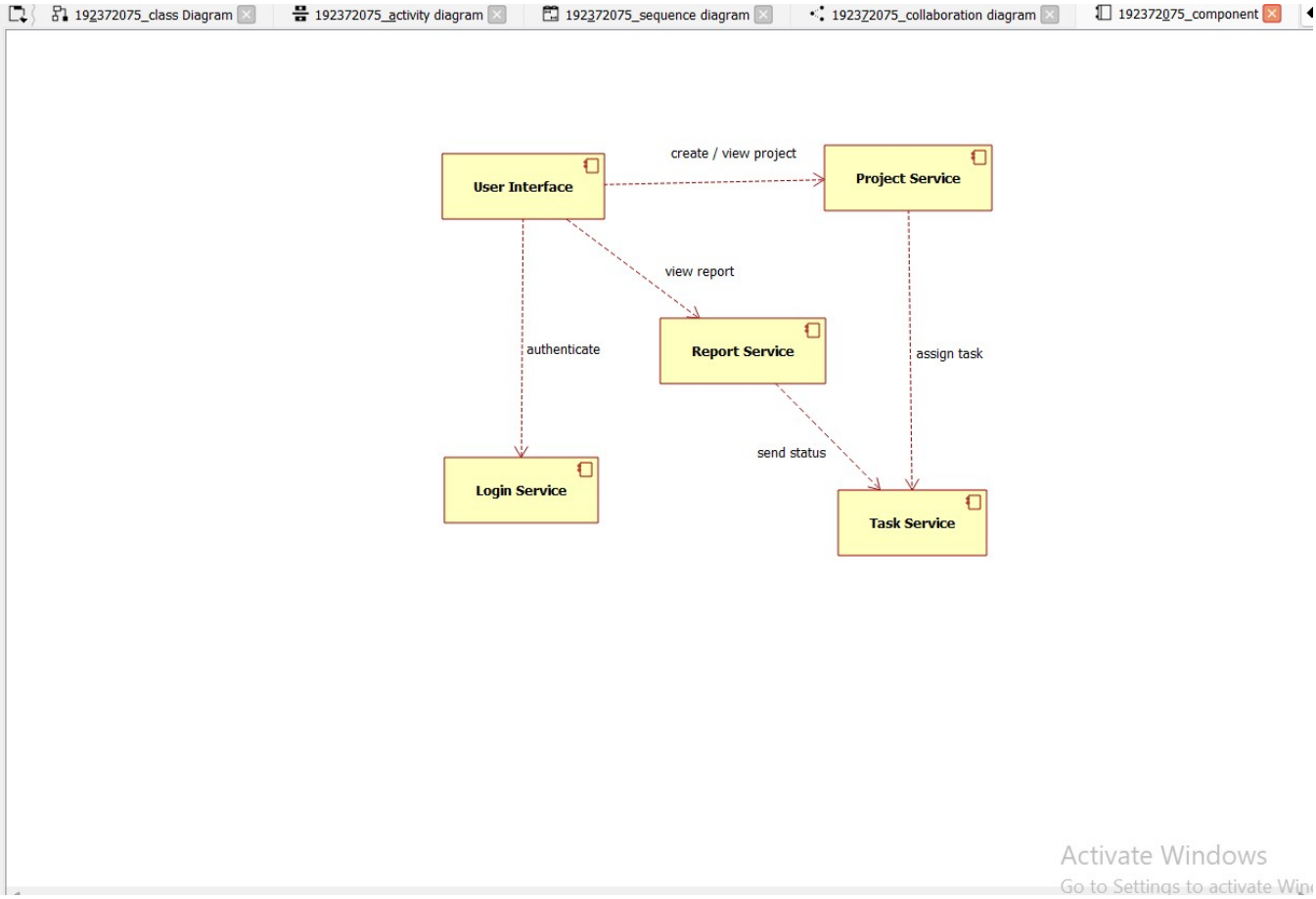
STATECHART DIAGRAM:

It is a technique to describe the behavior of the system. It describes all the possible states that a particular object gets into the object oriented technique. State diagram are drawn for a single class to show to the lifetime behaviour of a single objects



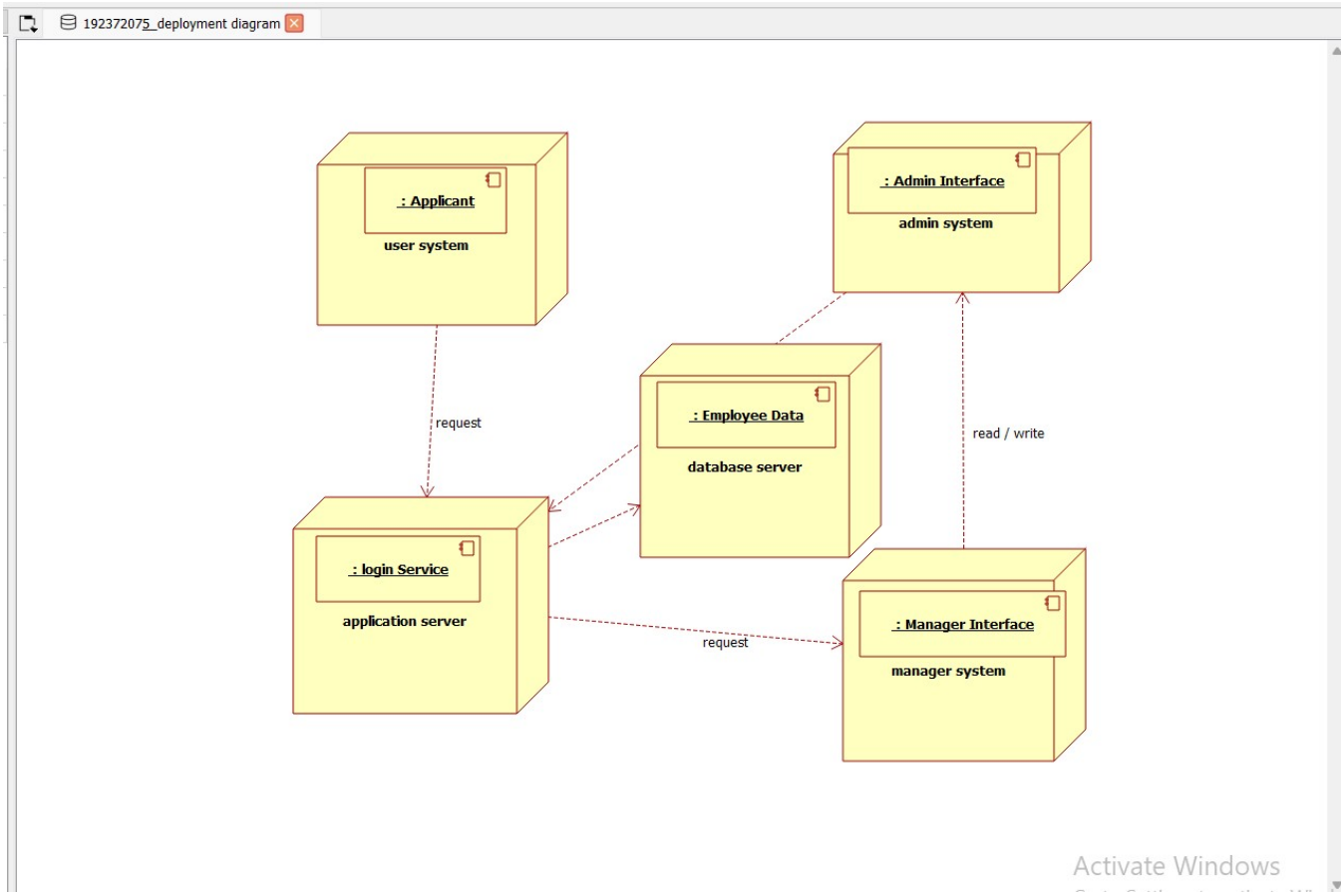
COMPONENT DIAGRAM:

The component diagram is represented by figure dependency and it is a graph of design of figure dependency. The component diagram's main purpose is to show the structural relationships between the components of a systems. It is represented by boxed figure. Dependencies are represented by communication association.



DEPLOYMENT DIAGRAM:

A deployment diagram in the unified modeling language serves to model the physical deployment of artifacts on deployment targets. Deployment diagrams show "the allocation of artifacts to nodes according to the Deployments defined between them. It is represented by 3-dimentional box. Dependencies are represented by communication associatio



OOAD LAB

REGISTER NO:

PACKAGE DIAGRAM:

A package diagram in unified modeling language that depicts the dependencies between the packages that make up a model. A Package Diagram (PD) shows a grouping of elements in the OO model, and is a Cradle extension to UML. PDs can be used to show groups of classes in Class Diagrams (CDs), groups of components or processes in Component Diagrams (CPDs), or groups of processors in Deployment Diagrams (DPDs).

There are three types of layer. They are

- o User interface layer
- o Domain layer
- o Technical services layer

PROGRAM CODING:

CENTRAL SYSTEM:

```
import java.util.Vector;
```

```
public class central system
 {
        public Integer store;
         public Integer update;
         public Vector mydealer;
         public void updating()
         {
         }
        public void processing()
          {
         }
}
```


CUSTOMER:

```
import java.util.Vector;
public class customer
 {
        public Integer name;
        private Integer product;
        public Vector mydealer;
        public void feedback()
        {
         }
         public void customer()
        {
         }
}
```

DEALER:

```
import java.util.Vector;
public class dealer
 {
        public Integer employeename;
        public Integer availability;
        public Integer newAttr;
         public Vector mycustomer;
         public Vector mycentral system;
        public void payment()
         {
        }
         public void delivery()
```