BUS RESERVATION SYSTEM

PROJECT REPORT

18CSC202J/ 18AIC203J - OBJECT ORIENTED DESIGN AND PROGRAMMING LABORATORY

(2018 Regulation)

II Year/ III Semester

Academic Year: 2022 -2023

By

SUBHAM JHA(RA2111026010420)

PRIYANSHU RAJ (RA2111026010436)

Under the guidance of

Dr. Om Prakash P G

Assistant Professor

Department of Computational Intelligence

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur

NOVEMBER 2022

BONAFIDE

This is to certify that **18CSC202J - OBJECT ORIENTED DESIGN AND PROGRAMMING LABORATORY project report** titled "**BUS RESERVATION SYSTEM"** is the bonafide work of **KALVA BHAGEERATH (RA2111026010072) CHEEDELLA S V ABHINAVA SAI (RA2111026010127)**who undertook the task of completing the project within the allotted time.

Signature of the Guide	**Signature of the II Year Academic Advisor**
Dr. S. Amudha	------------------------
Assistant Professor	**Professor and Head**
Department of CINTEL,	Department of CINTEL
SRM Institute of Science and Technology	SRM Institute of Science and Technology

About the course: -

18CSC202J/ 8AIC203J - Object Oriented Design and Programming are 4 credit courses with **L T P C as 3-0-2-4** (Tutorial modified as Practical from 2018 Curriculum onwards)

Objectives:

The student should be made to:

- Learn the basics of OOP concepts in C++
- Learn the basics of OOP analysis and design skills.
- Be exposed to the UML design diagrams.
- Be familiar with the various testing techniques

Course Learning Rationale (CLR): The purpose of learning this course is to:

1. Utilize class and build domain model for real-time programs
2. Utilize method overloading and operator overloading for real-time application

development programs

3. Utilize inline, friend and virtual functions and create application development programs
4. Utilize exceptional handling and collections for real-time object-oriented programming applications
5. Construct UML component diagram and deployment diagram for design of applications
6. Create programs using object-oriented approach and design methodologies for real-time application development

Course Learning Outcomes (CLO): At the end of this course, learners will be able to:

1. Identify the class and build domain model
2. Construct programs using method overloading and operator overloading
3. Create programs using inline, friend and virtual functions, construct programs using standard templates
4. Construct programs using exceptional handling and collections
5. Create UML component diagram and deployment diagram
6. Create programs using object-oriented approach and design methodologies

Table 1: Rubrics for Laboratory Exercises

(Internal Mark Split-up :- As per Curriculum)

CLAP-1	5=(2(E-lab Completion) + 2(Simple Exercises)(from CodeZinger, and any other coding platform) + 1(HackerRank/Code chef/LeetCode Weekend Challenge)	Elab test
CLAP-2	7.5=(2.0(E-lab Completion)+ 2.0 (Simple Exercises)(from CodeZinger, and any other coding platform) + 3.5 (HackerRank/Code chef/LeetCode Weekend Challenge)	Elab test
CLAP-3	7.5=(2.0(E-lab Completion(80 Pgms)+ 2.0 (Simple Exercises)(from CodeZinger, and any other coding platform) + 3.5 (HackerRank/Code chef/LeetCode Weekend Challenge)	**2 Mark -** E-lab Completion **80 Program** Completion from 10 Session (Each session min 8 program) **2 Mark -** Code to UML conversion GCR Exercises **3.5 Mark - Hacker Rank** Coding challenge completion
CLAP-4	5= 3 (Model Practical) + 2(Oral Viva)	• **3 Mark** – Model Test • **2 Mark** – Oral Viva
Total	25	

COURSE ASSESSMENT PLAN FOR OODP LAB

S.No	List of Experiments	Course Learning Outcomes (CLO)	Blooms Level	PI	No of Programs in each session
1.	Implementation of I/O Operations in C++	CLO-1	Understand	2.8.1	10
2.	Implementation of Classes and Objects in C++	CLO-1	Apply	2.6.1	10
3,	To develop a problem statement. 1. From the problem statement, Identify Use Cases and develop the Use Case model. 2. From the problem statement, Identify the conceptual classes and develop a domain model with a UML Class diagram.	CLO-1	Analysis	4.6.1	Mini Project Given
4.	Implementation of Constructor Overloading and Method Overloading in C++	CLO-2	Apply	2.6.1	10
5.	Implementation of Operator Overloading in C++	CLO-2	Apply	2.6.1	10
6.	Using the identified scenarios, find the interaction between objects and represent them using UML Sequence diagrams and Collaboration diagrams	CLO-2	Analysis	4.6.1	Mini Project Given
7.	Implementation of Inheritance concepts in C++	CLO-3	Apply	2.6.1	10
8.	Implementation of Virtual function & interface concepts in C++	CLO-3	Apply	2.6.1	10
9.	Using the identified scenarios in your project, draw relevant state charts and activity diagrams.	CLO-3	Analysis	4.6.1	Mini Project Given
10.	Implementation of Templates in C++	CLO-3	Apply	2.6.1	10
11.	Implementation of Exception of Handling in C++	CLO-4	Apply	2.6.1	10
12.	Identify the User Interface, Domain objects, and Technical Services. Draw the partial layered, logical architecture diagram with UML package diagram notation such as Component Diagram, Deployment Diagram.	CLO-5	Analysis	4.6.1	Mini Project Given
13.	Implementation of STL Containers in C++	CLO-6	Apply	2.6.1	10
14.	Implementation of STL associate containers and algorithms in C++	CLO-6	Apply	2.6.1	10
15.	Implementation of Streams and File	CLO-6	Apply	2.6.1	10

	Handling in C++				

LIST OF EXPERIMNENTS FOR UML DESIGN AND MODELLING:

To develop a mini-project by following the exercises listed below.

1. To develop a problem statement.
2. Identify Use Cases and develop the Use Case model.
3. Identify the conceptual classes and develop a domain model with UML Class diagram.
4. Using the identified scenarios, find the interaction between objects and represent them using UML Sequence diagrams.
5. Draw relevant state charts and activity diagrams.
6. Identify the User Interface, Domain objects, and Technical services. Draw the partial layered, logical architecture diagram with UML package diagram notation.

Suggested Software Tools for UML:

StarUML, Rational Suite, Argo UML (or) equivalent, Eclipse IDE and Junit

ABSTRACT

Bus reservation system is a simple project showing the implementation of class along with the object of C++ language. Any organization that runs the buses can use this project to give the users the chance to book the tickets for their travel. Here, the user can perform tasks like login, reserve bus seat, show reservation information and show information regarding the buses available. And an admin can install the bus information for users to select the bus of required location and timings. In this we goto login page, where the user should select the user login or abstract login. Then enter their credentials to login. If the user has no account the one should create an account by selecting the create account option. Admin can install the busses by entering the bus details. And also can view the list of busses with its details. The user can also view the list of the busses and details. User can book the ticket by selecting the bus and no of passengers and give their details to book the ticket. The user can also view the ticket after booking.

MODULE DESCRIPTION

Login Page:

In this the user should select an option between admin login or user login. If the person is Admin, one should select admin login else the person should select the User login. In admin login one can login by giving the credentials. And then one can find 3 options to install busses, display busses and logout. For user login too the one can enter by credentials or can create an account if one doesn't have it. Then by logging in one can select options like book ticket, view ticket, view busses and logout.

Install Busses:

Only Admin can access this. Admin can upload the bus details like:
Bus name
Bus number
Start and destination
Timings
Number of seats available
Cost of each seat
These are to give a list to users to select the bus that they need.

Display Busses:

This allows both the admin and user both to see the list of the bus with their details like available seats and cost. This helps the user to select the bus and admin to manage the bus.

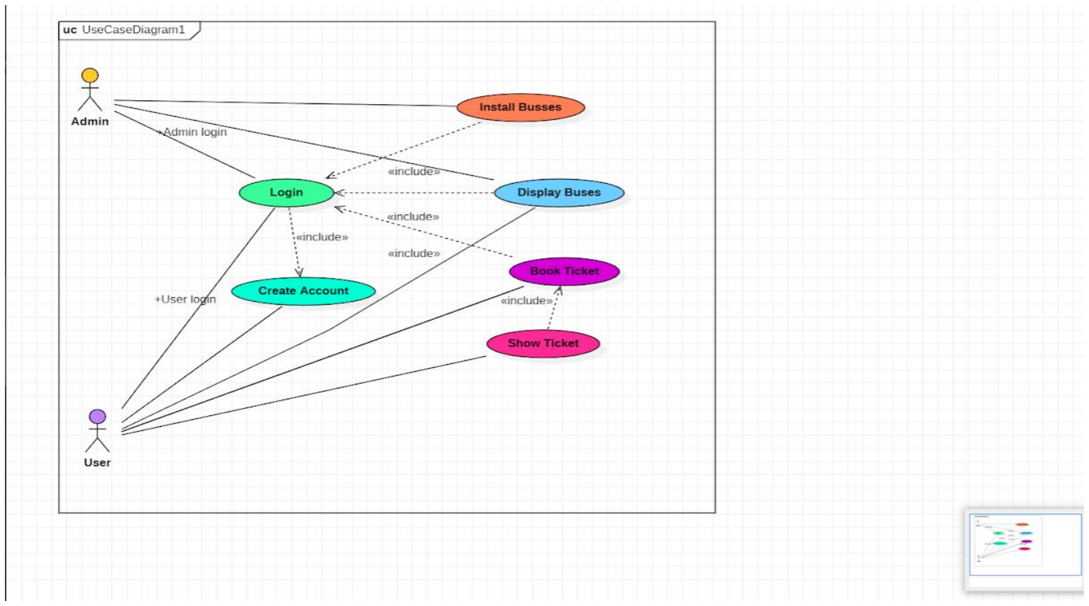
Book Ticket:

This is accessed by the user. In this user can book the ticket by selecting the desired bus and by giving the number of passengers. And the user gives passenger details for booking the ticket.

View Ticket:

This allows user to see their travel details. Here user can see the bus details and passenger details. This also shows the total amount of the ticket.

Use case diagram with explanation

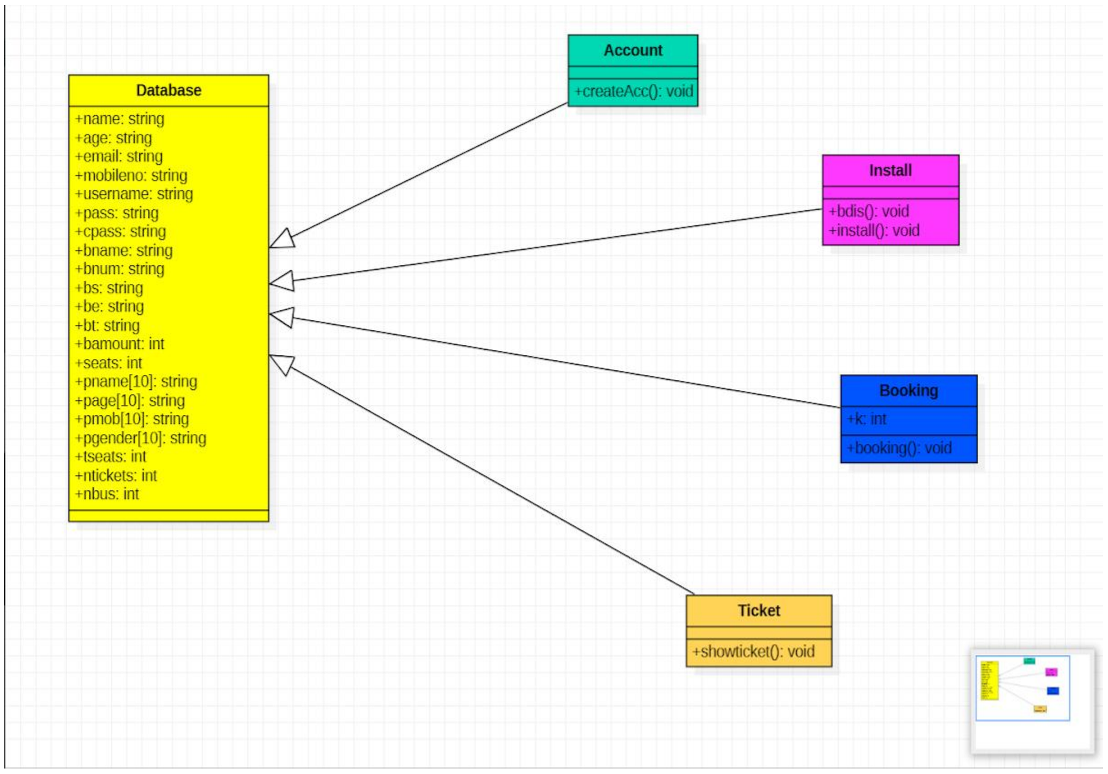


- Use cases: Horizontally shaped ovals that represent the different uses that a user might have.
- Actors: Stick figures that represent the people actually employing the use cases.
- Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.

System boundary boxes: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.

- Packages: A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.
- **Extend relationship:** The use case is optional and comes after the base use case. It is represented by a dashed arrow in the direction of the base use case with the notation <<extend>>.
- **Include relationship:** The use case is mandatory and part of the base use case. It is represented by a dashed arrow in the direction of the included use case with the notation <<include>>.

Class diagram with explanation



- Upper section: Contains the name of the class. This section is always required, whether you are talking about the classifier or an object.
- Middle section: Contains the attributes of the class. Use this section to describe the qualities of the class. This is only required when describing a specific instance of a class.

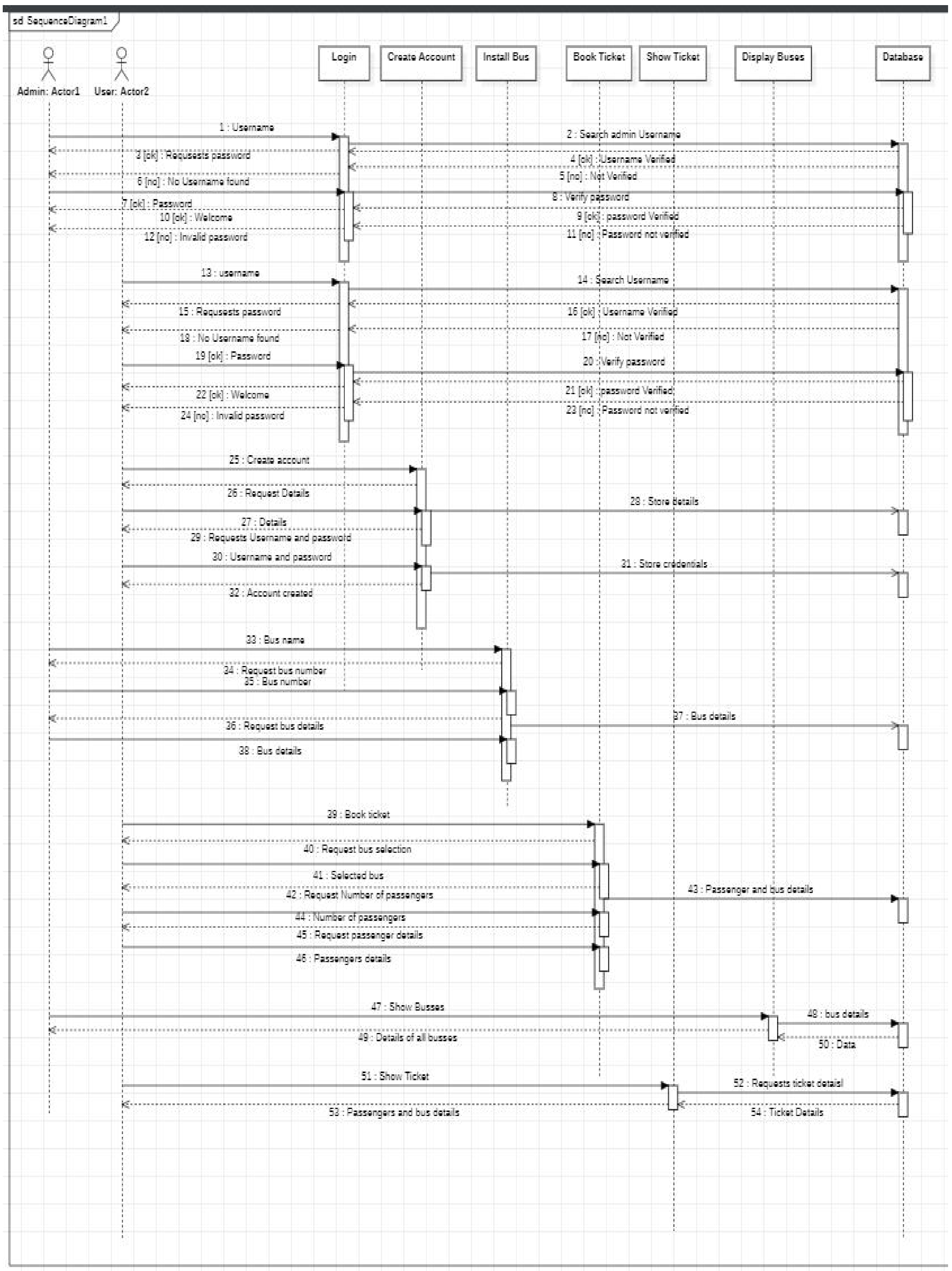
- Bottom section: Includes class operations (methods). Displayed in list format, each operation takes up its own line. The operations describe how a class interacts with data.

All classes have different access levels depending on the access modifier (visibility). Here are the access levels with their corresponding symbols:

- Public (+)
- Private (-)
- Protected (#)
- Package (~)
- Derived (/)
- Static (underlined)

Sequence diagram with

explanation



Actors – An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope

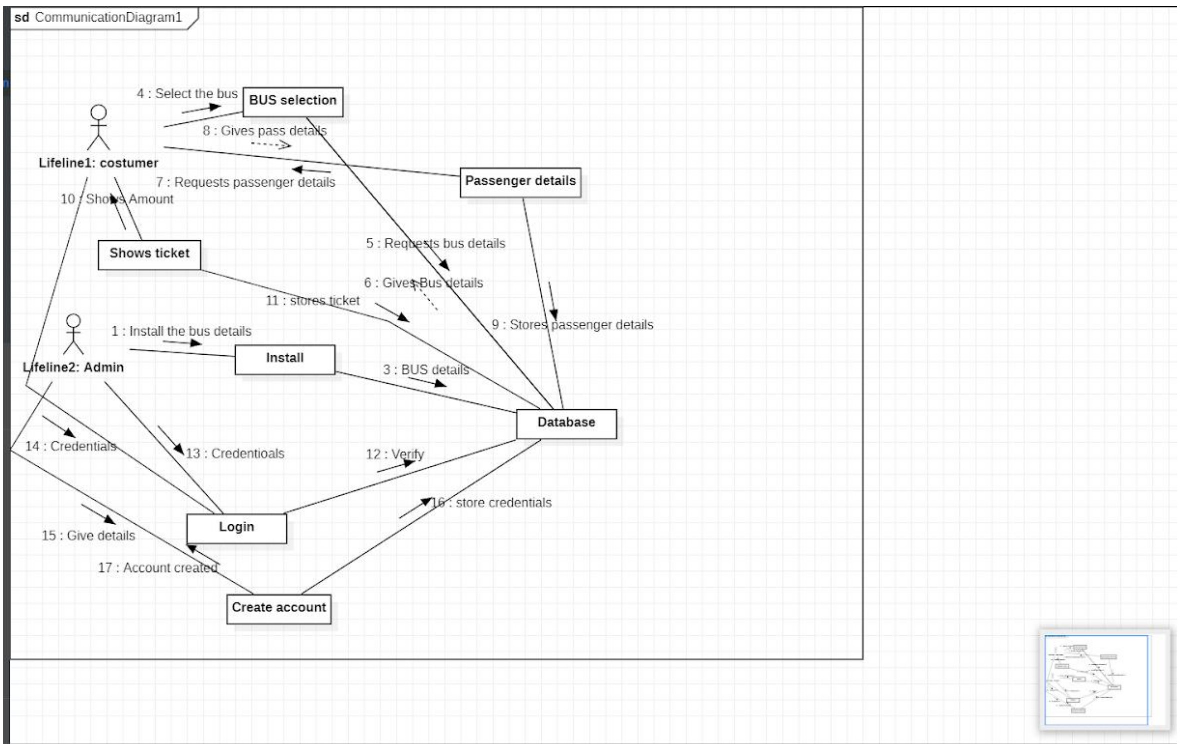
Lifelines – A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram. The standard in UML for naming a lifeline follows the following format – Instance Name : Class Name

Messages – Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram.

Synchronous messages – A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message. A large number of calls in object oriented programming are synchronous. We use a solid arrow head to represent a synchronous message.

Asynchronous Messages – An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not. We use a lined arrow head to represent an asynchronous

Communication diagram with explanation



Objects: The representation of an object is done by an object symbol with its

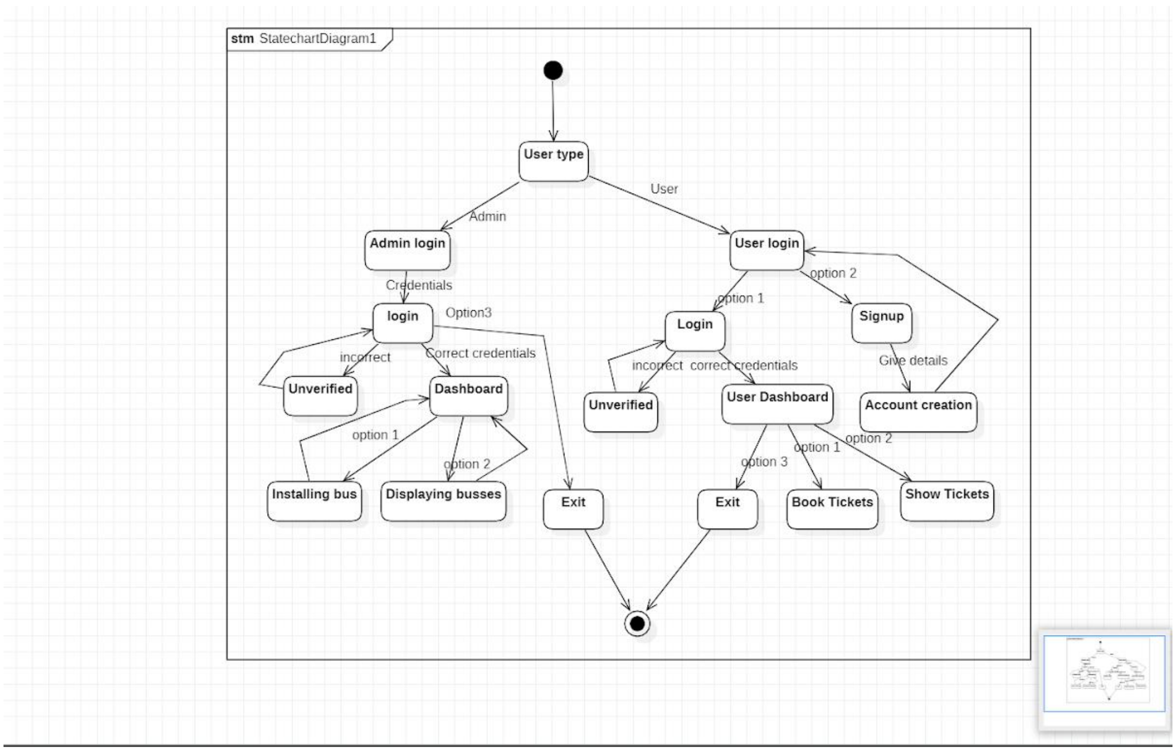
name and class underlined, separated by a colon.

Actors: In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.

Links: The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.

Messages: It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

State chart diagram with explanation



Initial state – We use a black filled circle represent the initial state of a System or a class.

Transition – We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

Transition – We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

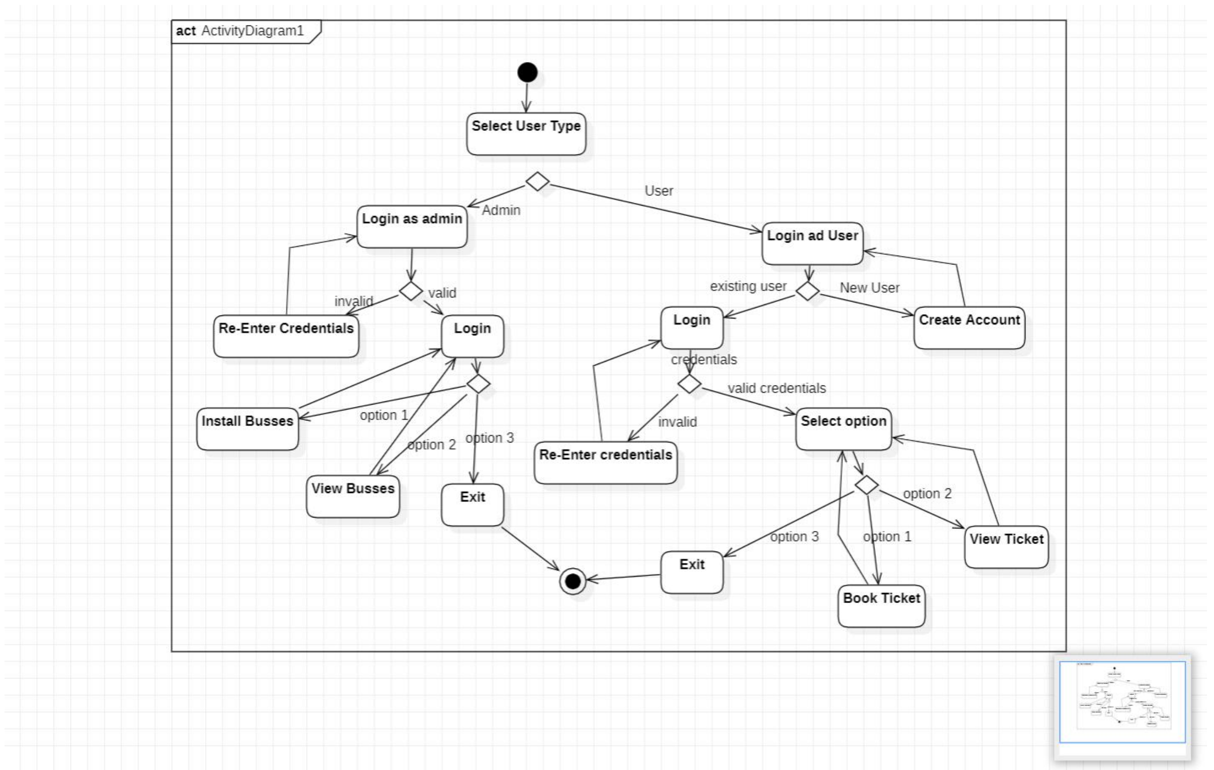
Fork – We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states.

Join – We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.

Self transition – We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases.

Final state – We use a filled circle within a circle notation to represent the final state in a state machine diagram.

Activity diagram with explanation



Initial State – The starting state before an activity takes place is depicted using the initial state. ● **Figure –** notation for initial state or start state A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated.

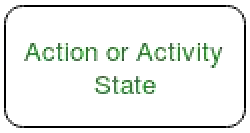
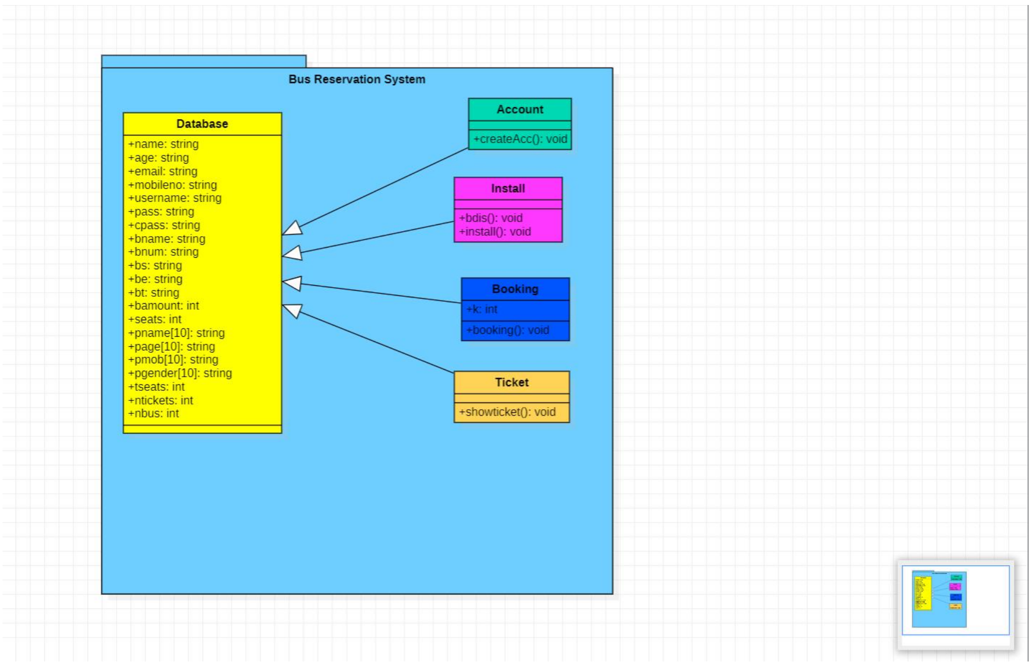
Action or Activity State – An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity. 

Figure – notation for an activity state For example – Consider the previous example of opening an application opening the application is an activity state in the activity diagram.

Action Flow or Control flows – Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another. **Figure –** notation for control Flow An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow.

Decision node and Branching – When we need to make a decision before deciding the flow

Package diagram with explanation



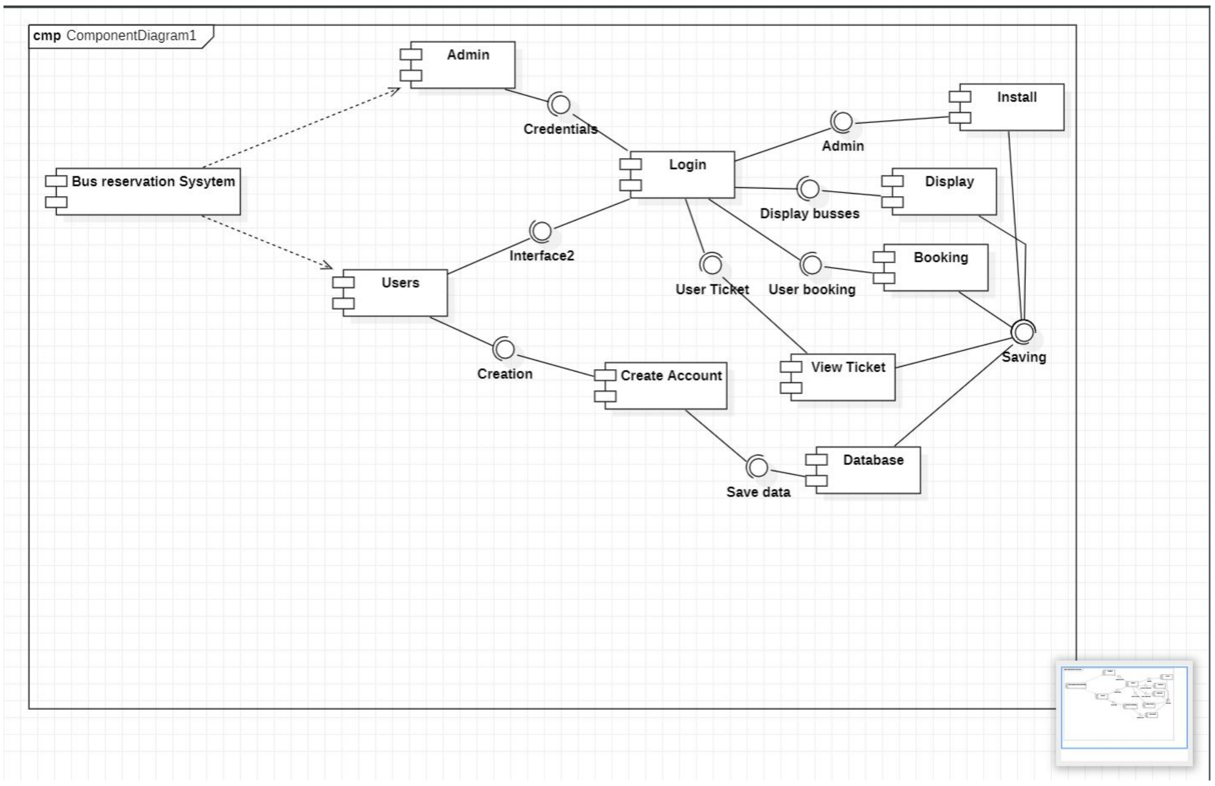
- Package: A namespace used to group together logically related elements within a system. Each element contained within the package should be a packageable element and have a unique name.
- Packageable element: A named element, possibly owned directly by a package. These can include events, components, use cases, and

packages themselves. Packageable elements can also be rendered as a rectangle within a package, labeled with the appropriate name.

- Dependencies: A visual representation of how one element (or set of elements) depends on or influences another. Dependencies are divided into two groups: access and import dependencies. (See next section for more info.)
- Element import: A directed relationship between an importing namespace and an imported packageable element. This is used to import select individual elements without resorting to a package import and without making it public within the namespace.
- Package import: A directed relationship between and importing namespace and an imported package. This type of directed relationship adds the names of the members of the imported package to its own namespace
- Package merge: A directed relationship in which the contents of one package are extended by the contents of another. Essentially,

the content of two packages are combined to produce a new package.

Component diagram with explanation



Component

A component is a logical unit block of the system, a slightly higher abstraction than classes. It is represented as a rectangle with a smaller rectangle in the upper right corner with tabs or the word written above the name of the component to help distinguish it from a class.

Interface

An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.

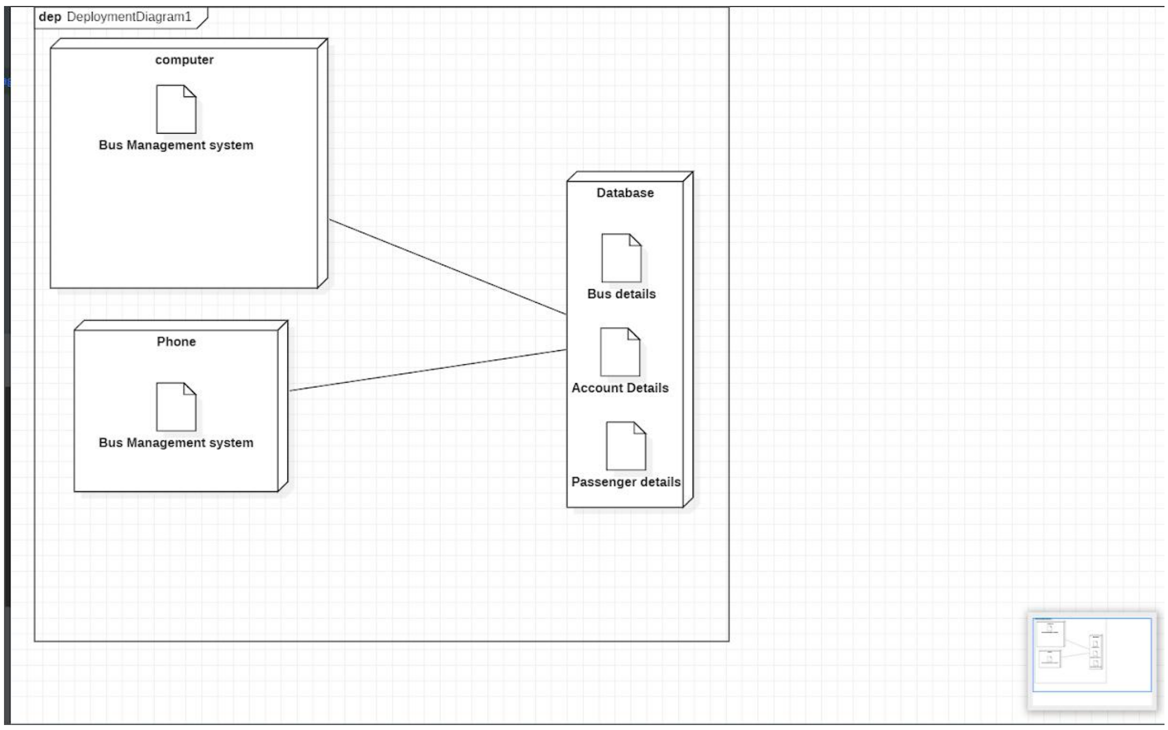
Dependencies

Draw dependencies among components using dashed arrows.

Port

Ports are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.

Deployment diagram with explanation



An artifact represents the specification of a concrete real-world entity related to software development. You can use the artifact to describe a framework which is used during the software development process or an executable file. Artifacts are deployed on the nodes. Node is a computational resource upon which artifacts are deployed for execution. A node is a physical thing that can execute one or more artifacts. A node may vary in its size depending upon the size of the project.

Node is an essential UML element that describes the execution of code and the communication between various entities of a system. It is denoted by a 3D box with the node-name written inside of it. Nodes help to convey the hardware which is used to deploy the software.

Deployment diagrams are mostly used by system administrators, network engineers, etc. These diagrams are used with the sole purpose of describing how software is deployed into the hardware system. It visualizes how software interacts with the hardware to execute the complete functionality.

Code For Bus Reservation System:

```
#include <iostream>
#include<stdlib.h>
#include<dos.h>
#include<conio.h>
using namespace std;
int i=0,b=0;
class Database
{
   public:
   string name,age,email,mobileno,username,pass,cpass;
   string bname,bnum,bs,be,bt;
   int bamount,seats;
   string pname[10],page[10],pmob[10],pgender[10];
   int tseats,ntickets=0,nbus;

};

class Account:public Database
{
   public:
   void createAcc()
   {
      cout<<"Enter your Name : ";
      cin>>name;
      cout<<"Enter your Age  : ";
      cin>>age;
```

```
        cout<<"Enter your E-mail : ";
        cin>>email;
        cout<<"Enter your Mobile number : ";
        cin>>mobileno;
        cout<<"Create your Username : ";
        cin>>username;
        loop:
        cout<<"Create your Password : ";
        cin>>pass;
        cout<<"Confirm your Password : ";
        cin>>cpass;
        if(pass==cpass)
            cout<<"...Account Created...";
        else
            goto loop;
            i++;
    }
}A[100];

class Install:public Database
{
    public:
    void bdis()
    {
            cout<<"Bus Name : "<<bname<<endl;
            cout<<"Bus Number : "<<bnum<<endl;
            cout<<"Bus Starting Point : "<<bs<<endl;
            cout<<"Bus Ending Point : "<<be<<endl;
            cout<<"Bus Timing at Starting Position : "<<bt<<endl;
            cout<<"Number of seats Available in Bus : "<<seats<<endl;
            cout<<"Cost of each Seat : "<<bamount<<endl<<endl<<endl;

    }

    void install()
    {
        cout<<"Bus Name : ";
        cin>>bname;
        cout<<"Bus Number : ";
        cin>>bnum;
        cout<<"Bus Starting Point : ";
        cin>>bs;
        cout<<"Bus Ending Point : ";
        cin>>be;
        cout<<"Bus Timing at Starting Position : ";
        cin>>bt;
        cout<<"Total number of seats in Bus : ";
        cin>>seats;
        cout<<"Cost of each Seat : ";
        cin>>bamount;
        b++;
    }

}l[100];

class Booking: public Database
```

```
{
   public:
      int k;
   void booking()
   {
      for(int c=1,k=0;k<b;k++,c++)
                  {
                   cout<<c<<endl;
                   l[k].bdis();
                   cout<<endl<<endl;
                  }
      cout<<"Select a Bus : ";
      cin>>nbus;
      cout<<"Enter number of Passengers : ";
      cin>>tseats;
      for(k=0;k<tseats;k++)
      {
         cout<<"Enter Name of Passenger "<<k+1<<" : ";
         cin>>pname[k];
         cout<<"Enter Age of  "<<pname[k]<<" : ";
         cin>>page[k];
         cout<<"Enter Gender of  "<<pname[k]<<" : ";
         cin>>pgender[k];
         cout<<"Enter Mobile number of  "<<pname[k]<<" : ";
         cin>>pmob[k];
         cout<<endl<<endl;
      }
      l[nbus-1].seats=l[nbus-1].seats-tseats;
```

```
        cout<<".....Ticket is booked....."<<endl;
    }
}B[100];

class Ticket:public Database
{
    public:
    void showticket(int m,int s)
    {
        int n;
        cout<<"Bus Name : "<<I[m].bname<<endl;
        cout<<"Bus Number : "<<I[m].bnum<<endl;
        cout<<"Bus Starting Point : "<<I[m].bs<<endl;
        cout<<"Bus Ending Point : "<<I[m].be<<endl;
        cout<<"Bus Timing at Starting Position : "<<I[m].bt<<endl;
        cout<<"No of seats booked : "<<B[m].tseats<<endl;
        cout<<"Total Amount : "<<I[m].bamount*B[m].tseats<<endl<<endl;
        cout<<"Details of Passengers :"<<endl;
        for(n=0;n<B[m].k;n++)
        {
            cout<<"Name of the Passenger "<<n+1<<" : "<<B[m].pname[n]<<endl;
            cout<<"Age of "<<B[m].pname[n]<<" : "<<B[m].age[n]<<endl;
            cout<<"Gender of "<<B[m].pname[n]<<" : "<<B[m].pgender[n]<<endl;
            cout<<"Mobile number of "<<B[m].pname[n]<<" :
"<<B[m].pmob[n]<<endl<<endl;
        }
    }
}T;
```

```
int main()
{
   int ch,k;
   string aname,apass;
   mloop:
   cout<<"Welcome to SRM BUS Portal"<<endl;
   cout<<"1.      Admin      Portal"<<endl<<"2.      User      Portal"<<endl<<"3.
Exit"<<endl<<"Enter Your Choice : ";
   cin>>ch;
   system("cls");
   if(ch==1)
   {
      loop:
      cout<<"Enter Admin Username : ";
      cin>>aname;
      if(aname=="srmadmin")
      {
         loop1:
         cout<<"Enter Admin Password : ";
         cin>>apass;
         if(apass=="Admin22")
         {
            cout<<"Welcome to Admin Portal"<<endl;
            aloop:
            cout<<"1.      Display      "<<endl<<"2.      Install"<<endl<<"3.
Logout"<<endl<<"Enter your choice : ";
            cin>>ch;
```

```
          system("cls");
          if(ch==1)
             {
                if(b==0){
                   cout<<"No Busses are Installed!"<<endl;delay(1000);}
                else{
                for(int j=0;j<b;j++)
                   I[j].bdis();}
                goto aloop;
             }
          if(ch==2)
          {
             I[b].install();
             goto aloop;
          }
          if(ch==3)
             goto mloop;
       }
          else{
             cout<<"Invalid Password"<<endl;goto loop1;}

    }
    else{
       cout<<"Invalid Username"<<endl;
       goto loop;}

 }
 if(ch==2)
```

```
{
    cout<<"1. Login"<<endl<<"2. Sign Up"<<endl<<"Enter your Choice : ";
    cin>>ch;
    if(ch==1)
    {
        lloop:
        cout<<"Enter Username : ";
        cin>>aname;
        for(int j=0;j<i;j++)
        {
            if(A[j].username==aname)
            {
                 ploop:
            cout<<"Enter Password : ";
            cin>>apass;
            if(apass==A[j].pass){
                cout<<endl<<"Welcome "<<A[j].name<<endl;
                tloop:
                cout<<"1.  Book  Tickets"<<endl<<"2.  Show  my  Ticket"<<endl<<"3.
Display Buses"<<endl<<"4. Logout"<<endl<<"Enter your Choice : ";
                cin>>ch;
                if(ch==1)
                {
                    if(B[j].ntickets==0){
                    B[j].booking();B[j].ntickets++;}
                    else
                        cout<<"...Max Limit is Exceeded..";
                    goto tloop;
```

```
          }
          if(ch==2)
          {
             if(B[j].ntickets==0){cout<<"No   Tickets   are   Booked"<<endl;goto
tloop;}
             else{
                T.showticket(j,B[j].nbus);
                goto tloop;}
          }
          if(ch==3)
          {
             for(k=0;k<b;k++)
             {
              I[k].bdis();
              cout<<endl<<endl;
             }
             goto tloop;
          }
          if(ch==4)
          {
             goto mloop;
          }
          }
          else
          {
             cout<<"Invalid Password"<<endl;
             goto ploop;
          }
```

```
                k=1;
                break;
            }
            }
            if(k!=1){
                cout<<"NO Account found!";
                goto lloop;}

        }



        if(ch==2)
        {
            A[i].createAcc();
            cout<<endl;
            goto mloop;
        }

    }
    if(ch==3)
    return 0;
}
```

Conclusion

Hence The Above are the representation of the Bus reservation system using UML Diagrams. This gives a clear idea about the working of the bus reservation system in all the processes and the things that can be done by the system. And the above there is the sample code that runs the bus reservation system which is represented by the UML diagrams.

References

Bus Reservation System C++ Project | Code with C

UML Diagrams - Javatpoint

UML - Standard Diagrams (tutorialspoint.com)