Practical 7

Structural View Diagram

Aim: To Design structural view diagram for the selected system using class Diagram, Object Diagram, and Component Diagram.

7.1 Structural View Diagrams

Structure diagrams depict the static structure of the elements in your system. i.e., how one object relates to another. It shows the things in the system – classes, objects, packages or modules, physical nodes, components, and interfaces. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems. The Seven UML structural diagrams are roughly organized around the major groups of things you'll find when modeling a system. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems.

Table 7.1:Structural View Diagram

Structural Diagram	**Brief Description**
Composite Structure Diagram	It shows the internal structure of a classifier, classifier interactions with the environment through ports, or behavior of a collaboration.
Deployment Diagram	It shows a set of nodes and their relationships that illustrates the static deployment view of an architecture.
Package Diagram	It groups related UML elements into a collection of logically related UML structure.
Profile Diagram	
Class Diagram	It shows a set of classes, interfaces, and collaborations and their relationships, typically, found in modeling object-oriented systems.
Object Diagram	It shows a set of objects and their relationships, which is the static snapshots of instances of the things found in class diagrams.
Component Diagram	It shows a set of components and their relationships that illustrates the static implementation view of a system.

7.1.1 Class Diagram

The **UML** Class diagram is a graphical notation used to construct and visualize object oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:

- classes,
- their attributes,
- operations (or methods),
- and the relationships among objects.

Purpose of Class diagram:

- Class diagram is a static diagram.
- It represents the static view of an application.
- Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Notations

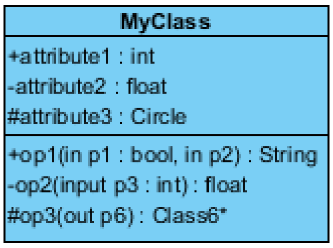
Class Notation

A class notation consists of three parts:

1.Class Name
- The name of the class appears in the first partition.

2.Class Attributes
- Attributes are shown in the second partition.
- The attribute type is shown after the colon.
- Attributes map onto member variables (data members) in code.

3.Class Operations (Methods)
- Operations are shown in the third partition. They are services the class provides.
- The return type of a method is shown after the colon at the end of the method signature.
- The return type of method parameters is shown after the colon following the parameter name.
- Operations map onto class methods in code
-



The graphical representation of the class - MyClass as shown above:

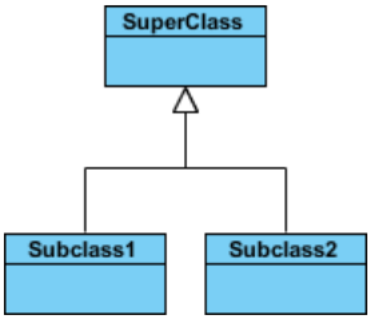
- MyClass has 3 attributes and 3 operations
- Parameter p3 of op2 is of type int
- op2 returns a float
- op3 returns a pointer (denoted by a *) to Class6

4.Class Relationships

A class may be involved in one or more relationships with other classes. A relationship can be one of the following types: (Refer to the figure on the right for the graphical representation of relationships).

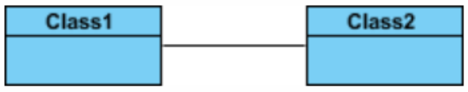
Inheritance (or Generalization):

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class



Simple Association:

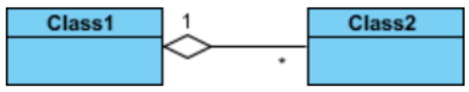
- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes



Aggregation:

A special type of association. It represents a "part of" relationship.

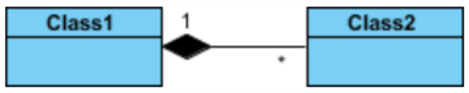
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite



Composition:

A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.
- A solid line with a filled diamond at the association connected to the class of composite.



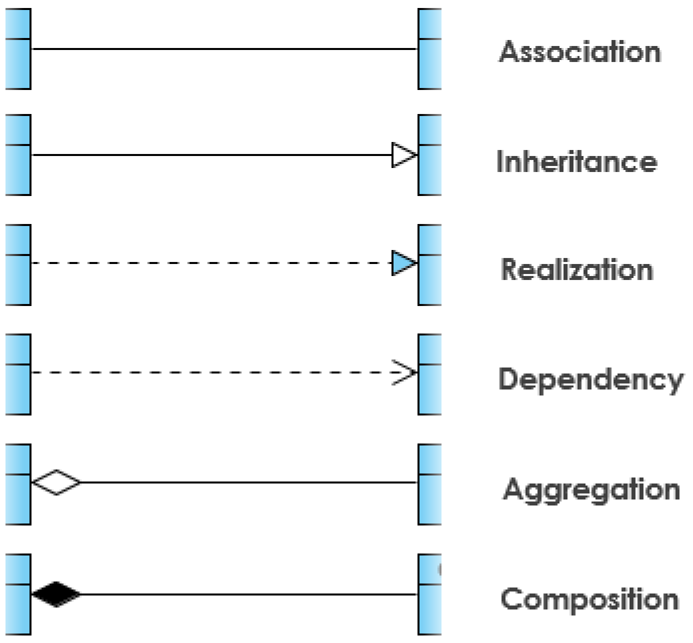
Dependency:

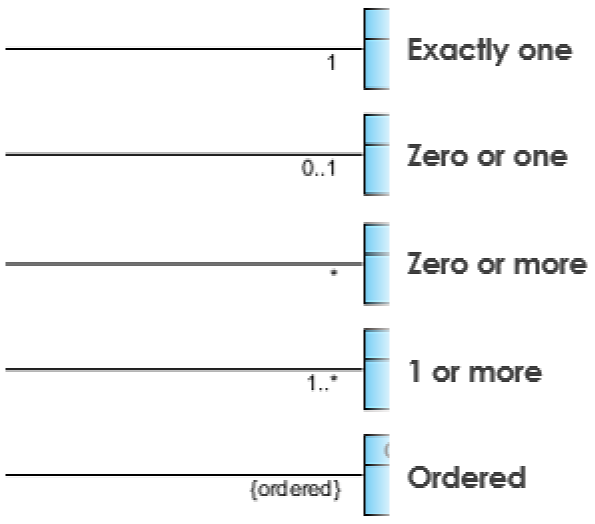
- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow

Multiplicity

How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*


Association
Inheritance
Realization
Dependency
Aggregation
Composition


1
Exactly one
0..1
Zero or one
*
Zero or more
1..*
1 or more
{ordered}
Ordered

7.1.2 Class Diagram for Bus Management system

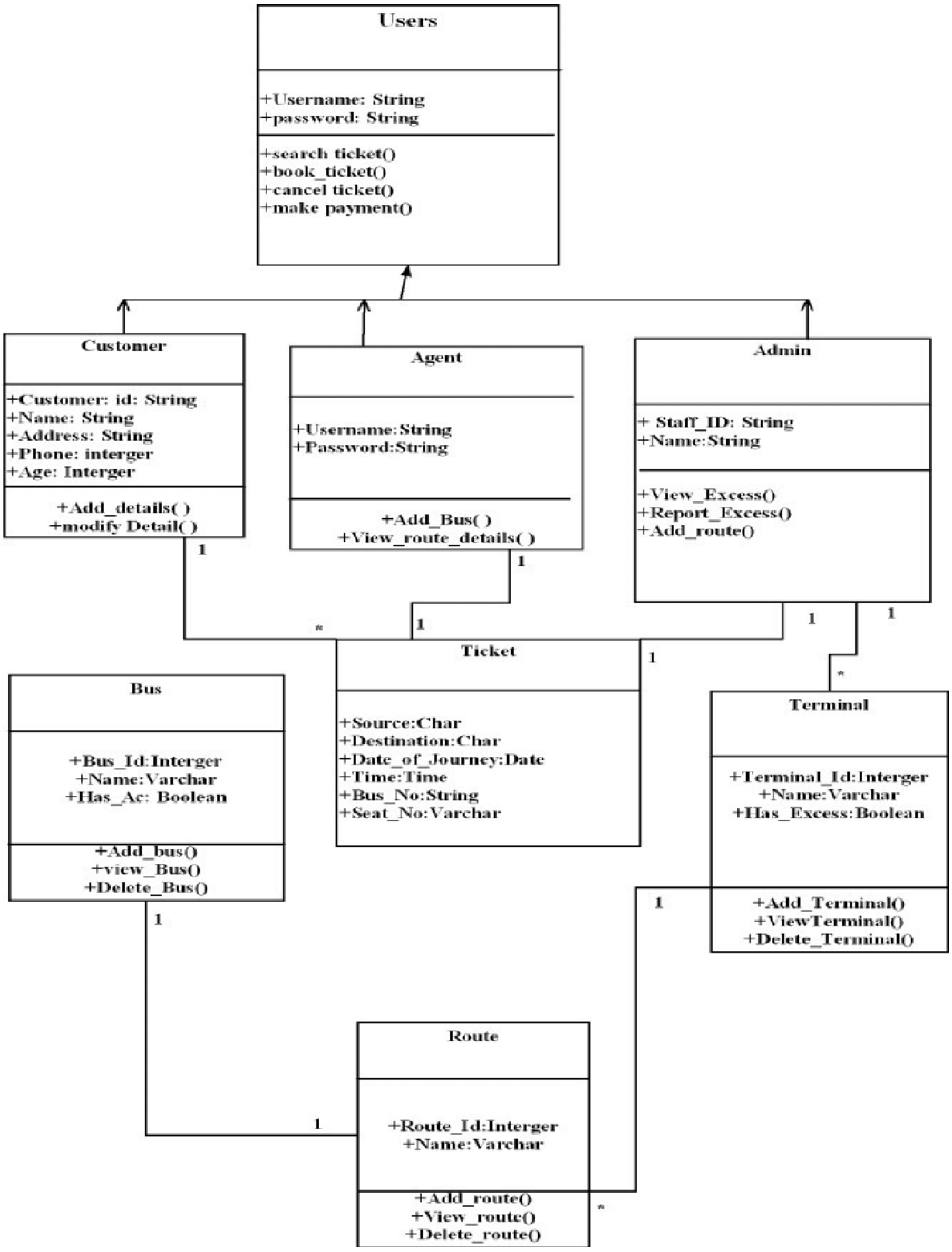


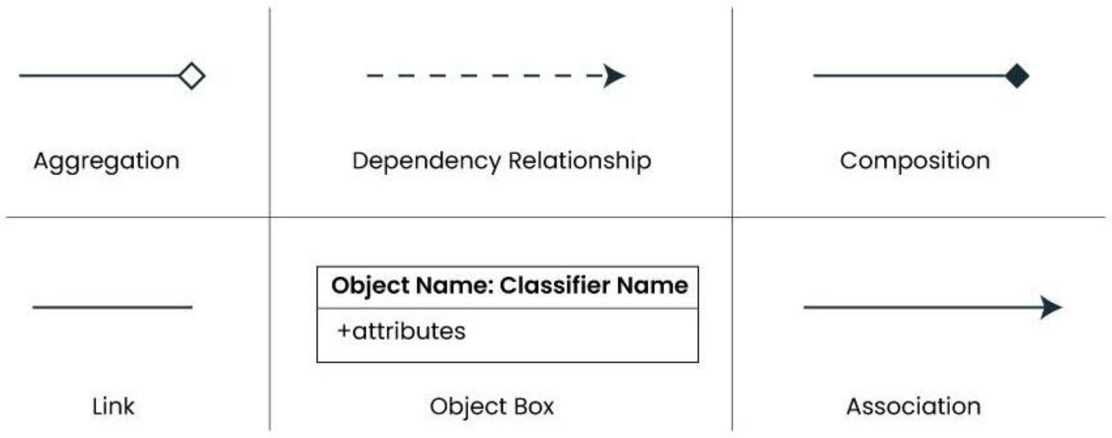
Figure 7.1 class diagram

7.1.2 Object Diagram

Object diagrams are a visual representation in UML (Unified Modeling Language) that illustrates the instances of classes and their relationships within a system at a specific point in time. They display objects, their attributes, and the links between them, providing a snapshot of the system's structure during execution.

An object diagram in UML is useful because it provides a clear and visual representation of specific instances of classes and their relationships at a particular point in time, aiding in understanding and communicating the structure and interactions within a system.

Object Diagram Notations
The object diagram in UML uses specific notations to represent instances of classes and their relationships at a particular moment in time.



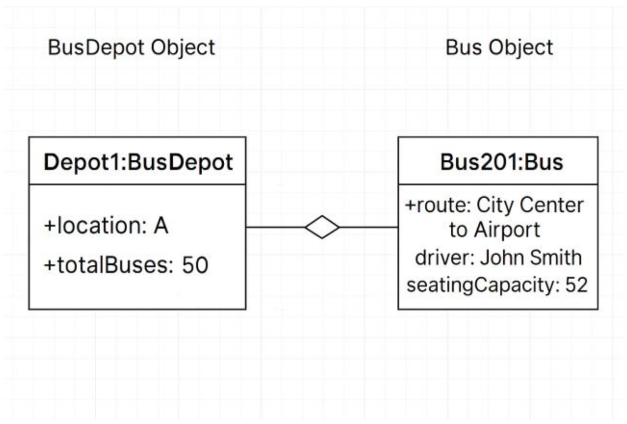
1. Objects or Instance specifications
When we instantiate a classifier in a system, the object we create represents an entity which exists in the system. We can represent the changes in object over time by creating multiple instance specifications. We use a rectangle to represent an object in an object diagram

2.Attributes and Values
Inside the object box, attributes of the object are listed along with their specific values.

7.1.3 Object Diagram for A Bus Management System

(At least include any one scenario)

An Object diagram showing aggregation in Bus system

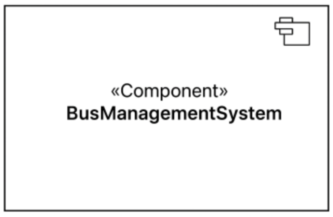


7.1.4 Component Diagram

Component-based diagrams are essential tools in software engineering, providing a visual representation of a system's structure by showcasing its various components and their interactions. These diagrams simplify complex systems, making it easier for developers to design, understand, and communicate the architecture.

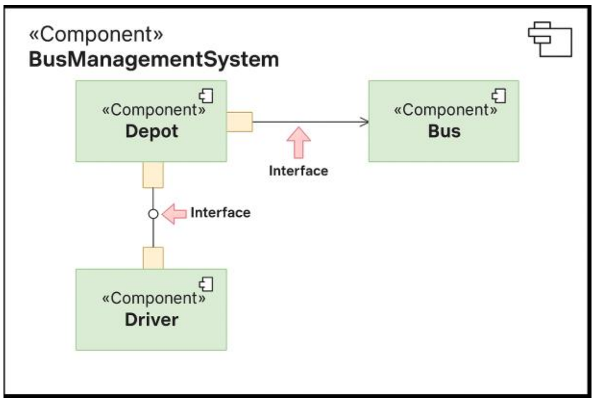
Components of Component-Based Diagram

1. Component
Represent modular parts of the system that encapsulate functionalities. Components can be software classes, collections of classes, or subsystems.
Symbol: Rectangles with the component stereotype («component»).
Function: Define and encapsulate functionality, ensuring modularity and reusability.



2. Interfaces
Specify a set of operations that a component offers or requires, serving as a contract between the component and its environment.
Symbol: Circles (lollipops) for provided interfaces and half-circles (sockets) for required interfaces.

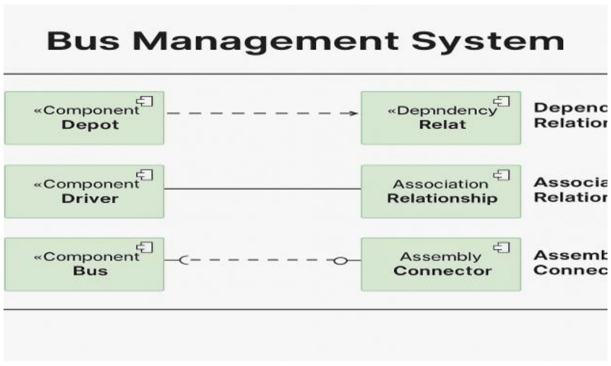
Function: Define how components communicate with each other, ensuring that components can be developed and maintained independent.



3.Relationships

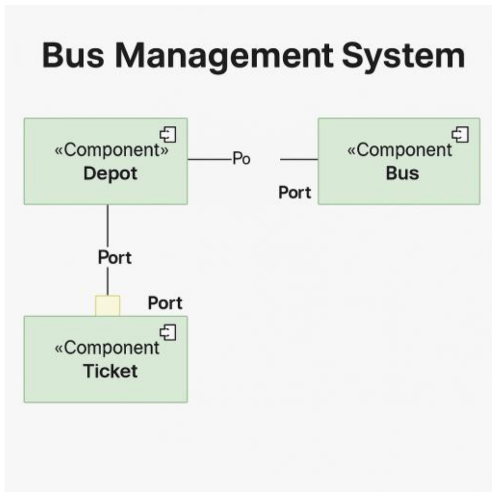
Depict the connections and dependencies between components and interfaces.

- **Symbol:** Lines and arrows.
  -Dependency (dashed arrow): Indicates that one component relies on another.
  -Association (solid line): Shows a more permanent relationship between components.
  -Assembly connector: Connects a required interface of one component to a provided interface of another.
- **Function:** Visualize how components interact and depend on each other, highlighting communication paths and potential points of failure.



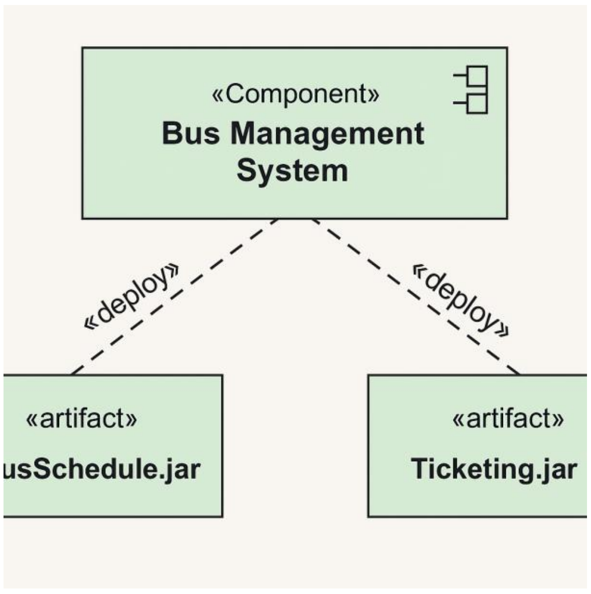
4. Ports

Role: Represent specific interaction points on the boundary of a component where interfaces are provided or required.
Symbol: Small squares on the component boundary.
Function: Allow for more precise specification of interaction points, facilitating detailed design and implementation.



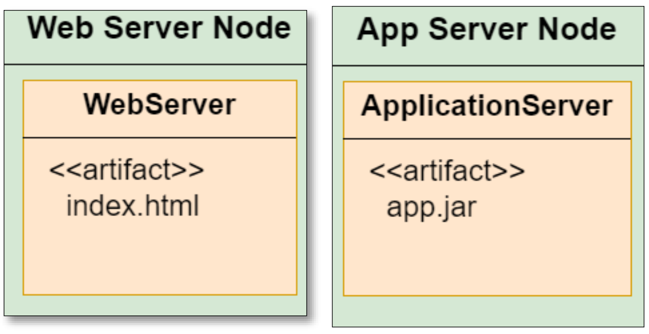
5. Artifacts

Represent physical files or data that are deployed on nodes.
Symbol: Rectangles with the artifact stereotype («artifact»).
Function: Show how software artifacts, like executables or data files, relate to the components.

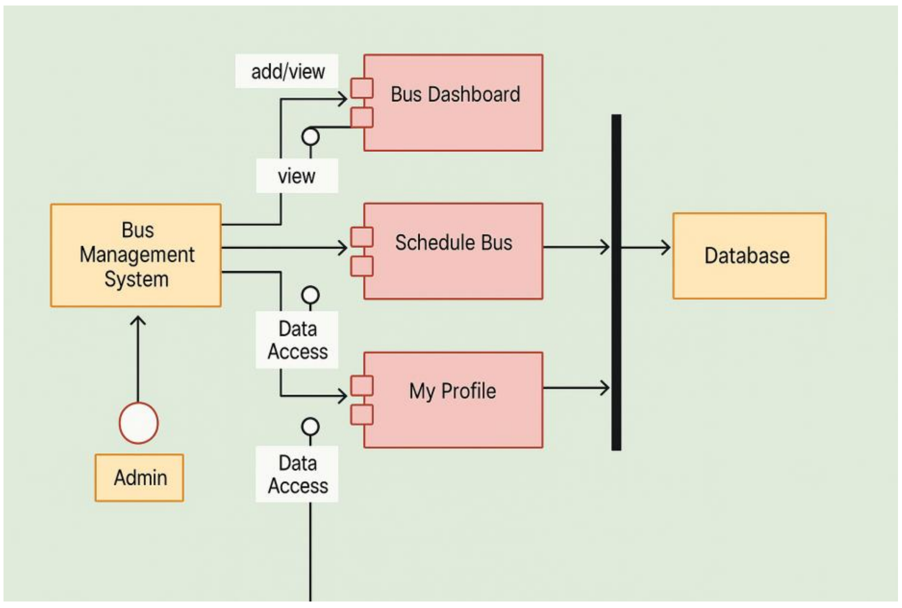


6. Nodes

Represent physical or virtual execution environments where components are deployed.
Symbol: 3D boxes.
Function: Provide context for deployment, showing where components reside and execute within the system’s infrastructure.



7.1.5 Component Diagram of A Bus Management system



Enroll no.

Class-Batch