EXPERIMENT – 5

AIM:

To draw use case diagram for Hostel Management System.

Theory:

A use case diagram is a visual representation in software engineering that shows how users or external systems interact with a system or software application, using actors (representing users) and use cases (representing system functions). It helps depict the system's functionality and user interactions, aiding in requirements analysis and system design.

Actors:

Actors are external entities or users who interact with the system being modelled. They are represented as stick figures or icons. Actors initiate use cases and are essential for defining system boundaries.

Use Cases:

Use cases represent specific functionalities or actions that the system can perform. They are depicted as ovals or ellipses and labelled with descriptive names. Use cases help define the system's behaviour and functionality.

Relationships:

Relationships show how actors and use cases are connected within the diagram.

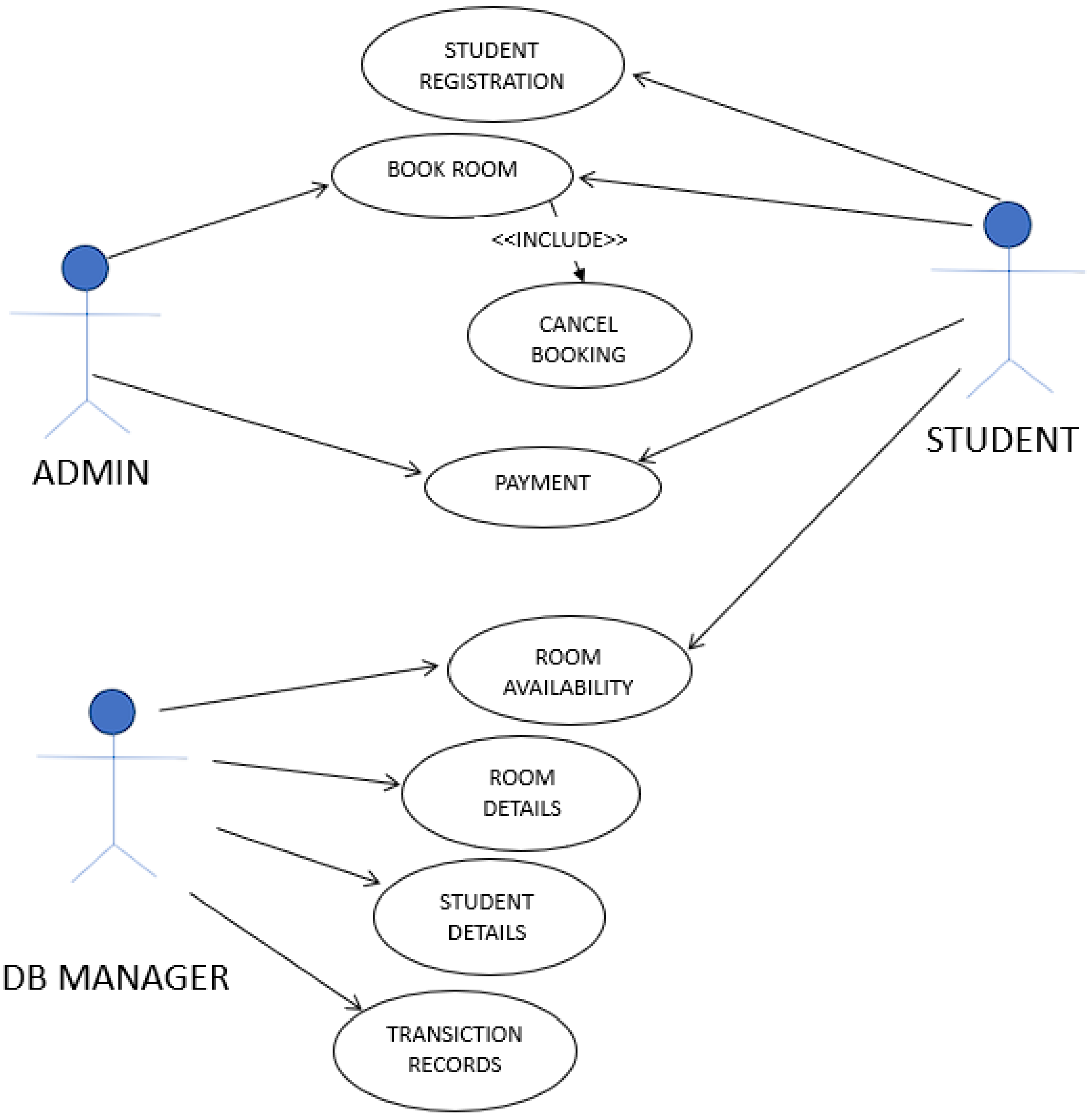
There are three primary types of relationships:

• Associations: Lines connecting actors and use cases, representing a connection.

• Extends: Arrows indicating optional or exceptional behaviour that extends a use case.

• Includes: Arrows showing the inclusion of one use case within another.

Use-Case Diagram:



HOSTEL MANAGEMENT SYSTEM

EXPERIMENT – 6

AIM:

To draw structural view diagram: Class Diagram and Object Diagram for Hostel Management System.

Theory:

Class Diagram:

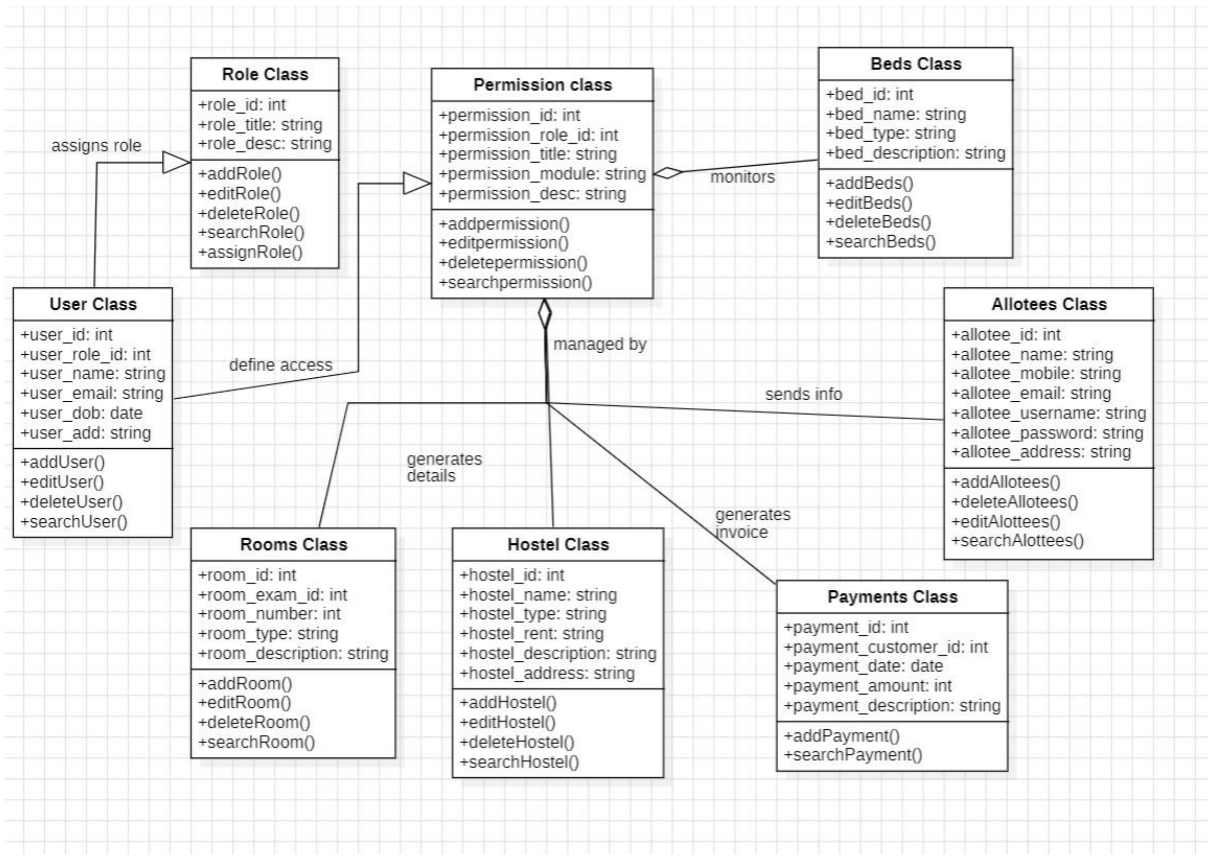
A class diagram is a type of UML (Unified Modelling Language) diagram that represents the structure and relationships of classes within a system or software application. It provides a visual representation of the static structure of the system, showing classes, their attributes, methods, and the relationships between classes. In a class diagram, classes are typically depicted as rectangles, with the class name at the top, attributes in the middle section, and methods at the bottom. Associations between classes are represented by lines connecting them, and additional information such as multiplicity, roles, and constraints can be included to detail the relationships.

Object Diagram:

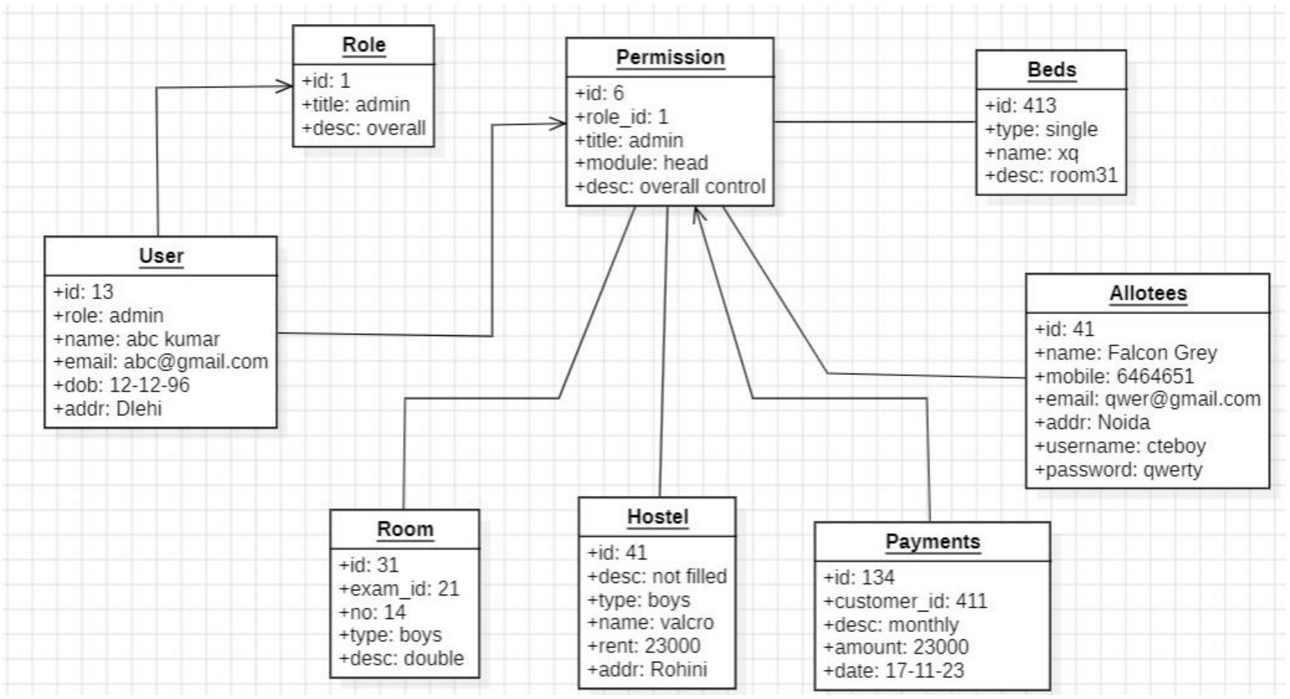
An object diagram is another type of UML diagram that focuses on representing instances of classes and the relationships between these instances at a specific point in time. It provides a snapshot of the objects and their relationships in a system, illustrating how objects collaborate and interact during a particular scenario or scenario instance. In an object diagram, objects are depicted as rectangles, like classes in a class diagram, and lines connecting them represent the relationships between these objects. Object diagrams help visualize the runtime aspects of a system and are useful for understanding specific instances and their interactions.

In summary, while a class diagram describes the static structure of a system by illustrating classes and their relationships, an object diagram provides a snapshot of the dynamic runtime behaviour by representing instances of these classes and their interactions at a specific moment. Together, these diagrams contribute to a comprehensive understanding of both the static and dynamic aspects of a software system.

CLASS DIAGRAM:



OBJECT DIAGRAM:



EXPERIMENT – 7

AIM:

To draw behavioural view diagram: State chart Diagram and Activity Diagram for Hostel Management System.

Theory:

Behavioural View Diagrams:

1. State Chart Diagram:

   A State Chart Diagram is a type of UML diagram that illustrates the dynamic behaviour of a system by representing the various states an object can exist in and the transitions between these states. It is particularly useful for modelling the behaviour of an entity in response to events and stimuli. State Chart Diagrams consist of states, transitions, events, and actions. States are depicted as rounded rectangles, transitions as arrows between states, and events trigger transitions. This diagram provides a clear visualization of the different states an object or system can go through and the events that cause these transitions.

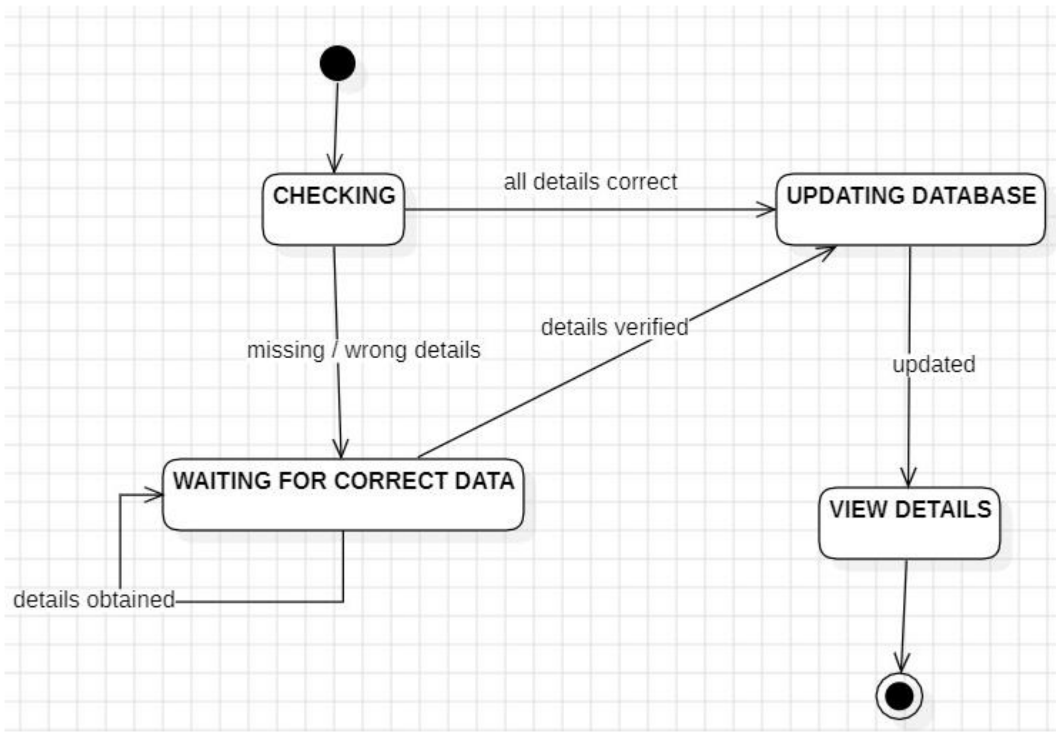
2. Activity Diagram:

   An Activity Diagram is another UML diagram that focuses on the dynamic aspects of a system, depicting the flow of activities or processes within it. It is especially useful for modelling business processes, use cases, or the workflow of a system. Activities are represented by rounded rectangles, and arrows denote the flow of control between activities. Decision points, forks, and joins can be used to model conditional or parallel flows. Activity Diagrams help in understanding the sequence of activities and the decision points in a process, making them valuable for both analysis and design phases of software development.

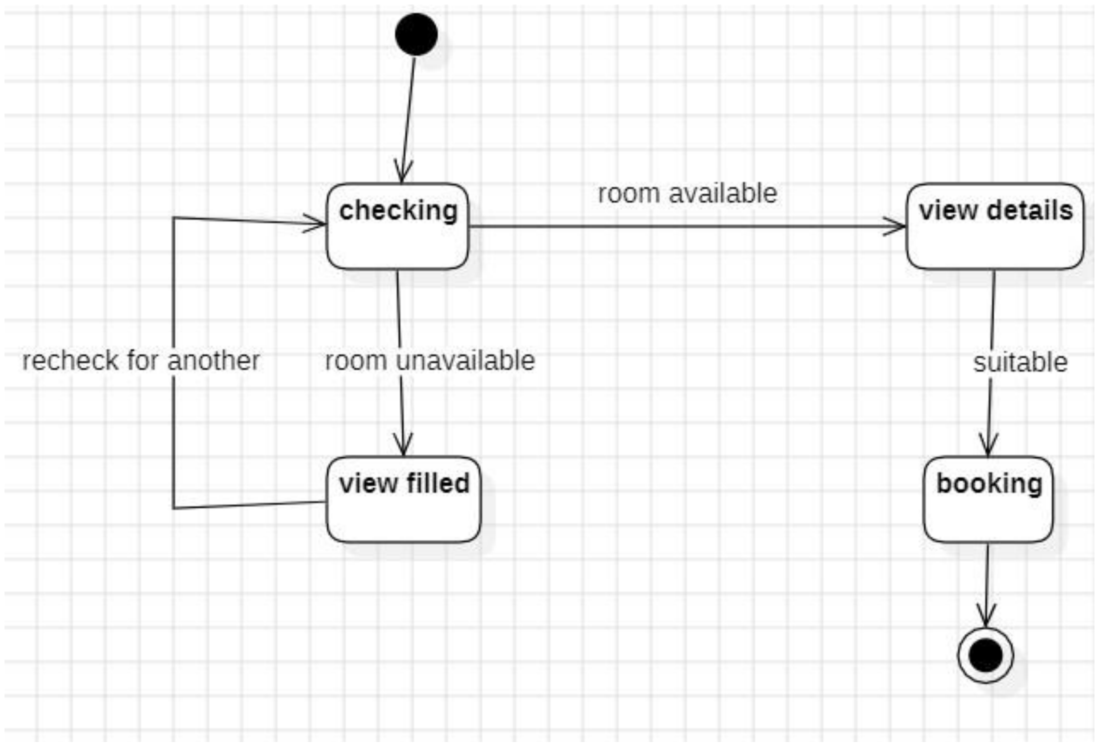
In summary, both State Chart Diagrams and Activity Diagrams are part of the UML Behavioural View, aiming to provide insights into the dynamic behaviours of a system, but they focus on different aspects. State Chart Diagrams emphasize the states and transitions of an object, while Activity Diagrams emphasize the flow of activities within a system.

State Chart Diagram:

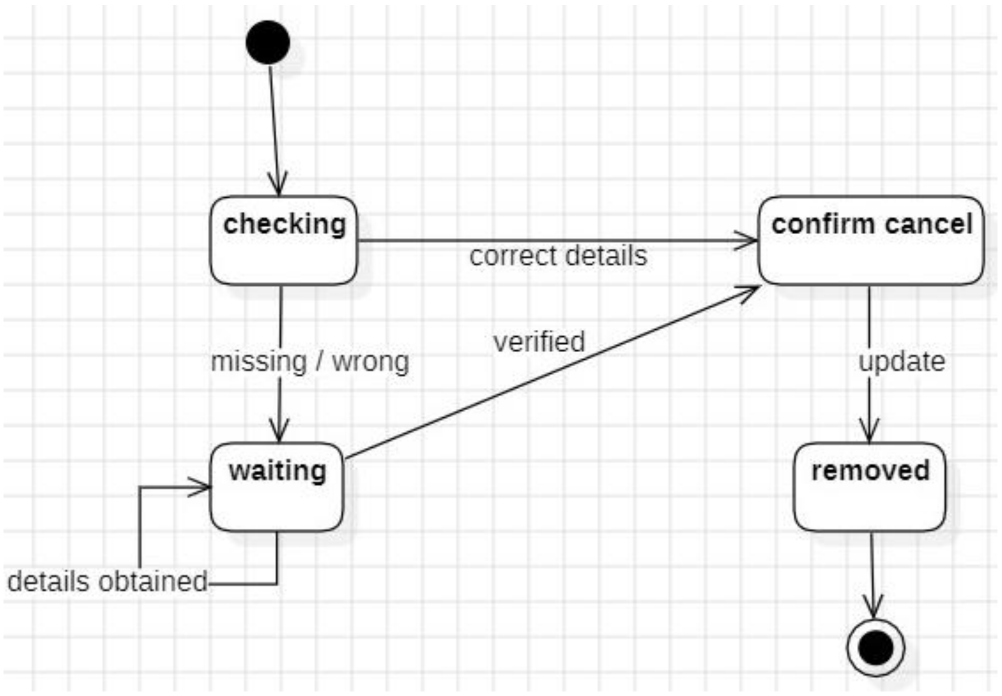
1.STUDENT DETAILS



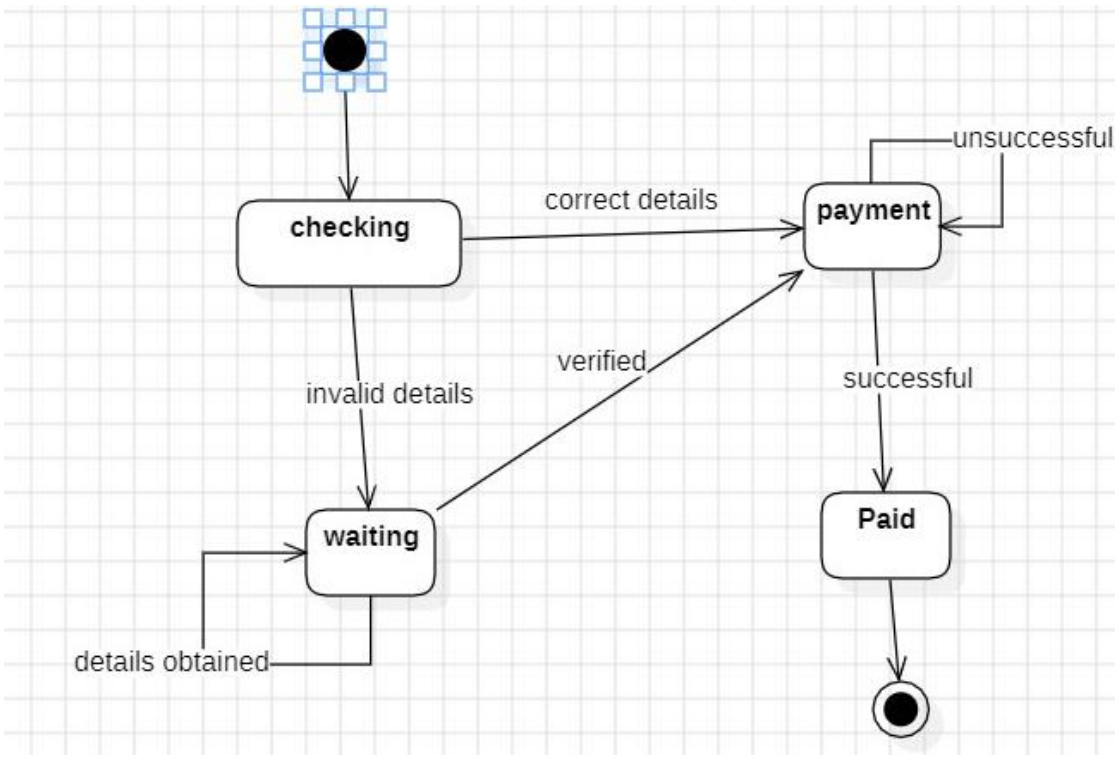
2.ROOM BOOKING



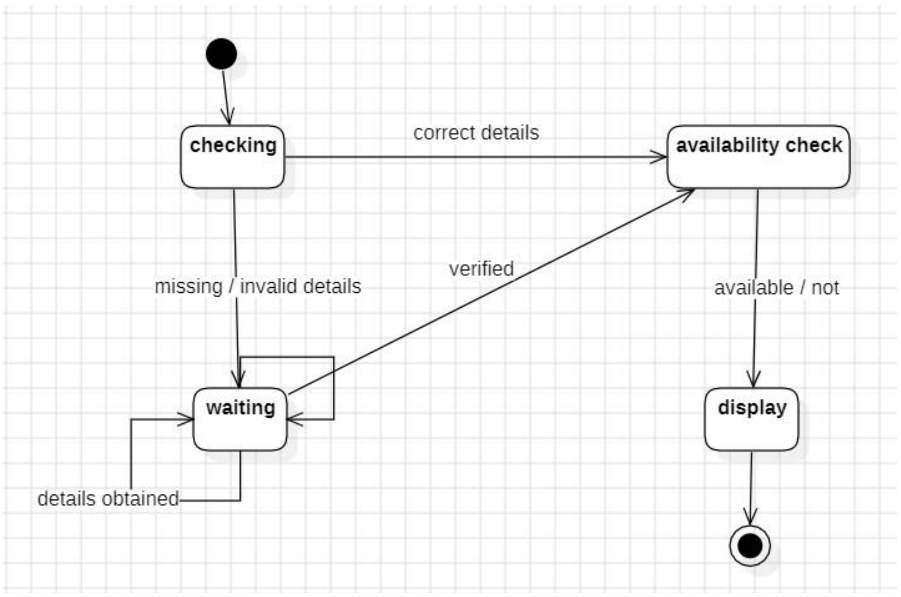
3.CANCEL BOOKING



4.PAYMENT

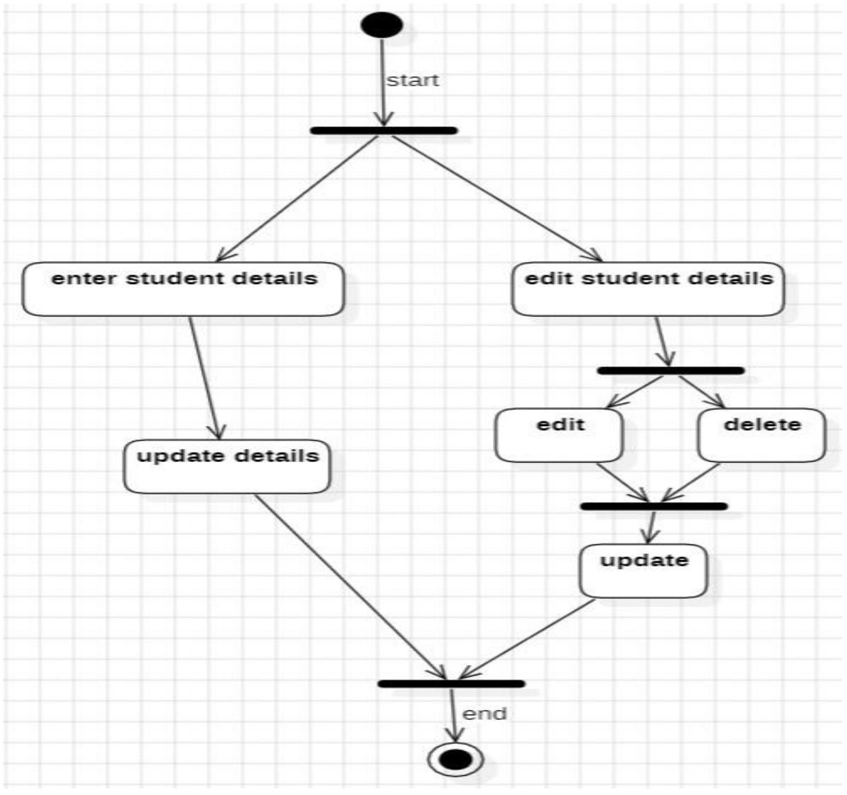


5.ROOM AVAILABILITY

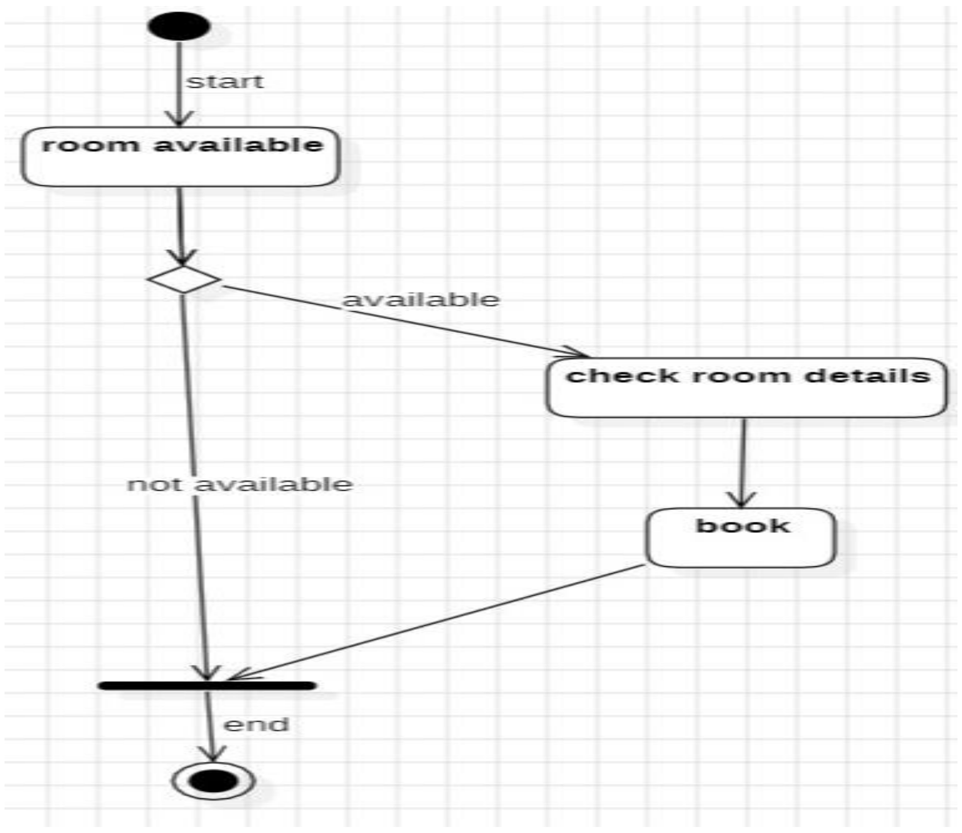


Activity Diagram:

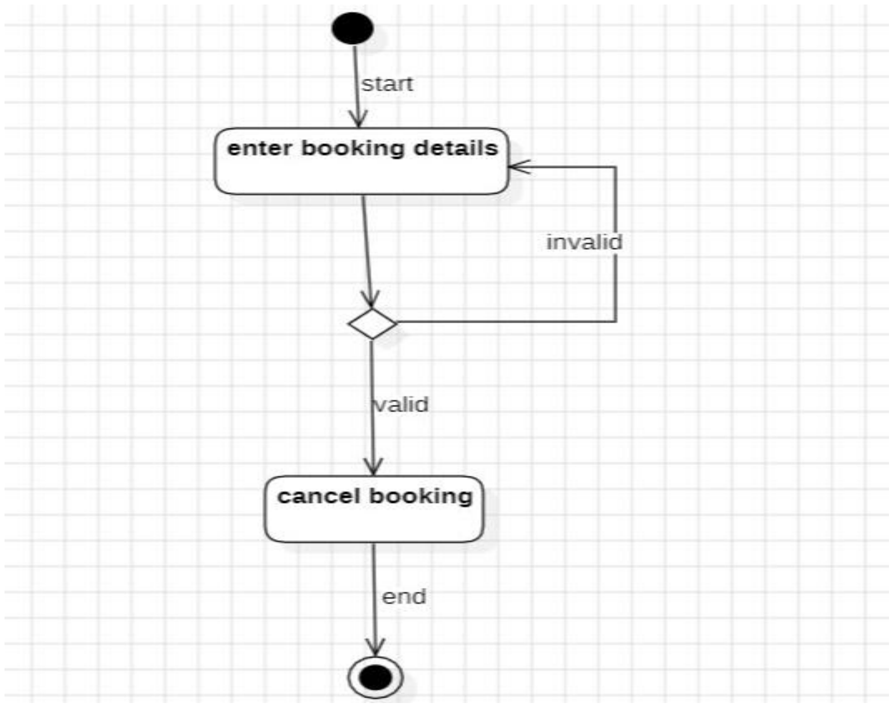
1.STUDENT DETAILS



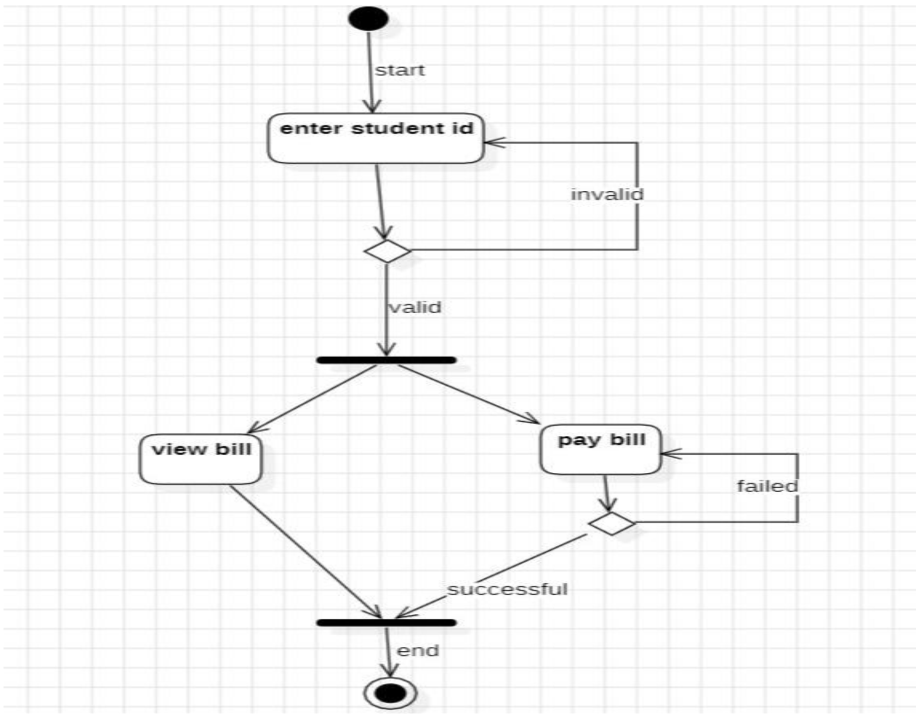
2.ROOM BOOKING



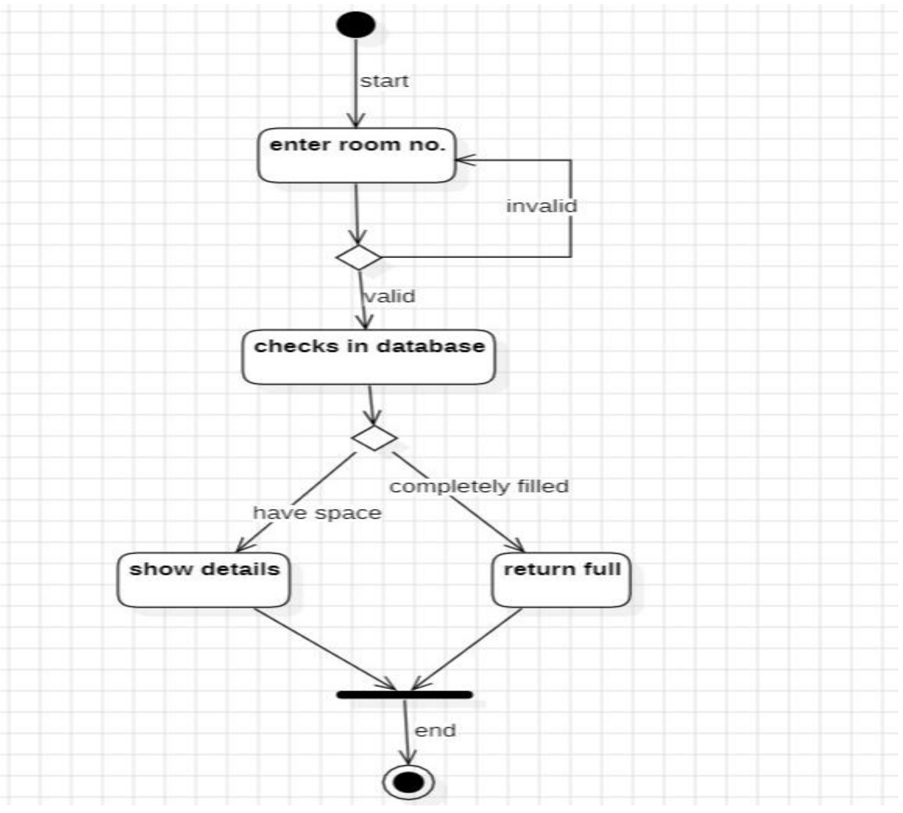
3.CANCEL BOOKING



4.PAYMENT



5.ROOM AVAILABILITY



EXPERIMENT – 8

AIM:

To perform the behavioural view diagram: Sequence Diagram and Collaboration Diagram for Hostel Management System.

Theory:

Sequence Diagram:

A Sequence Diagram is a type of UML diagram that illustrates the interactions between objects or components in a system over time. It focuses on the sequence of messages exchanged between these entities to achieve a specific functionality or to represent a particular scenario. Sequence Diagrams are particularly useful for visualizing the dynamic behaviour of a system during a specific use case or operation. The diagram typically consists of lifelines (vertical lines representing objects or components), messages (horizontal arrows indicating communication between lifelines), and activations (rectangular boxes representing the duration of an operation). Sequence Diagrams help in understanding the chronological order of interactions, the flow of control, and the exchange of messages among different elements in a system.

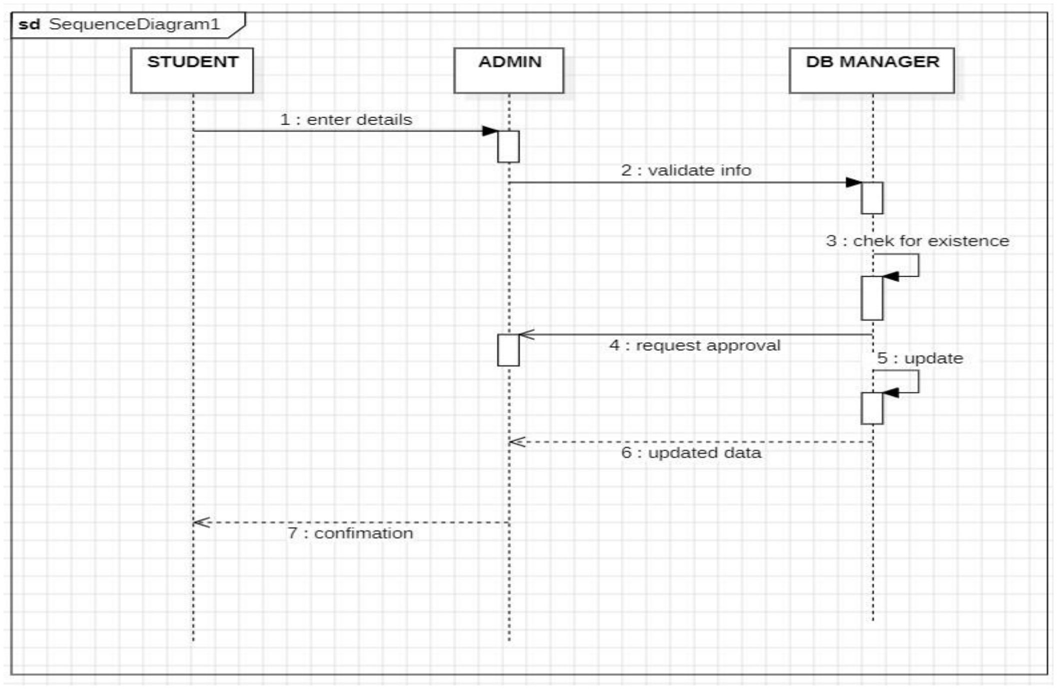
Collaboration Diagram:

A Collaboration Diagram, also known as a Communication Diagram, is another type of UML diagram that illustrates the interactions between objects or components in a system. Like Sequence Diagrams, Collaboration Diagrams focus on the relationships and interactions among system elements, but they emphasize the structural organization of these interactions rather than their chronological order. In a Collaboration Diagram, objects or components are represented by rectangles, and communication links between them are shown using lines with arrows indicating the flow of messages. The emphasis is on the structural organization of the system and the relationships between objects during a specific scenario or use case. Collaboration Diagrams are particularly useful for understanding the overall structure of a system and how different elements collaborate to achieve a particular functionality.

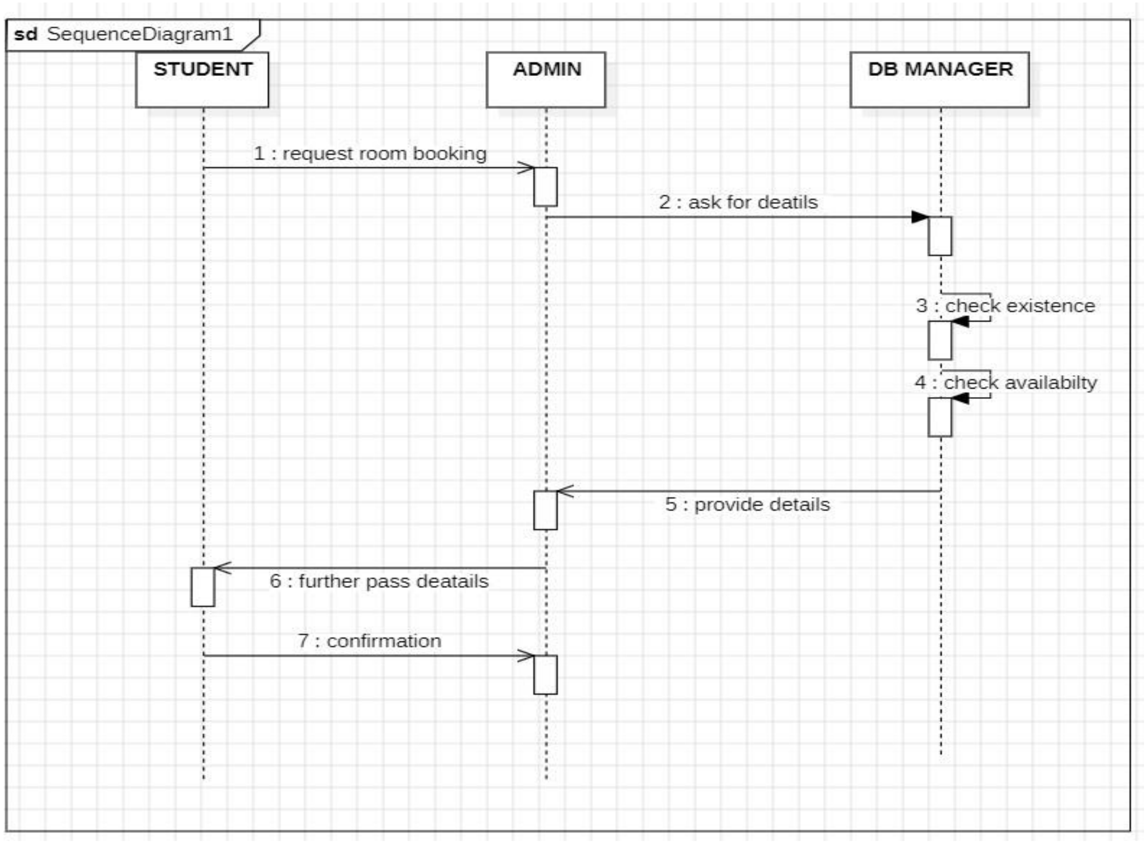
In summary, while Sequence Diagrams emphasize the chronological order of interactions over time, Collaboration Diagrams focus on the structural aspects of how objects or components collaborate within a system. Both diagrams contribute to a comprehensive understanding of the dynamic behaviour and structural organization of a software system.

Sequence Diagram:

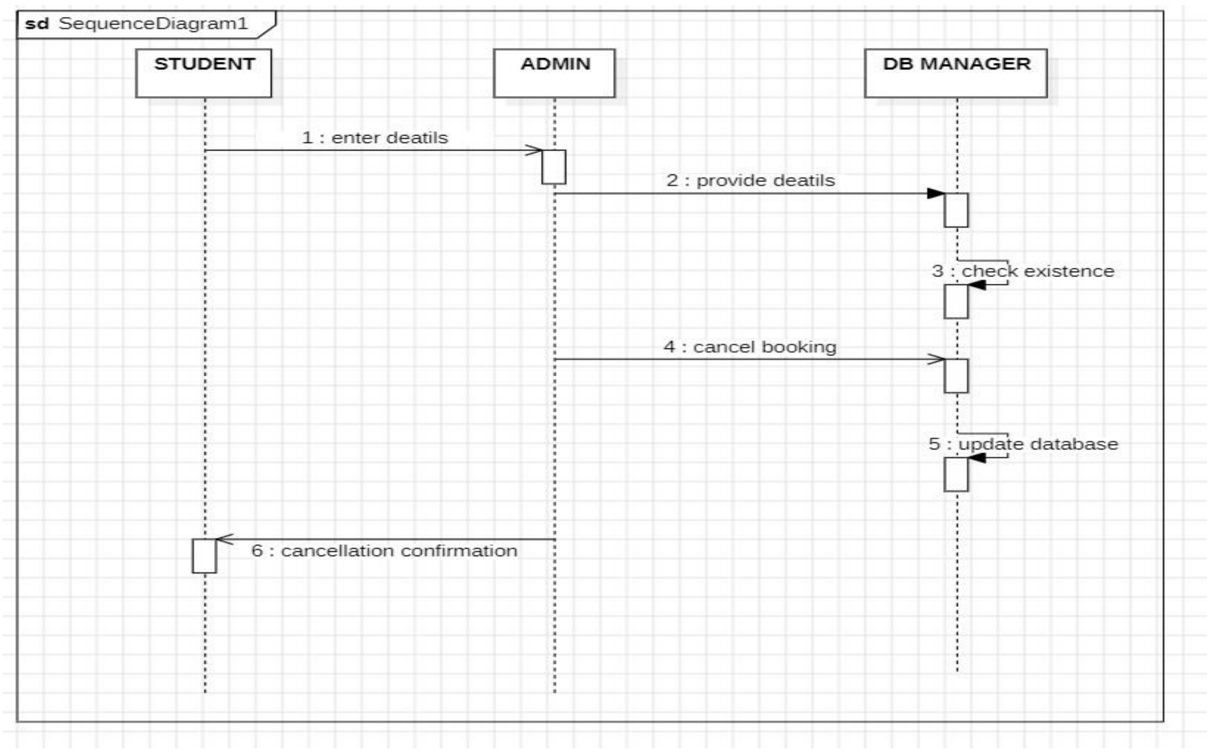
1.STUDENT DETAILS



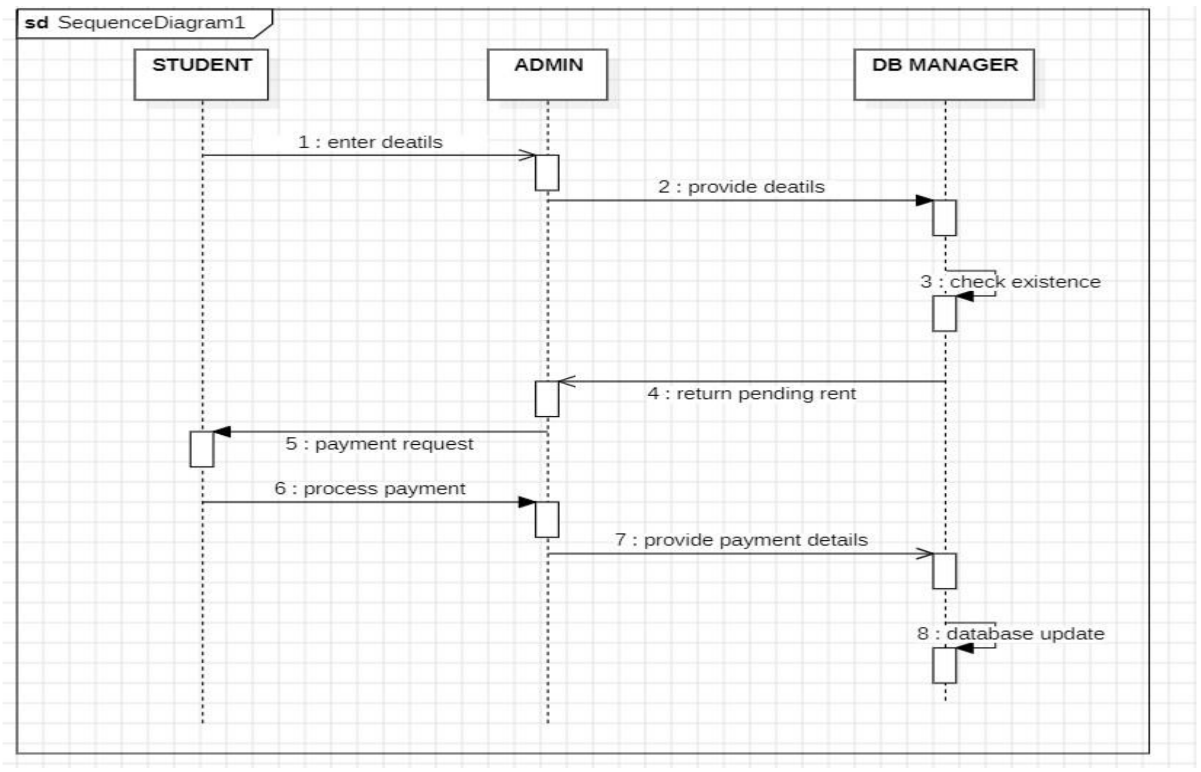
2.ROOM BOOKING



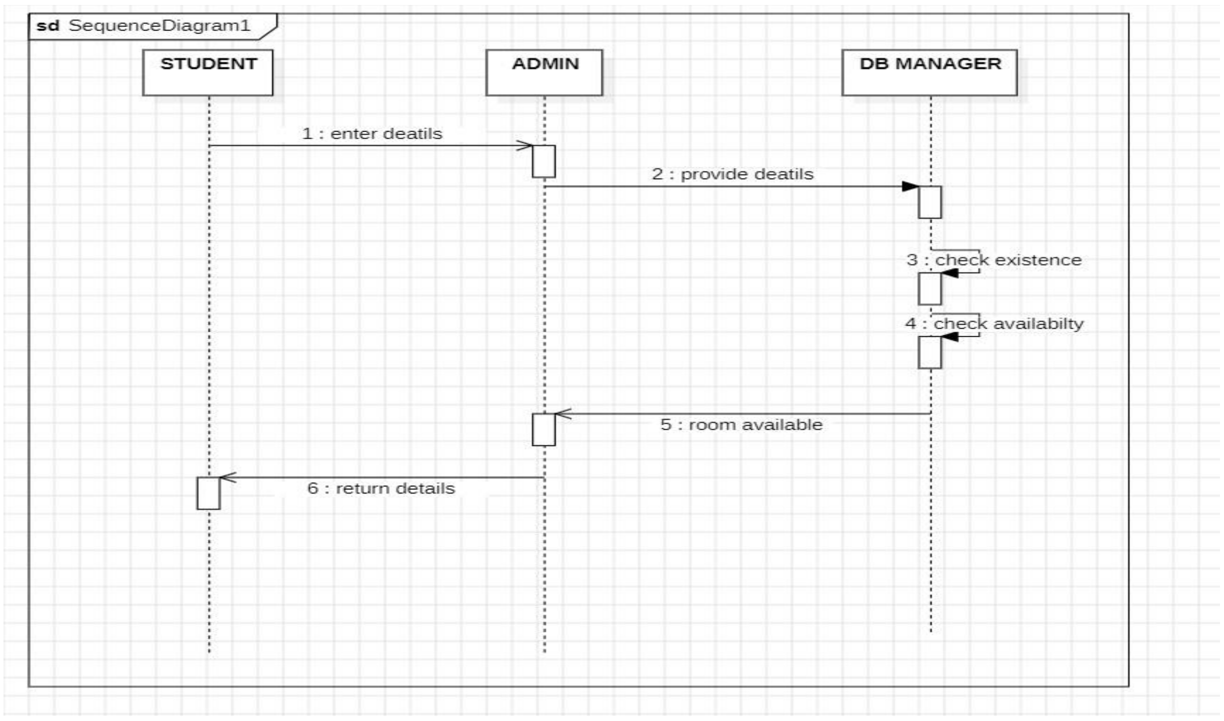
3.CANCEL BOOKING



4.PAYMENT

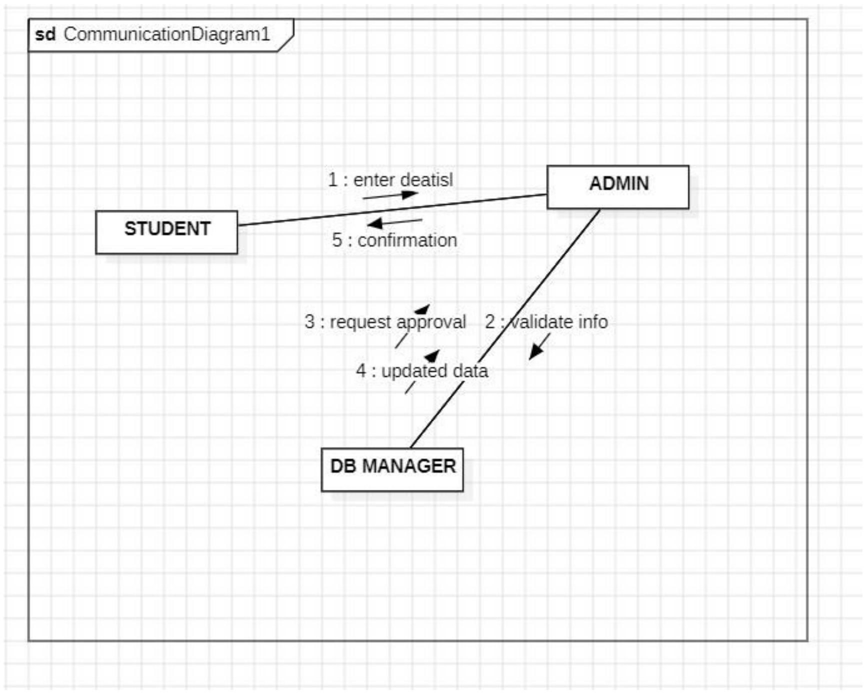


5.ROOM AVAILABILITY

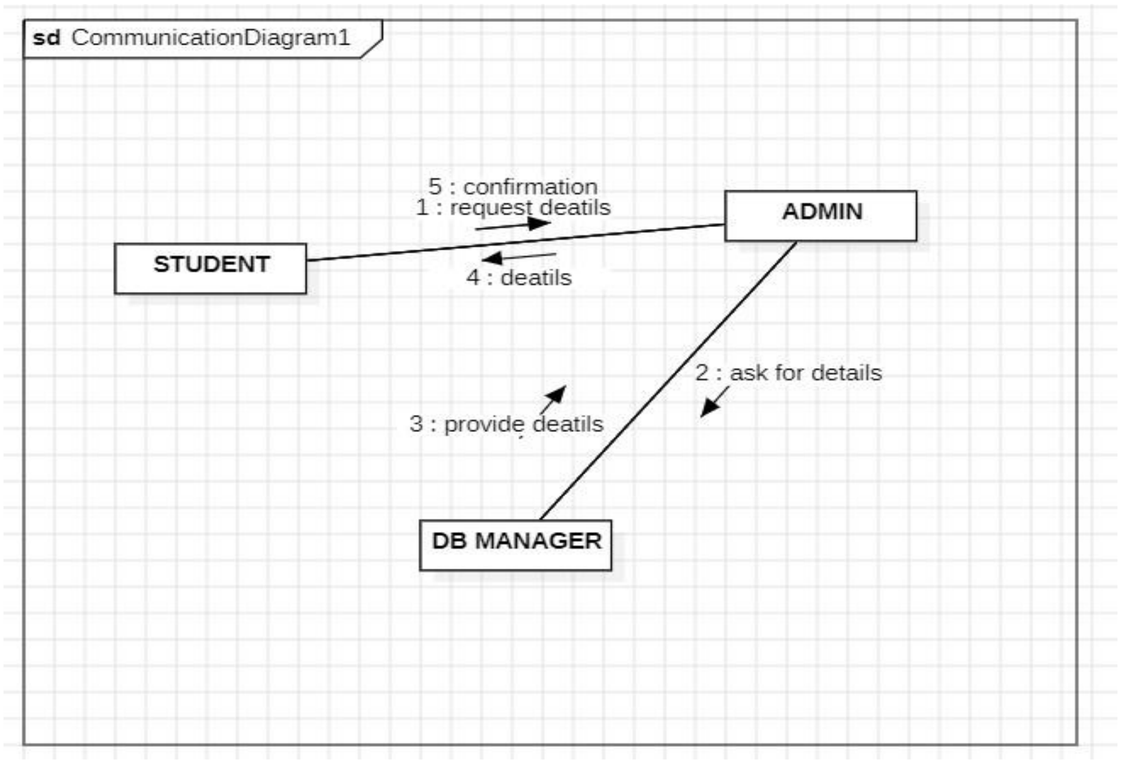


Collaboration Diagram:

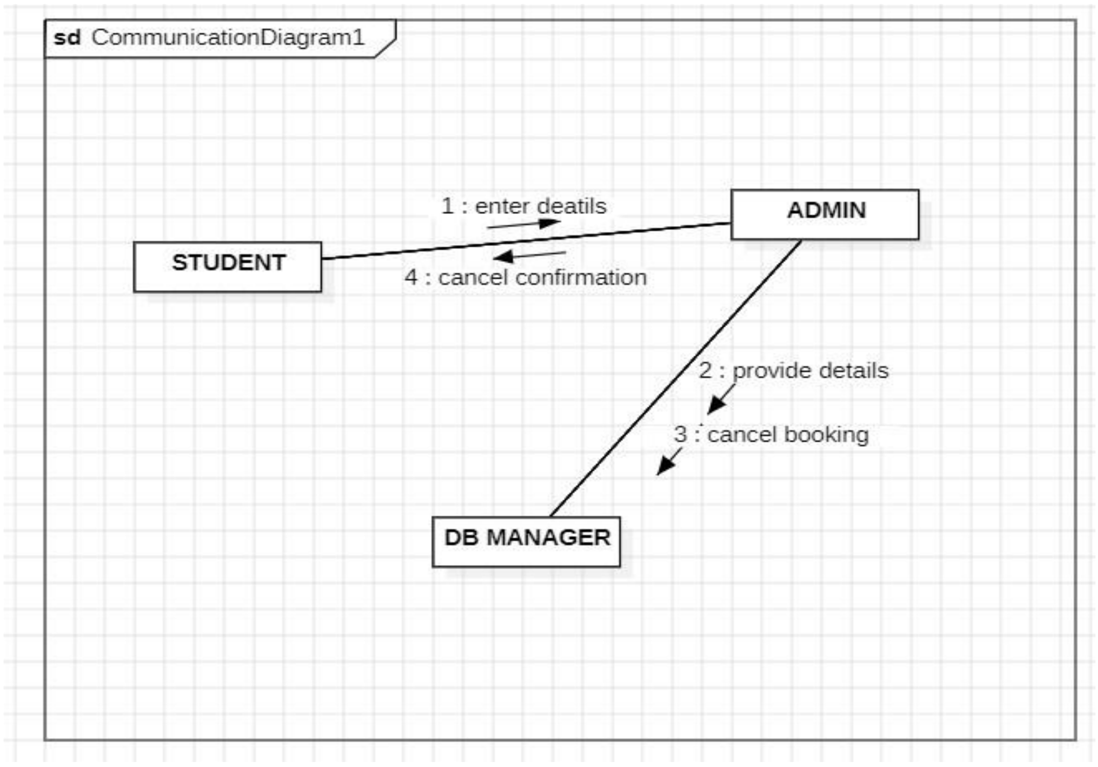
1.STUDENT DETAILS



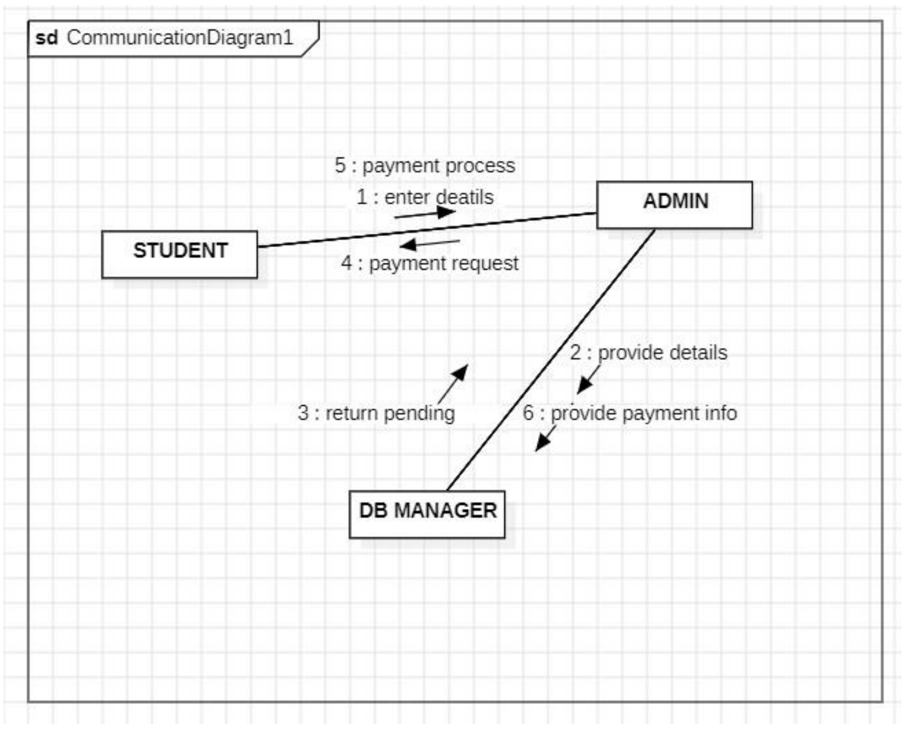
2.ROOM BOOKING



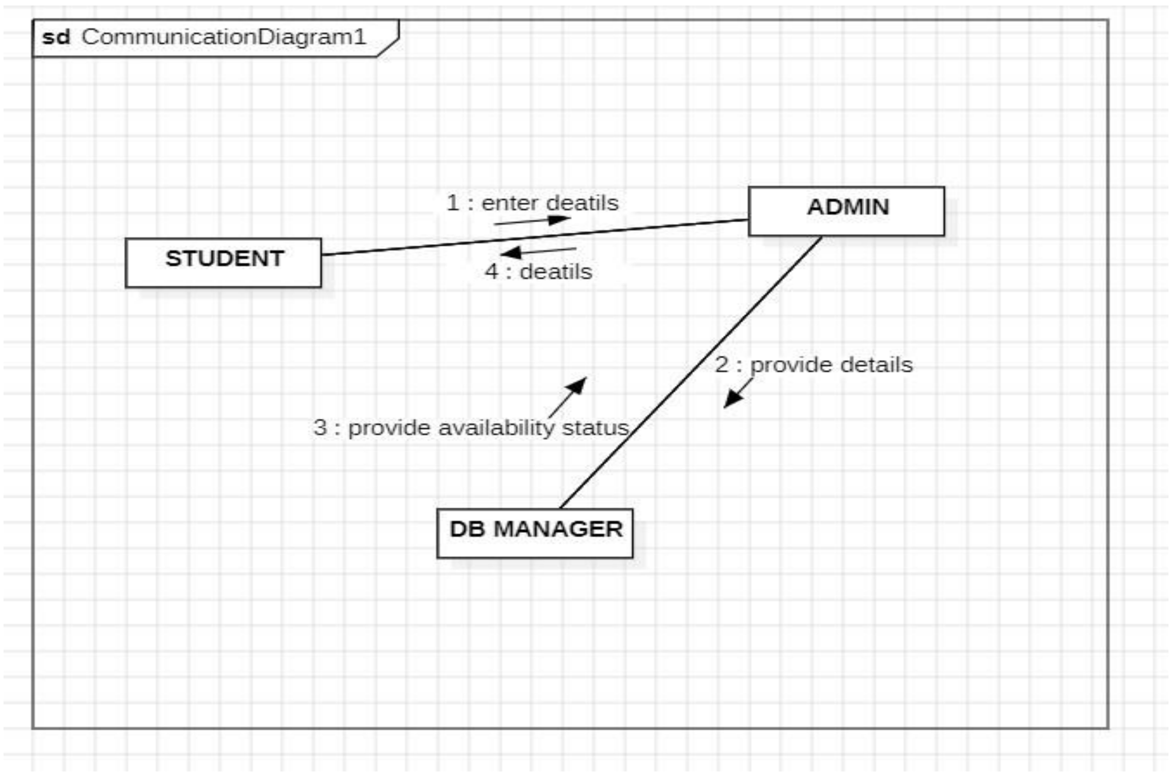
3.CANCEL BOOKING



4.PAYMENT



5.ROOM AVAILABILITY



EXPERIMENT – 9

AIM:

To perform the implementation view diagram: Component Diagram for Hostel Management System.

Theory:

A Component Diagram is a type of UML (Unified Modelling Language) diagram that depicts the organization and dependencies between components in a software system. Components are modular, independent, and replaceable parts of a system that encapsulate related functionality. The primary purpose of a Component Diagram is to show the high-level structure of a system and how its components interact.

Key elements in a Component Diagram include:

Component:

Components are represented by rectangles, and each rectangle typically contains the name of the component. A component can be a class, module, or any other modular unit that encapsulates a set of related functionalities.

Interface:

Interfaces define the contract or set of operations that a component provides or requires. Interfaces are depicted with a circle, and the name of the interface is written inside it. Arrows connecting components to interfaces show the direction of the dependency.

Dependency Arrows:

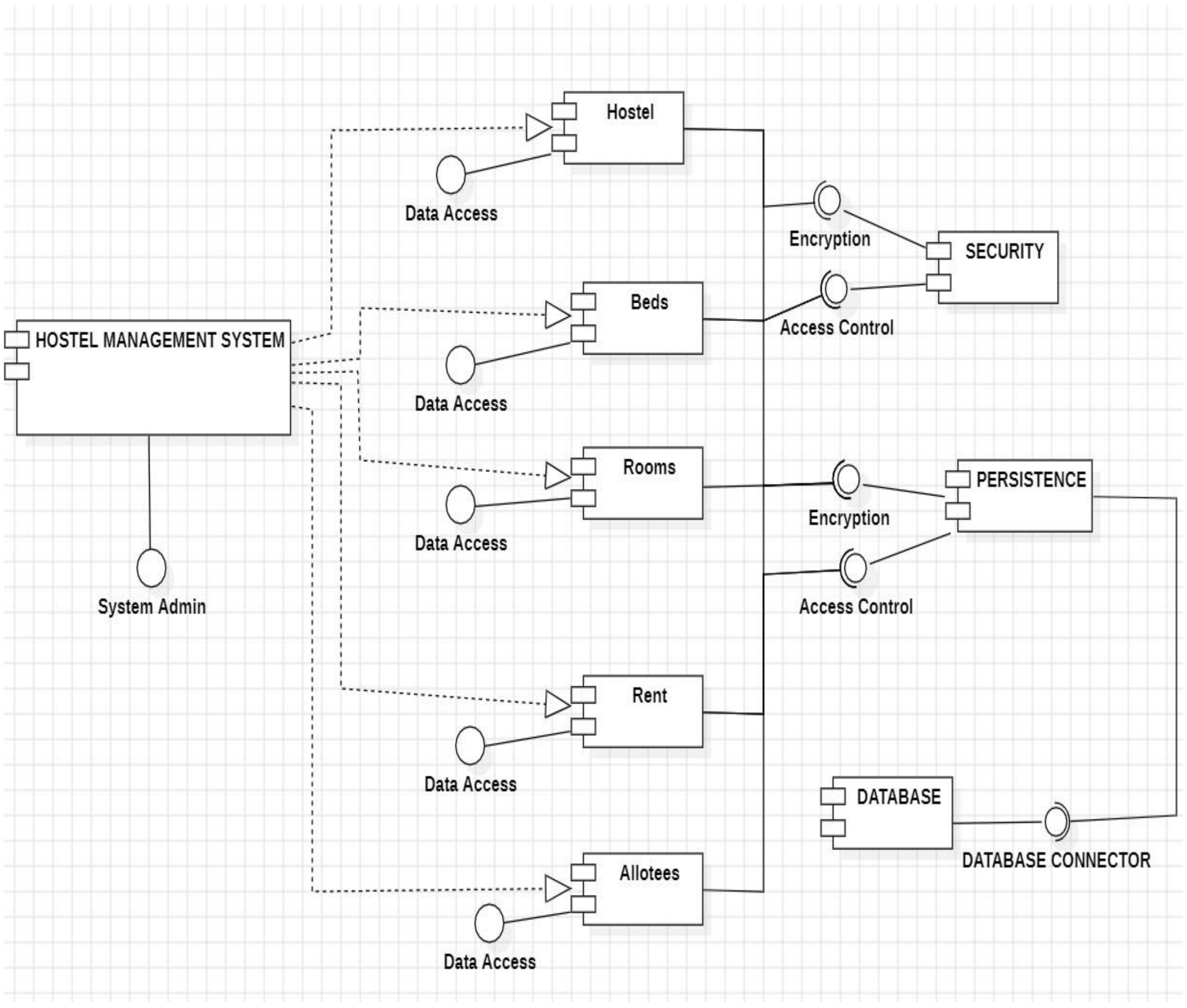
Arrows between components indicate dependencies, showcasing the relationships between components. Dependencies can be generalizations, realizations, associations, or dependencies based on provided or required interfaces.

Packages:

Components can be organized into packages, which are depicted as folders. Packages help in grouping related components, providing a higher level of abstraction and organization.

Component Diagrams are particularly beneficial in the early stages of software design and architecture, providing a visual representation of how different components interact within a system. They help software architects, developers, and stakeholders to understand the overall structure of the software, identify potential bottlenecks, and plan for modularity and reusability. Component Diagrams play a crucial role in visualizing and communicating the system's architecture and are an integral part of the UML suite for modelling software systems.

Component Diagram:



EXPERIMENT – 10

AIM:

To perform the implementation view diagram: Deployment Diagram for Hostel Management System.

Theory:

A Deployment Diagram is a type of UML (Unified Modelling Language) diagram that models the physical deployment of software components in a hardware environment. It provides a visual representation of how software artifacts, such as modules, processes, and components, are distributed across hardware nodes like servers, computers, or devices in a system. Deployment Diagrams are particularly useful for understanding the physical architecture of a software system and its runtime environment.

Key elements in a Deployment Diagram include:

Nodes:

Nodes represent hardware devices or computing resources such as servers, workstations, or other physical entities. Each node is depicted as a box, and the name of the node is usually written inside.

Components:

Components represent software artifacts, modules, or services that are deployed on nodes. Components are typically depicted as rectangles, and lines connecting components to nodes indicate the deployment relationships.

Artifacts:

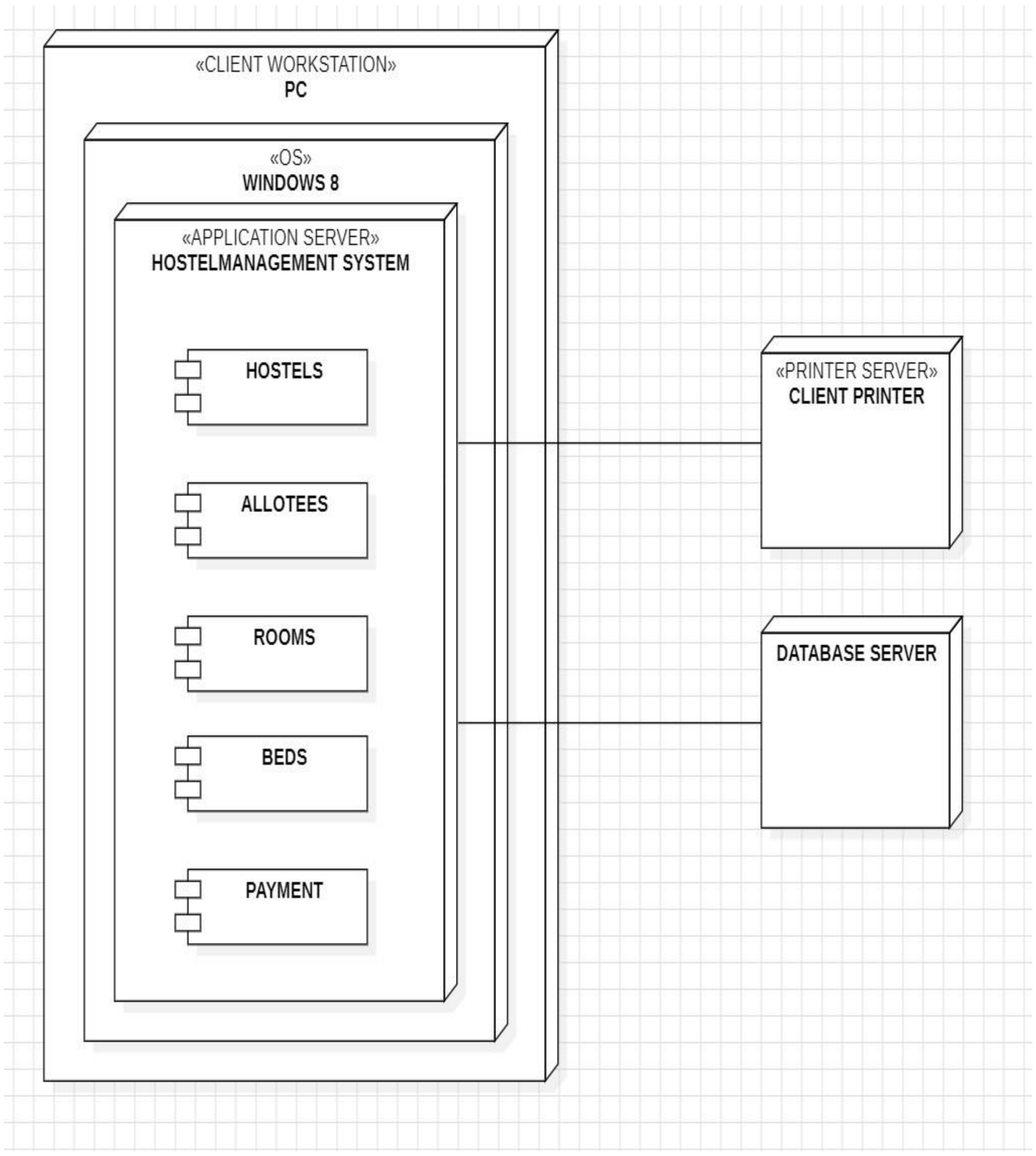
Artifacts represent the physical files or data used or produced by components. They can be shown inside components to illustrate the actual data or files associated with the software.

Communication Paths:

Communication paths, shown as arrows or lines, represent the communication or interaction between components or nodes. They illustrate the flow of data or control between different parts of the system.

Deployment Diagrams help stakeholders, including system architects and administrators, to visualize and understand how the software components interact with the hardware infrastructure. They are valuable for planning, optimizing, and documenting the deployment configuration of a system. Deployment Diagrams are especially useful in scenarios where a system involves multiple servers, distributed components, and complex network configurations. Overall, Deployment Diagrams contribute to a holistic view of the system's physical deployment, aiding in system analysis, design, and maintenance.

Deployment Diagram:


«CLIENT WORKSTATION»
PC
«OS»
WINDOWS 8
«APPLICATION SERVER»
HOSTELMANAGEMENT SYSTEM
HOSTELS
ALLOTEES
ROOMS
BEDS
PAYMENT
«PRINTER SERVER»
CLIENT PRINTER
DATABASE SERVER