Experiment No: 1

OBJECTIVE:

Prepare an SRS document in line with the IEEE recommended standards.

Theory:

A software requirements specification (SRS) describes a software system to be developed. It lays out functional and non-functional requirements and may include a set of use cases that describe user interactions that the software must provide.

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules.

The software requirements specification document lists enough and necessary requirements that are required for the project development. To derive the requirements, we need to have a clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

Implementation:

Expense Management System (EMS):

Introduction

Purpose:
This document describes the Expense Management System (EMS). It defines the software's functionalities, constraints, and interfaces. The primary goal is to help users efficiently track, manage, categorize, and report their financial expenses.

Scope:

EMS is a web-based platform that allows users (individuals or organizations) to:

- Record daily expenses.

- Categorize expenses (Food, Travel, Office, Personal, etc.).

- Generate monthly/annual expense reports.

- Set budget limits and receive alerts.

- Visualize expenses via graphs and charts.

Objective:

Provide a user-friendly, secure, and efficient platform to manage personal and organizational expenditures.

Overview:

The document details system description, specific functional and non-functional requirements, and interface requirements.

References:

- IEEE Standards for Software Requirements Specifications

- UML Diagrams Wikipedia

Functional Requirements:

- User Registration/Login

- Add/Edit/Delete Expenses

- Categorize Expenses

- Generate Reports

- Set Monthly Budget

- Send Budget Exceed Alerts

- Export Data (PDF/Excel)

- Dashboard with Analytics

Interfaces Requirements:

User Interface:

- Keyboard and mouse.

Hardware Interface:
Minimum:

- Processor: Pentium III

- HDD: 80GB

- RAM: 256MB

Preferred:

- Processor: Pentium IV

- HDD: 100GB

- RAM: 512MB

Software Interface:

- Windows XP/Windows 10+

- XAMPP/Apache Server (for web version)

- Chrome/Firefox Browser

Non-Functional Requirements:

ID	Requirement	Priority
NFR1	System shall encrypt all user data.	High
NFR2	System shall have 99.9% uptime.	High
NFR3	System dashboard shall load under 3 seconds.	Medium
NFR4	System shall be mobile-responsive.	Medium
NFR5	Data backups every 12 hours.	High

Performance Requirements:

- Expense entries must reflect instantly.

- Reports generated within 15 seconds.

Logical Database Requirements:

- MySQL database to store user profiles, expenses, budgets, and logs.

- Database backup every 12 hours.

Conclusion:

The SRS provides a blueprint to develop an efficient Expense Management System, clearly outlining all functionalities and constraints.

Experiment No:- 2

OBJECTIVE:-

Draw the use case diagram and specify the role of each of the actors. Also state the precondition, post condition and function of each of the use case.

Theory:-

A use case diagram at its simplest is a representation of a user's interaction with the system that
shows the relationship between the user and the different use cases in which the user
is involved. A use case diagram can identify the different types of users of a system
and the different use cases and will often be accompanied by other types of diagrams
as well.

Use case diagrams are considered for high level requirement analysis of a system. So
when the requirements of a system are analyzed the functionalities are captured in
use cases.
So we can say that uses cases are nothing but the system functionalities written in an
organized manner. Now the second things which are relevant to the use cases are the
actors. Actors can be defined as something that interacts with the system.

The actors can be human user, some internal applications or may be some external
applications. So in a brief when we are planning to draw an use case diagram we
should have the following items identified.
Functionalities to be represented as an
use case Actors. Relationships among
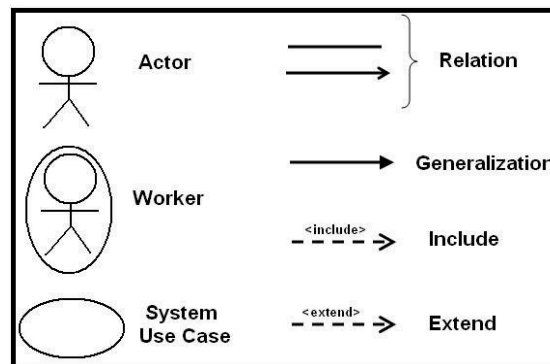the use cases and actors.

Implementation:

Use case diagram:
Role of each of the actors:

- Admin → Manage Users

- Admin, User → Login/Authentication

- Analyst → Add Expense

- Analyst → View Expense Report

- Analyst → Generate Reports

- Analyst → Receive Budget Alerts

- Analyst → Set Budget Goals

- Viewer → View Expense Report (read-only)
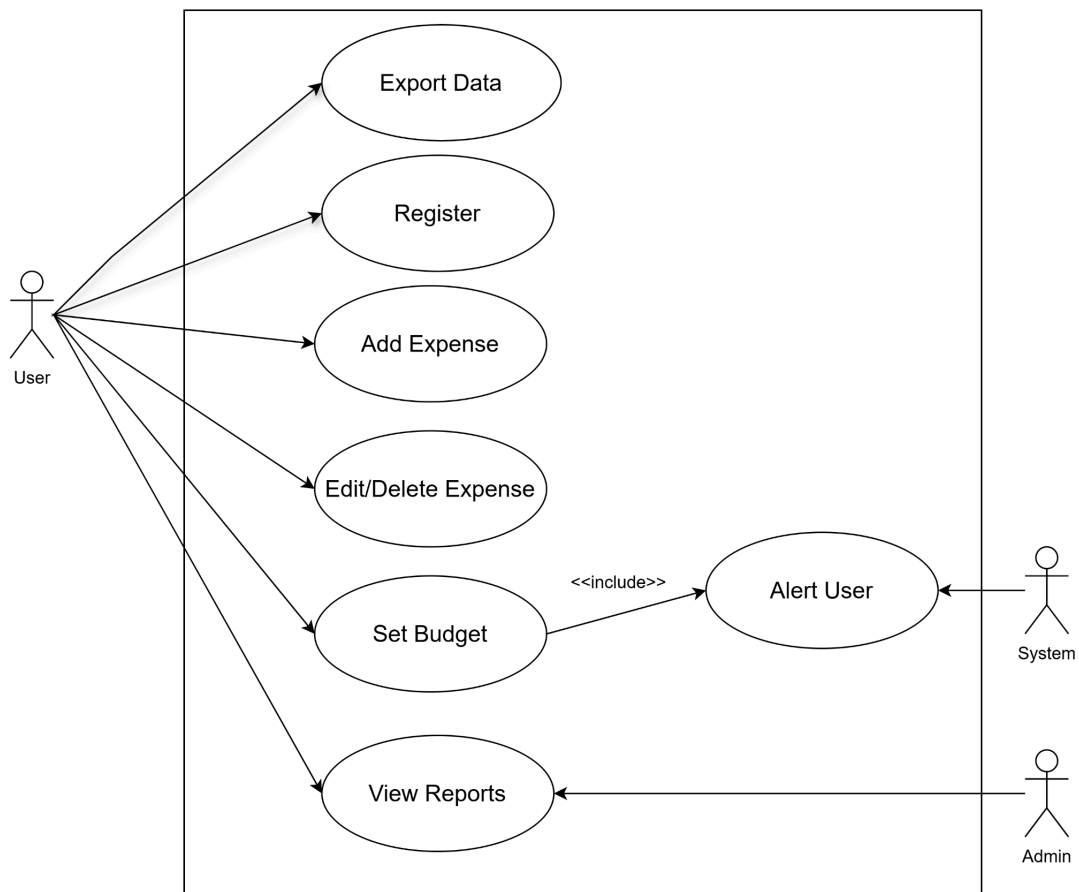
- External Systems → Provide Energy Data

Notations used:-

Class Diagram Relationship Type	Notation
Association	⟶
Inheritance	⟹
Realization/ Implementation	⇢ (dashed, hollow triangle)
Dependency	⇢ (dashed, open arrow)
Aggregation	⟶◇ (hollow diamond)
Composition	⟶◆ (filled diamond)

Use Case Diagram :

Include Relationship:

The "include" relationship denotes that one use case includes the functionality of another use case. It represents a common behavior that is shared by multiple use cases. The included use case is always executed whenever the base use case is executed. The arrow in the diagram points from the base use case to the included use case.
Example: In an online shopping system, the "Checkout" use case may include the "Login" and "View Cart" use cases. This means that whenever a customer proceeds to checkout, they must first log in and view their shopping cart.

Extend Relationship:

 The "extend" relationship denotes that the behaviour of a base use case can be extended by another use case under certain conditions. The extending use case adds optional or conditional behaviour to the base use case but is not necessarily executed every time the base use case is performed.
The arrow in the diagram points from the extending use case to the base use case, labelled with the

<<extend>> stereotype.

Output:- The use case diagram is made successfully.

Experiment No:- 3

## OBJECTIVE:-

 Draw the activity diagram.

Theory:-

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is some time considered as the flow chart. Although the diagrams looks like a flow chart but it is not. It shows different flow like parallel, branched, concurrent and single.

Notations used:-
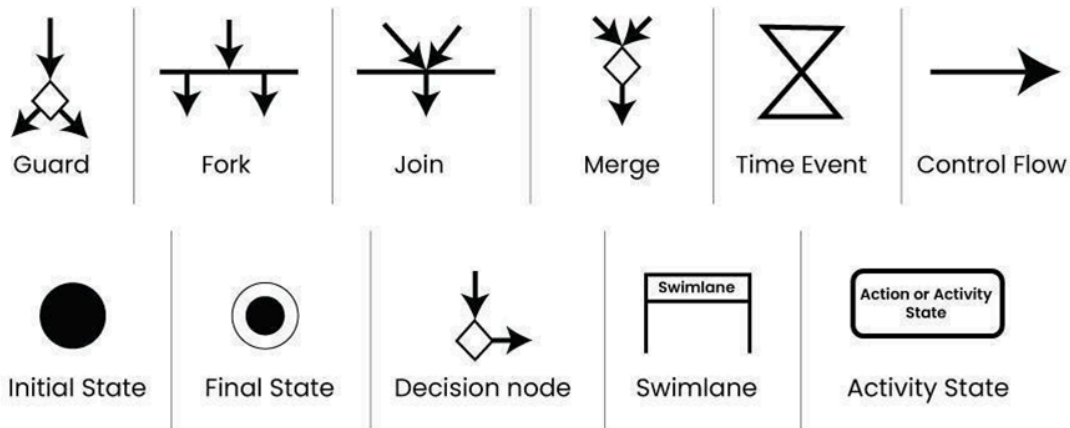
Initial State

The starting state before an activity takes place is depicted using the initial 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. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.

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.

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 activity state.

An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

Decision node and Branching

When we need to make a decision before deciding the flow of control, we use the decision node. The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.

Guard

A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.

Fork

Fork nodes are used to support concurrent activities. When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement. We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.

Join

Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.

Merge or Merge Event

Scenarios arise when activities which are not being executed concurrently have to be merged. We use the merge notation for such scenarios. We can merge two or more activities into one if the control proceeds onto the next activity irrespective of the path chosen.

Swimlanes

We use Swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram. It is not mandatory to use swimlanes. They usually give more clarity to the activity diagram. It's similar to creating a function in a program. It's not mandatory to do so, but, it is a recommended practice.

We use a rectangular column to represent a swimlane as shown in the figure above.

Time Event

This refers to an event that stops the flow for a time; an hourglass depicts it. We can have a scenario where an event takes some time to completed.

Final State or End State

The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

Activity Diagram for adding expenses and report generation :
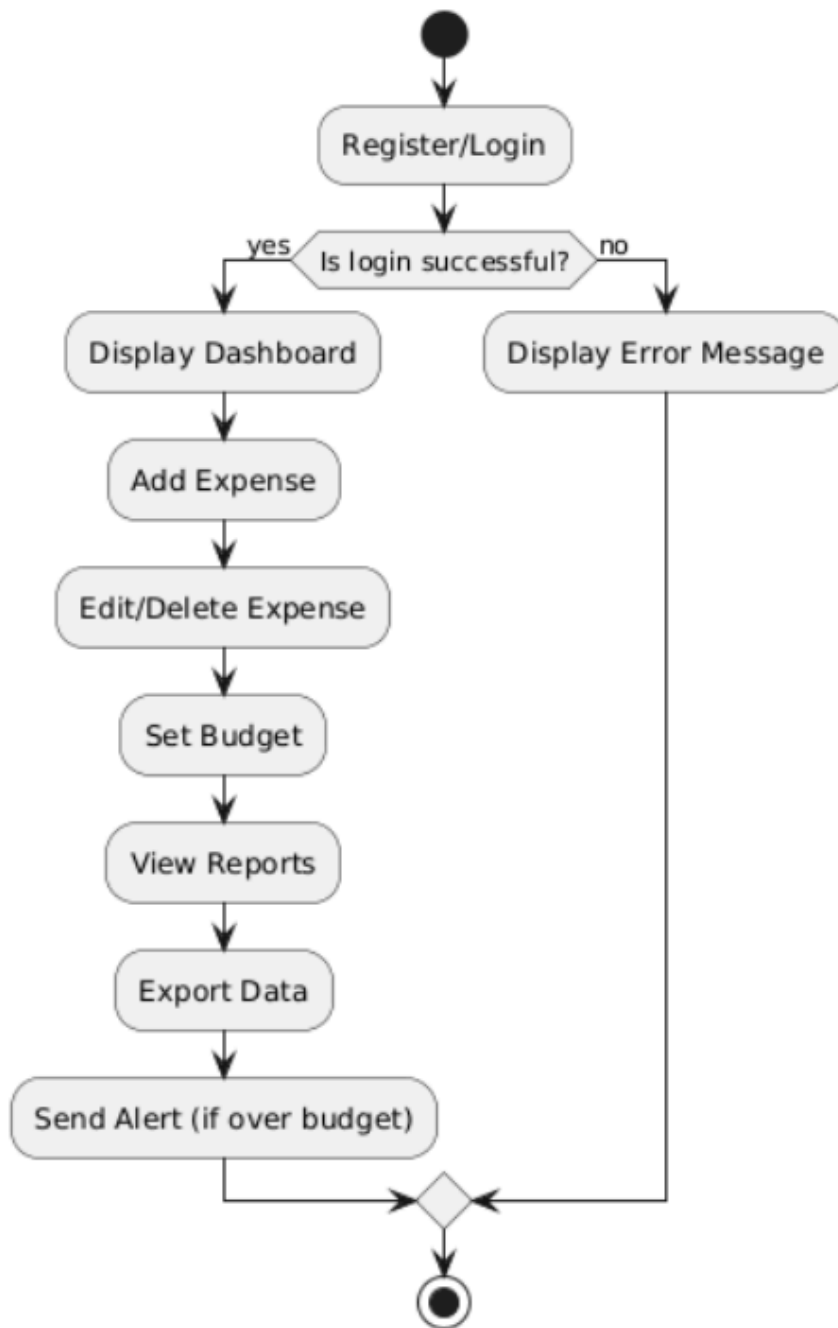
Output:

The Activity Diagram was made successfully.

Experiment No:- 4

OBJECTIVE:-

Identify the classes. Classify them as weak and strong classes and draw the class diagram.

Theory:-

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modelling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity.
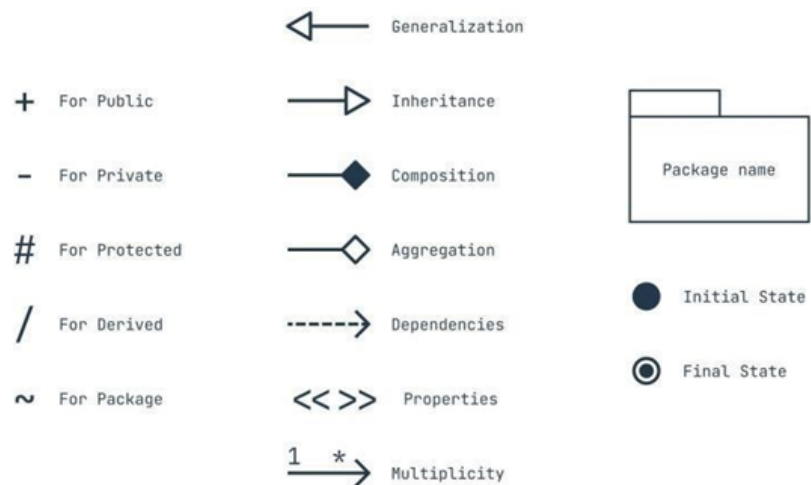
The class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualising, describing and documenting different aspects of a system but also for constructing executable code of the software application.

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modelling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages.The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram.
To specify the visibility of a class member (i.e. any attribute or method), these notations must be placed before the member's name.

+	Public
-	Private
#	Protected
/	Derived (can be combined with one of the others)
~	Package

Notations used:-

+ For Public	◁—— Generalization	
	——▷ Inheritance	Package name
− For Private	——◆ Composition	
# For Protected	——◇ Aggregation	● Initial State
/ For Derived	┄┄▶ Dependencies	◉ Final State
~ For Package	<< >> Properties	
	1 * ——▶ Multiplicity	

Implementation:

Strong Classes:

- User

- Expense

Weak Classes:

- Category

- Budget

- Report

Class Diagram:
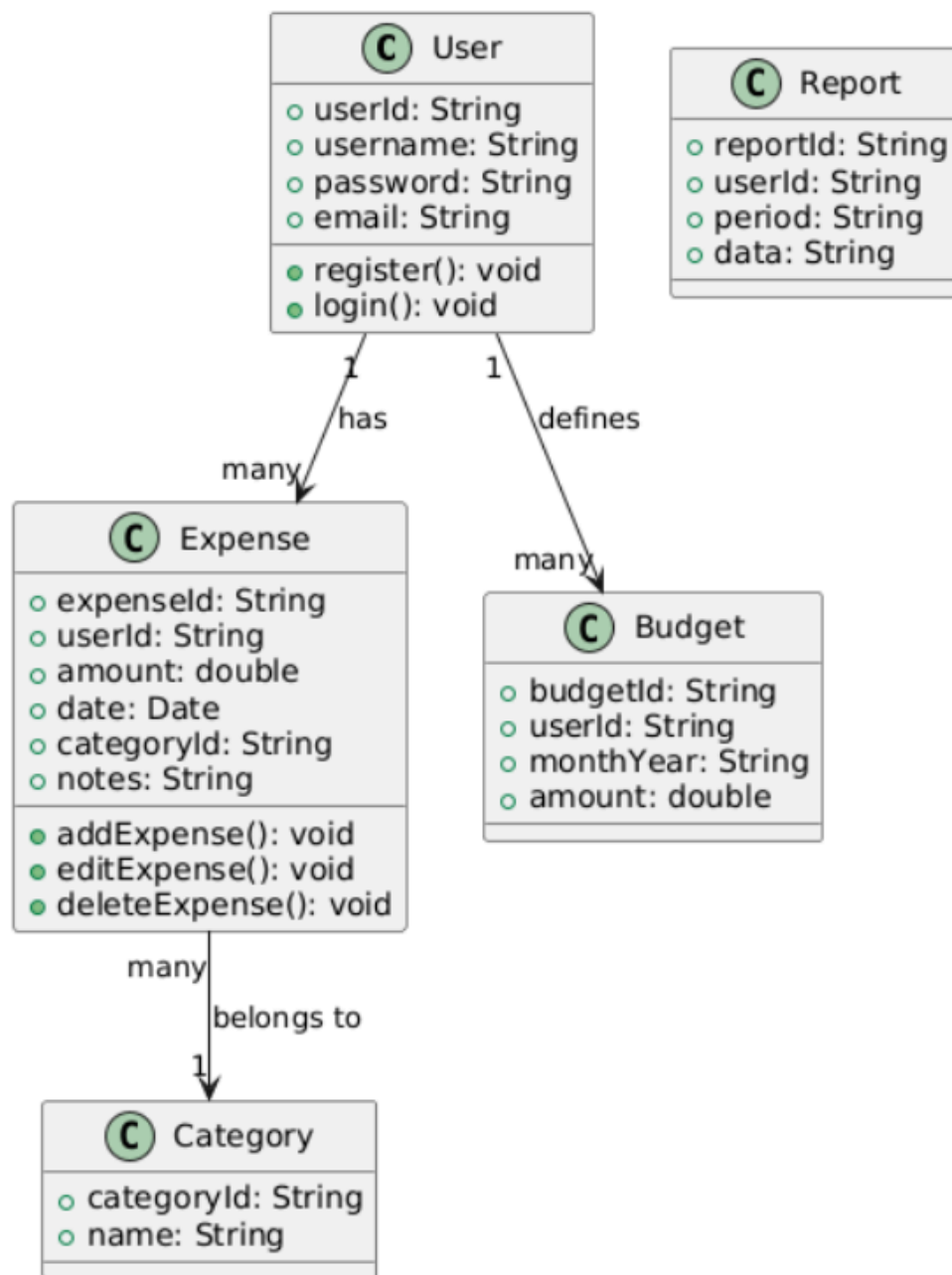
Output:

The Class Diagram was made successfully.

Experiment No:- 5

OBJECTIVE:-

 Draw the sequence diagram for any two scenarios.

Theory:-

Sequence diagram is the most common kind of interaction diagram, which
focuses on the message interchange.
Sequence diagram describes an interaction by focusing on the sequence of messages that are
exchanged, along with their corresponding occurrence specifications on the lifelines.
UML sequence diagrams are used to show how objects interact in a given situation.
An important characteristic of a sequence diagram is that time passes from top to
bottom: the interaction starts near the top of the diagram and ends at the bottom.
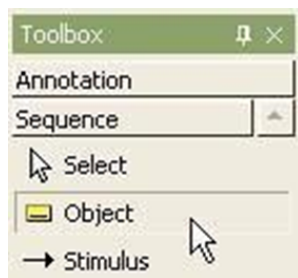
Implementation:

There are many elements in a sequence diagram, let's see how to create an **Object** using
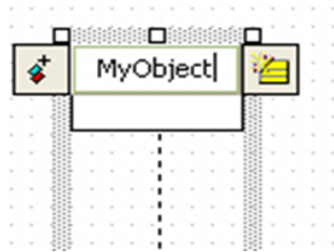StarUML.

Object:

Procedure for creating object:

In order to create object,
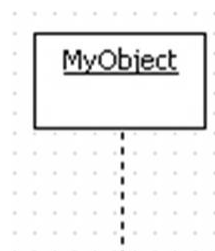   • Click [Toolbox] -> [Sequence] -> [Object] button.

   • And click at the position where object will be placed in the [main window]. Object
     quick dialog is shown. At the quick dialog, enter the object name.
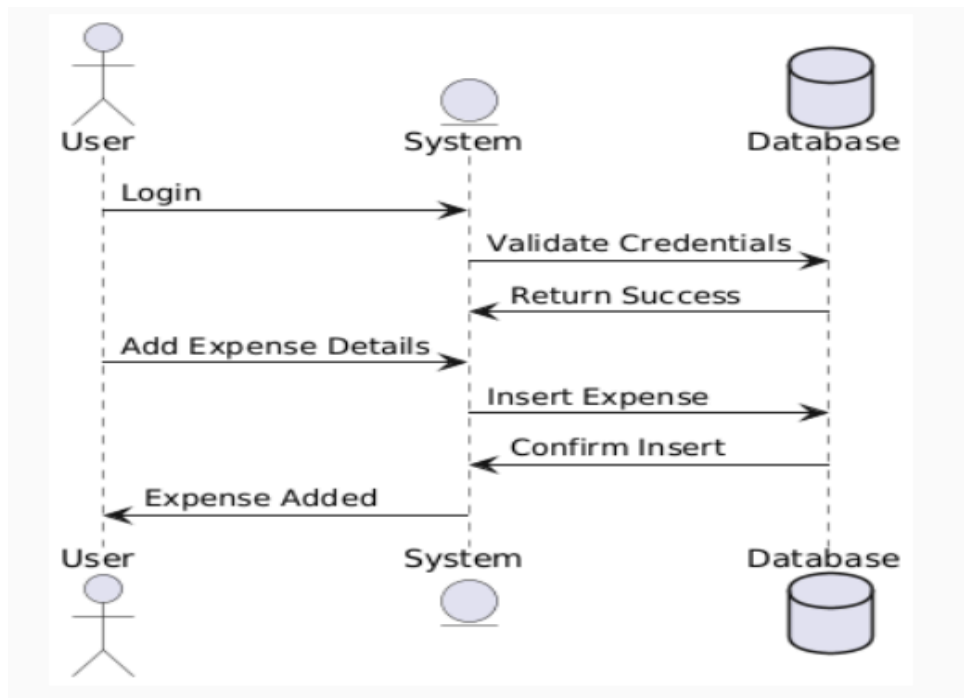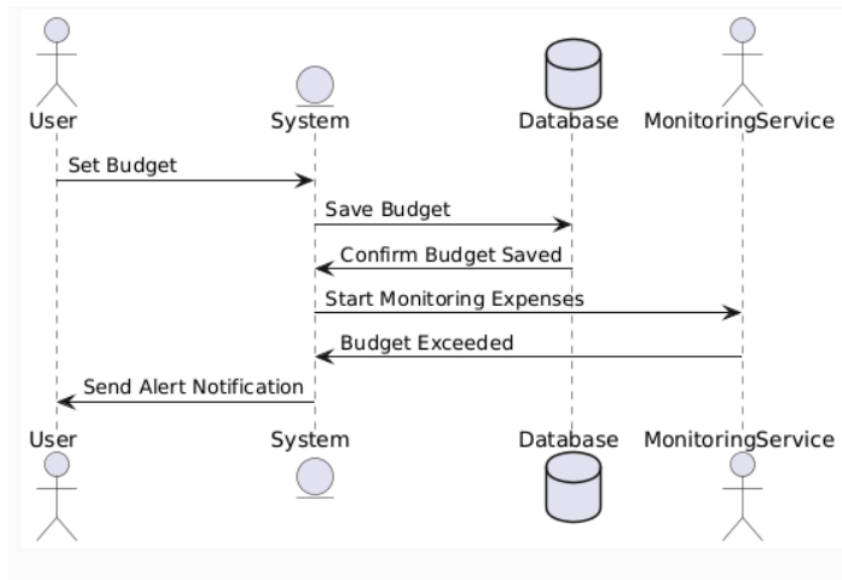
- Press [Enter] key.

Implementation:

Scenario 1: User Adds an Expense

Scenario 2: User Sets Budget and Receives Alert

Output:

The Sequence Diagrams were made successfully.

Experiment No:- 6

**Objective:-** Draw the sequence diagram for any two scenarios.

Theory:-

The second interaction diagram is collaboration diagram. It shows the object organization as shown below. In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. A collaboration diagram is also called a communication diagram or interaction diagram.

The method calls are similar to that of a sequence diagram. Sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.
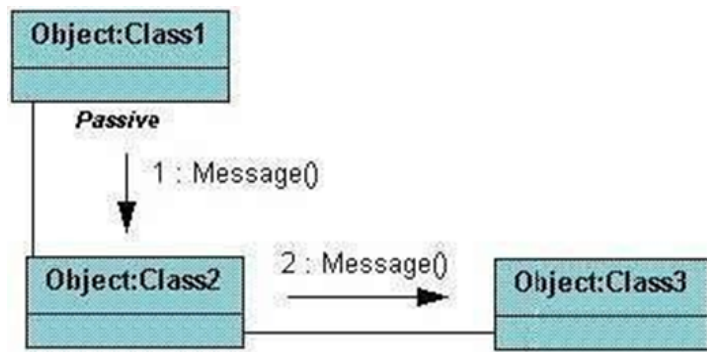
Notations of a Collaboration Diagram:

Components of a collaboration diagram
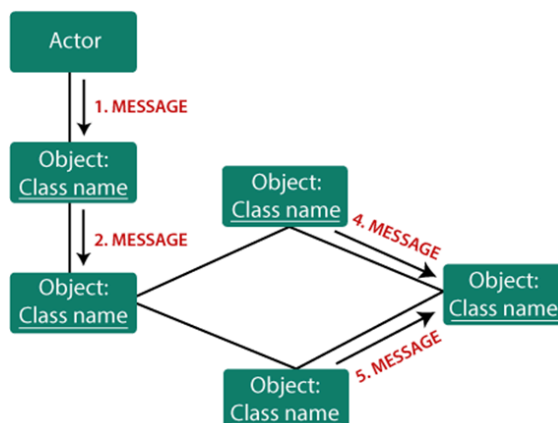
Following are the components of a component diagram that are enlisted below:
Objects: The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.

In the collaboration diagram, objects are utilized in the following ways:

- The object is represented by specifying their name and class.

- It is not mandatory for every class to appear.
- A class may constitute more than one object.
- In the collaboration diagram, firstly, the object is created, and then its class is specified.
- To differentiate one object from another object, it is necessary to name them.

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.

Implementation:

Collaboration Diagram for Expense Management System:

- **User** interacts with **System**

- **System** saves data to **Database**

- **System** sends alerts to **User** when budget exceeds
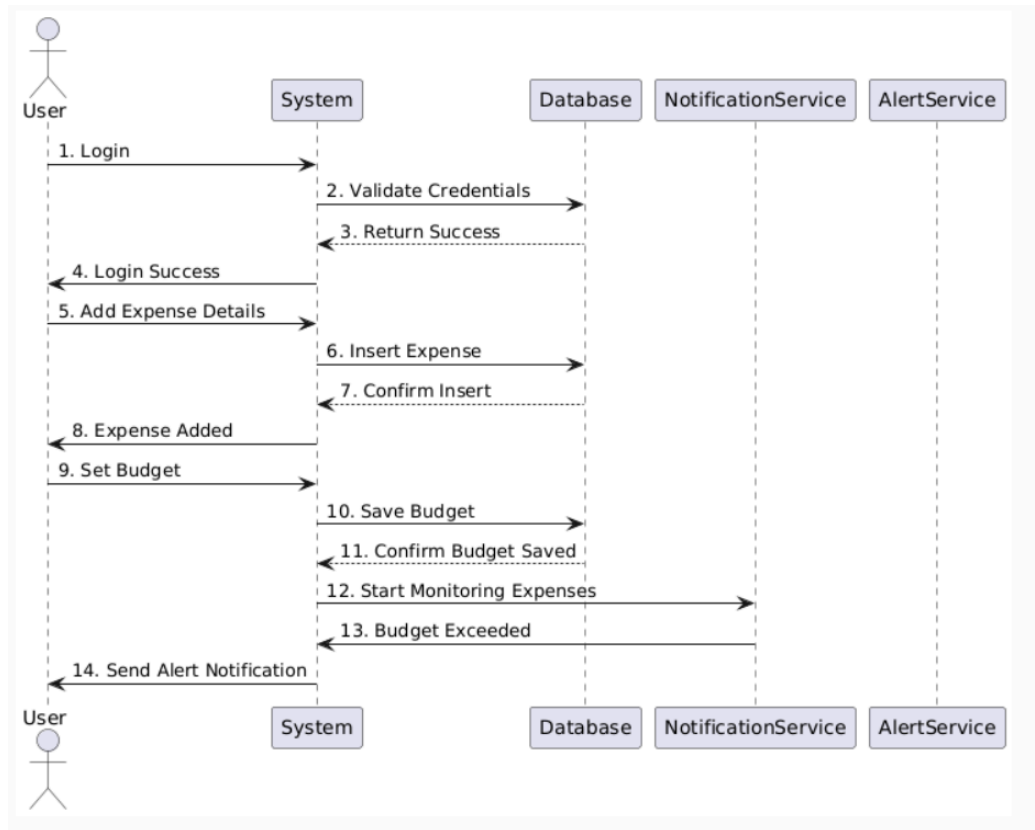
Output:

The Collaboration Diagram was made successfully.

Experiment No:- 7

OBJECTIVE:

Draw the state chart diagram.

Theory:-

State chart diagrams are used for describing the states. States can be identified as the condition of objects when a particular event occurs.

Before drawing a Statechart diagram we should:

- Identify important objects to be analyzed.
- Identify the states.
- Identify the events.

During the life cycle of an object (here order object) it goes through the following states and there may be some abnormal exists also. This abnormal exit may occur due to some problem in the system. When the entire life cycle is complete it is considered as the complete transaction as mentioned below.

Notation of a State Machine Diagram:

Following are the notations of a state machine diagram enlisted below:
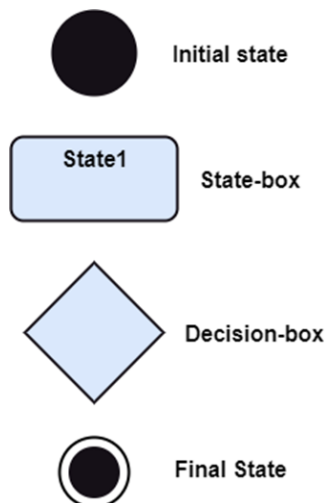
Initial state: It defines the initial state (beginning) of a system, and it is represented by a black filled circle.

Final state: It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.

Decision box: It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.

Transition: A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change

has ensued.

State box: It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

## Types of State

The UML consist of three states:

Simple state: It does not constitute any substructure.

Composite state: It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.

Submachine state: The submachine state is semantically identical to the composite state, but it can be reused.

Implementation:

State Chart Diagram for User Activity:
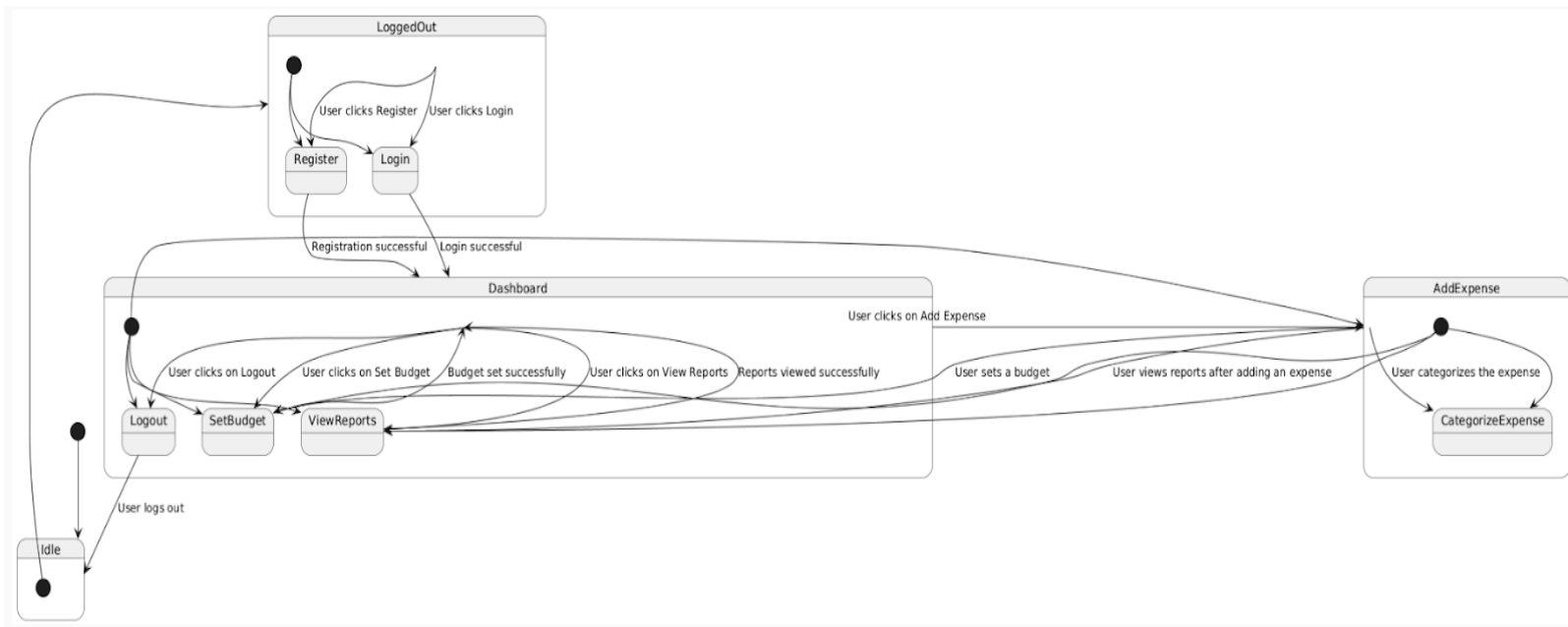
Output:

The State Chart Diagram was made successfully.

Experiment No:- 8

OBJECTIVE:

Draw the component diagram.

Theory:-

Component diagram shows components, provided and required interfaces, ports, and relationships between them. This type of diagrams is used in Component-Based Development (CBD) to describe systems with Service-Oriented Architecture (SOA).

Components in UML could represent

✓ **Logical components** (e.g., business components, process components), and

✓ **Physical components** (e.g., CORBA components, EJB components, COM+ and
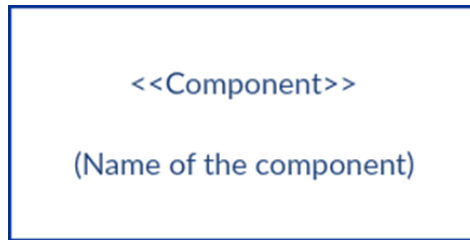
.NET components, WSDL components, etc.),

Notation of a Component Diagram:

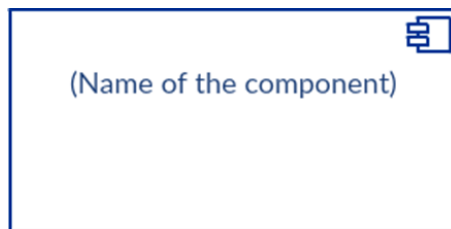### Component

There are three ways the component symbol can be used.
1) Rectangle with the component stereotype (the text <<component>>). The component
stereotype is usually used above the component name to avoid confusing the shape with a class icon.
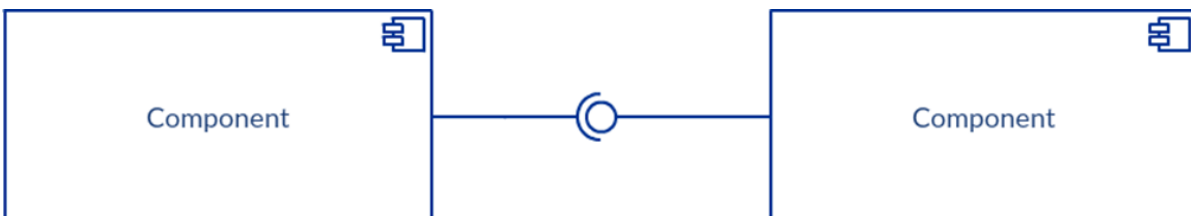
<<Component>>

(Name of the component)

2) Rectangle with the component icon in the top right corner and the name of the component.

(Name of the component)

3) Rectangle with the component icon and the component stereotype.

<<Component>>

Provided Interface and the Required Interface

Component

Component

Interfaces in component diagrams show how components are wired together and interact with each other. The assembly connector allows linking the component's required interface (represented with a semi-circle and a solid line) with the provided interface (represented with a circle and solid line) of another component. This shows that one component is providing the service that the other is requiring.

Port
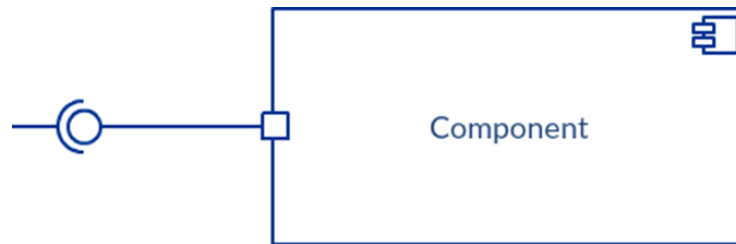
Port (represented by the small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.

Dependencies

Although we can show more detail about the relationship between two components using the ball-and- socket notation (provided interface and required interface), we can just as well use a dependency arrow to show the relationship between two components.

Implementation:

Component Diagram for Expense Management System:

- **Web Interface**: Provides UI to users

- **Backend Server**: Processes requests

- **Database**: Stores expenses, budgets, users

- **Notification Service**: Sends budget alerts
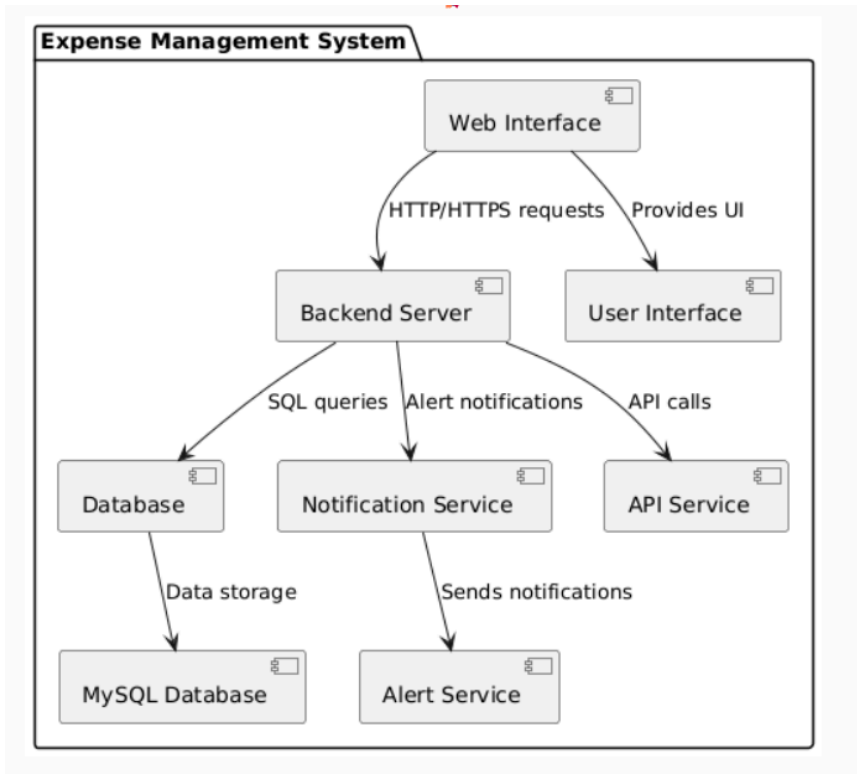
Output:

The Component Diagram was made successfully.

Experiment No:- 9

OBJECTIVE:

Perform forward engineering in JAVA. (Model to code conversion)

Theory:

Forward Engineering is the process of creating a new software product from scratch based on requirements and specifications. It typically involves creating UML diagrams, designing classes, and implementing the code in a programming language.

CLASSES:

1. Expense

2. ExpenseTracker Java Code for Expense.java:

Java Code for Expense.java

```
public class Expense {
private String expenseId;
private String description;
 private double amount;
 private String date;

public Expense(String expenseId, String description, double amount, String date) {
this.expenseId = expenseId;
this.description = description;
 this.amount = amount; this.date = date;
}
 public String getExpenseId()
{
return expenseId;
 }
 public void setExpenseId(String expenseId)
 {
this.expenseId = expenseId;
 }
 public String getDescription()
{
```

```java
 return description;
 }
his.description = description;
 }
 public double getAmount()
 {
 return amount;
 }
 public void setAmount(double amount)
 {
 this.amount = amount;
}
 public String getDate()
 {
 return date;
 }
public void setDate(String date)
 {
 this.date = date;
}
}
```

Java Code for ExpenseTracker.java:

```java
import java.util.ArrayList;
 import java.util.List;

public class ExpenseTracker
 {
private List expenses; public ExpenseTracker()
 {
 this.expenses = new ArrayList<>();
 }
 public void addExpense(Expense e)
 {
 expenses.add(e);
 }
 public void removeExpense(Expense e)
{
 expenses.remove(e);
}
 public List getExpenses()
 {
 return expenses;
```

```
}
 }
```

CONCLUSION: Forward engineering completed for the Expense Management System.

Experiment No:- 10

OBJECTIVE:

Perform reverse engineering in JAVA. (Code to Model conversion)

Theory:

Reverse Engineering is the process of analyzing and understanding the design, structure, and functionality of a product or system by examining its final implementation and reconstructing the underlying architecture or components.

CLASSES:

1. Expense

2. ExpenseTracker

Java Code for Expense.java:

```java
public class Expense
{

private String expenseId;
private String description;
private double amount;
private String date;

public Expense(String expenseId, String description, double amount, String date)
{
this.expenseId = expenseId;
this.description = description;
this.amount = amount;
this.date = date;
}
public String getExpenseId()
{
return expenseId;
}
public void setExpenseId(String expenseId)
{
```

```java
    this.expenseId = expenseId;
}
public String getDescription()
{
 return description;
}
 public void setDescription(String description)
 {
 this.description = description;
 }
public double getAmount()
{
 return amount;
}
public void setAmount(double amount)
{
 this.amount = amount;
}
 public String getDate()
 {
 return date;
 }
 public void setDate(String date)
 {
 this.date = date;
 }
 }
```

Java Code for ExpenseTracker.java:

```java
import java.util.ArrayList;
 import java.util.List;
 public class ExpenseTracker
 {
 private List expenses;
 public ExpenseTracker()
 {
 this.expenses = new ArrayList<>();
 }
 public void addExpense(Expense e)
 { expenses.add(e);
 }
 public void removeExpense(Expense e)
 {
```

```
 expenses.remove(e);
 }
 public List getExpenses()
{
 return expenses;
 }
 }
```

CONCLUSION: Reverse engineering completed for the Expense Management
System.

Experiment No:- 11

OBJECTIVE:

Draw the deployment diagram.

Hardware Interfaces

- Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM Screen resolution of at least 800 x 600 required for proper Hardware Interfaces
- Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM
- Screen resolution of at least 800 x 600 required for proper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows98/2000/XP/NT)
- Web Browser

Theory:-

A Deployment Diagram shows how the software design turns into the actual physical system where the software will run. They show where software components are placed on hardware devices and shows how they connect with each other. This diagram helps visualize how the software will operate across different devices Key elements of a Deployment Diagram • Nodes: These represent the physical hardware entities where software components are deployed, such as servers, workstations, routers, etc. • Components: Represent software modules or artifacts that are deployed onto nodes, including execuA Deployment Diagram shows how the software design turns into the actual physical system where the software will run. They show where software components are placed on hardware devices and shows how they connect with each other. This diagram helps visualize how the software will operate across different devices
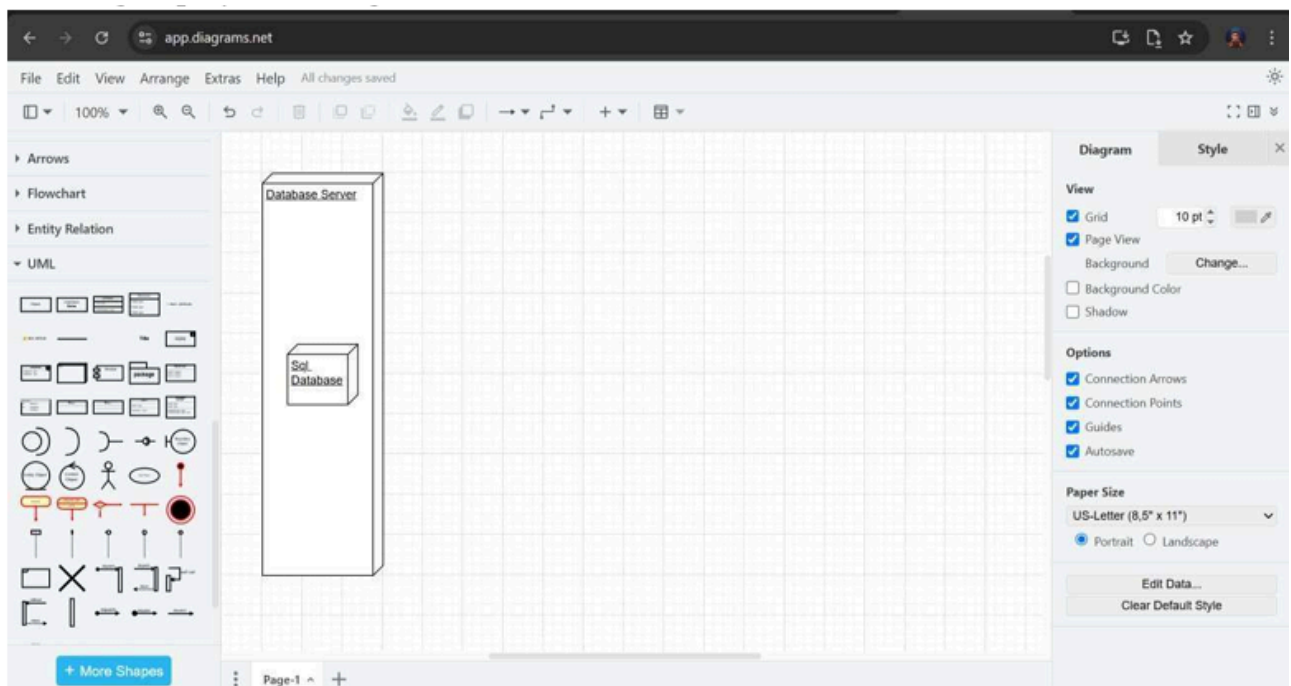
Key elements of a Deployment Diagram

• Nodes: These represent the physical hardware entities where software components are deployed, such as servers, workstations, routers, etc.

• Components: Represent software modules or artifacts that are deployed onto nodes,

including executable files, libraries, databases, and configuration files.

• Artifacts: Physical files that are placed on nodes represent the actual implementation of software components. These can include executable files, scripts, databases, and more.

• Dependencies: These show the relationships or connections between nodes and components, highlighting communication paths, deployment constraints, and other dependencies.

• Associations: Show relationships between nodes and components, signifying that a component is deployed on a particular node, thus mapping software components to physical nodes.

• Deployment Specification: This outlines the setup and characteristics of nodes and components, including hardware specifications, software settings, and communication protocols.

• Communication Paths: Represent channels or connections facilitating communication between nodes and components and includes network connections, communication protocols, etc
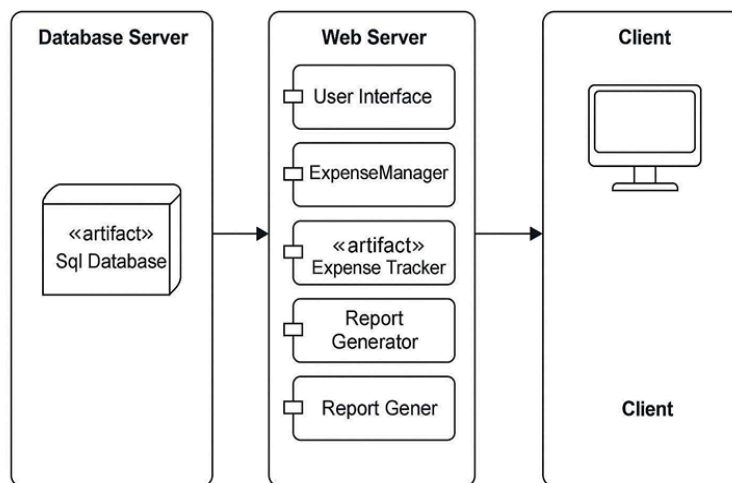
Creating Deployment Diagram:

Implementation:

Deployment Diagram:

Nodes and Artifacts

1. **Client Device (PC / Mobile)**
   Runs the **Web Browser (UI)** used to interact with the system.

2. **Web Server**
   Hosts the **Application Backend** that handles business logic and request routing.

3. **Database Server**
   Hosts the **MySQL Database** used for data storage and retrieval.

Output: The Deployment Diagram was made successfully.