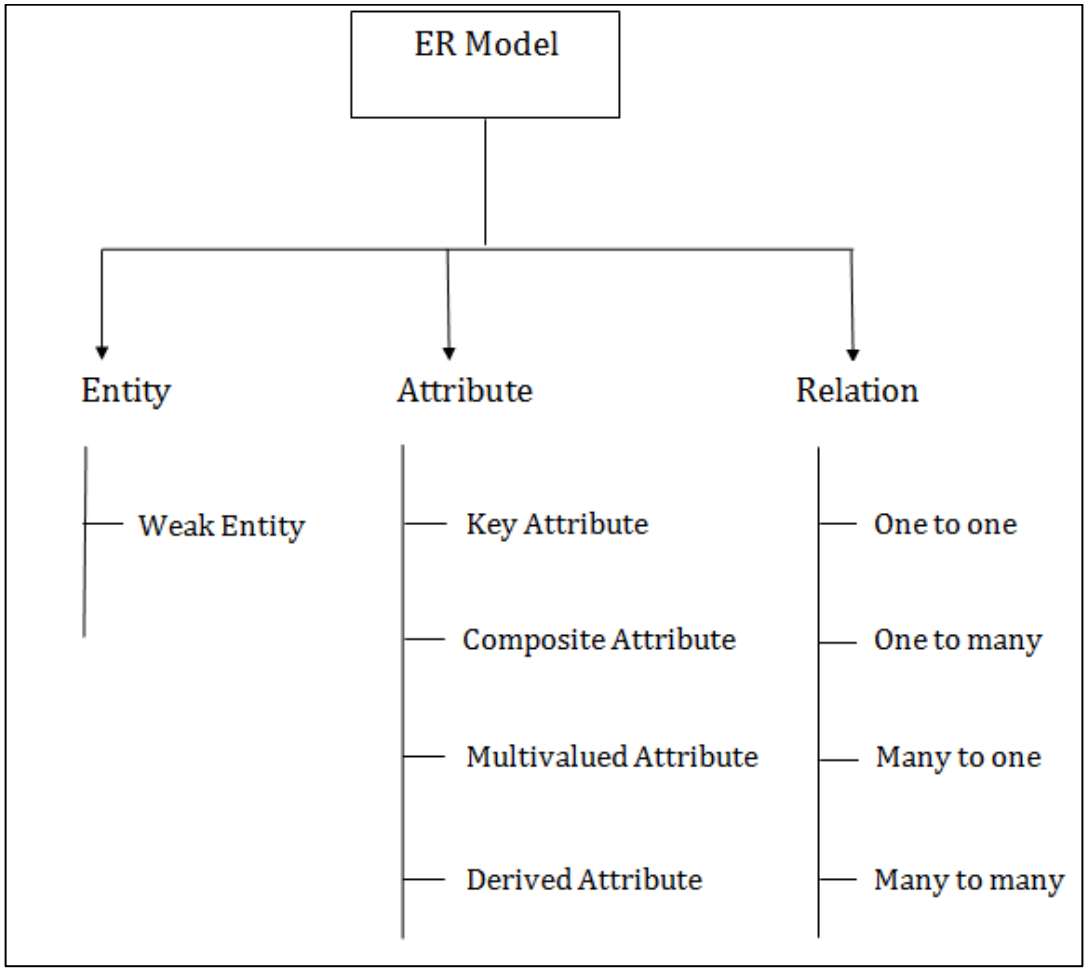
EXPERIMENT-1

AIM: To draw Entity-Relationship diagram for University Employee Management System.

INTRODUCTION TO THE DIAGRAM:

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

Components of ER-Diagram:



INTRODUCTION TO THE PROBLEM STATEMENT:

For the given problem statement, we need to consider the following points:

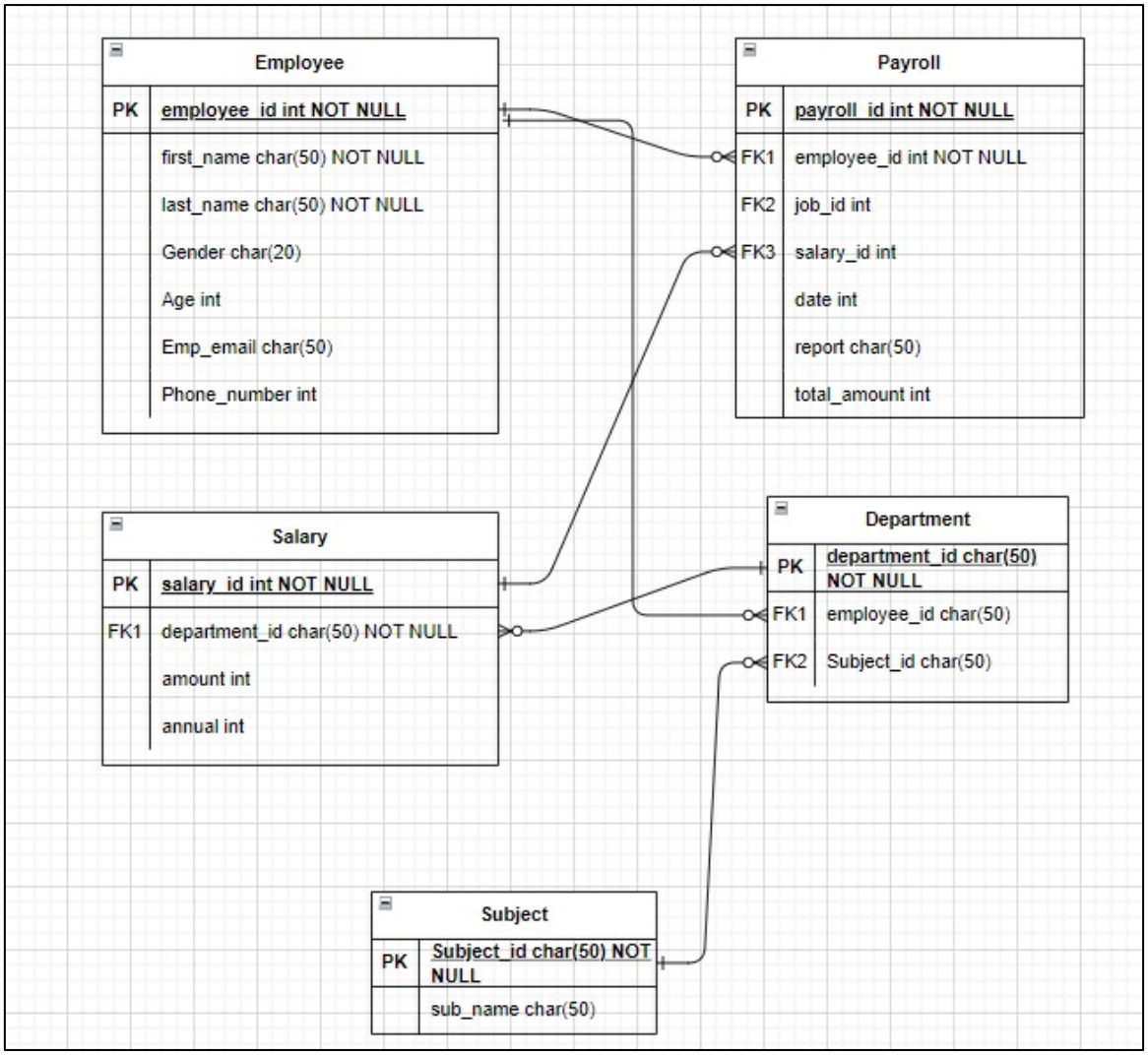
a) Entities:

1. **Employee:** It represents the staff members who work in the university. Attributes include employee id (Primary Key), first name, last name, Gender, Age, Employee email, Phone Number.
2. **Payroll:** It represents the salary based on the number of leaves taken. Attributes include payroll id (Primary Key), employee id (Foreign Key), job id (Foreign Key), salary id (Foreign Key), date, report, total amount.
3. **Salary:** It represents the actual salary paid to the employees. Attributes include salary id (Primary Key), department id (Foreign Key), amount, annual.
4. **Department:** It represents the department to which the employee belongs. Attributes include department id (Primary Key), employee id (Foreign Key), subject id (Foreign Key).
5. **Subject:** It represents the subjects that are taught by the faculties. Attributes include subject id (Primary Key), subject name.

b) Relationships:

1. **Works In:** An employee works in a department. This is a one-to-many relationship from Employee to Department.
2. **Has:** A Department has subjects. This is a one-to-many relationship from Department to Subject.
3. **Teach:** An employee teaches a subject. This is a many-to-many relationship between Employee and Subject.
4. **Earns:** Each employee earns a salary. This is a one-to-many relationship between Employee and Salary.

ER-DIAGRAM:

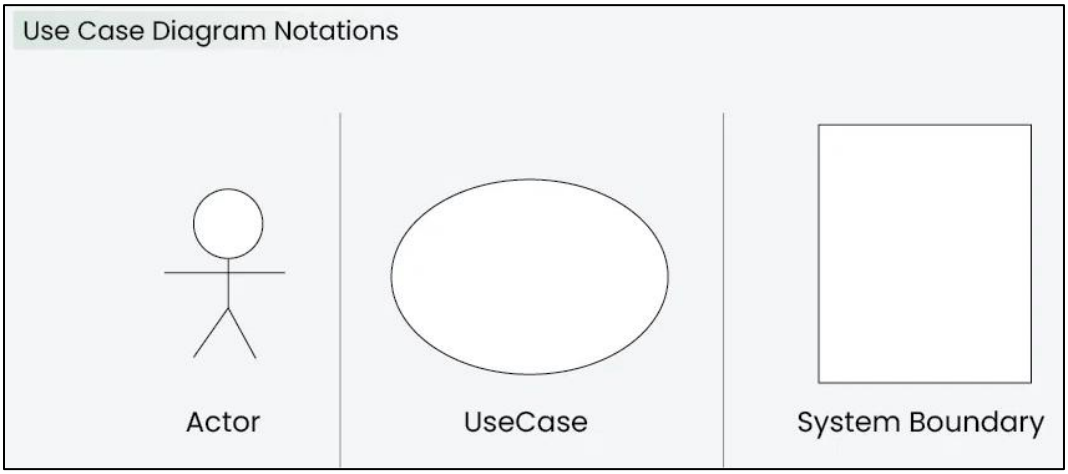

Employee
PK
employee_id int NOT NULL
first_name char(50) NOT NULL
last_name char(50) NOT NULL
Gender char(20)
Age int
Emp_email char(50)
Phone_number int
Payroll
PK
payroll_id int NOT NULL
FK1
employee_id int NOT NULL
FK2
job_id int
FK3
salary_id int
date int
report char(50)
total_amount int
Salary
PK
salary_id int NOT NULL
FK1
department_id char(50) NOT NULL
amount int
annual int
Department
PK
department_id char(50) NOT NULL
FK1
employee_id char(50)
FK2
Subject_id char(50)
Subject
PK
Subject_id char(50) NOT NULL
sub_name char(50)

EXPERIMENT-2

AIM: To draw Use Case diagram for Smart Agriculture Management System.

INTRODUCTION TO THE DIAGRAM: A Use Case Diagram is a type of Unified Modeling Language (UML) diagram that represents the interaction between actors (users or external systems) and a system under consideration to accomplish specific goals. It provides a high-level view of the system's functionality by illustrating the various ways users can interact with it.

Components of Use Case Diagram:



Use Case Diagrams Relationships:

a. Association
b. Include
c. Extend
d. Generalization

INTRODUCTION TO THE PROBLEM STATEMENT: The problem statement consists of the following:

1. **Actors**:

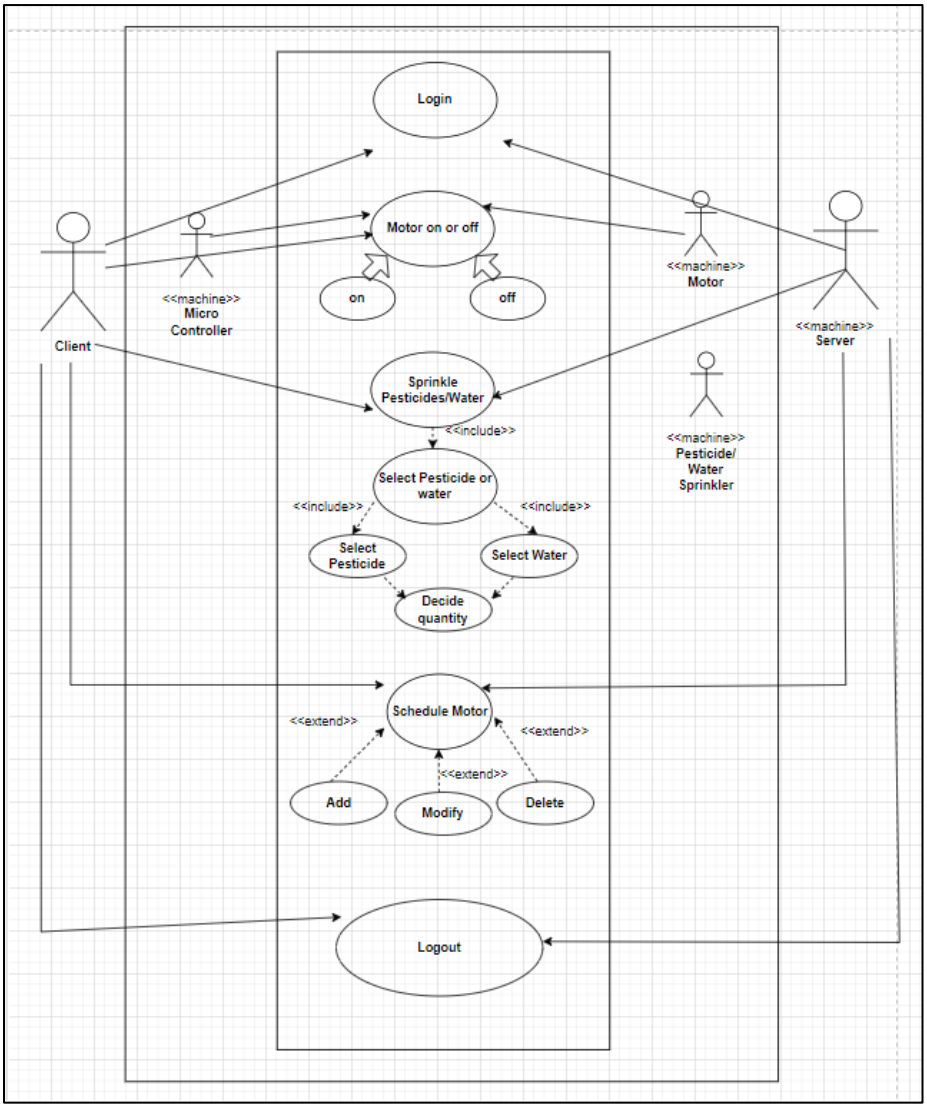
- **Client**: Represents the user interacting with the system. The client initiates actions such as logging in, turning the motor on/off, selecting substances (pesticides/water), deciding quantities, and scheduling the motor.
- **Microcontroller**: Manages motor control and communicates with the motor.
- **Motor**: Performs the sprinkling action (either pesticides or water).

- **Pesticide/Water Sprinkler**: Delivers the chosen substance (pesticides or water).
- **Server**: Likely handles authentication, scheduling, and overall system management.

2. **Relationships**:

- **Client-Microcontroller**: The client interacts with the microcontroller to control the motor.
- **Microcontroller-Motor**: The microcontroller communicates with the motor to turn it on/off and manage sprinkling.
- **Client-Pesticide/Water Sprinkler**: The client selects the substance (pesticides or water) to be sprinkled.
- **Client-Server**: The server handles authentication, scheduling, and overall system coordination.

USE CASE DIAGRAM:



EXPERIMENT-3

AIM: To draw Class Diagram for Pollution Control Management System.

INTRODUCTION TO THE DIAGRAM: Class diagrams are a type of UML (Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems.

- In these diagrams, classes are depicted as boxes, each containing three compartments for the class name, attributes, and methods. Lines connecting classes illustrate associations, showing relationships such as one-to-one or one-to-many.
- Class diagrams provide a high-level overview of a system's design, helping to communicate and document the structure of the software. They are a fundamental tool in object-oriented design and play a crucial role in the software development lifecycle.

Components of Class Diagram:

- **Class Name:** The name of the class is typically written in the top compartment of the class box and is centered and bold.
- **Attributes:** Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.
- **Methods:** Methods, also known as functions or operations, represent the behavior or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.
- **Visibility Notation:** Visibility notations indicate the access level of attributes and methods. Common visibility notations include:
  - + for public (visible to all classes)
  - - for private (visible only within the class)
  - # for protected (visible to subclasses)
  - ~ for package or default visibility (visible to classes in the same package)

INTRODUCTION TO THE PROBLEM STATEMENT: The problem statement consists of the following:

1. **Classes:**

a) **Pollution Management:** This class serves as a controller or manager for the pollution control system. It includes the methods Monitoring Station and Remove station.

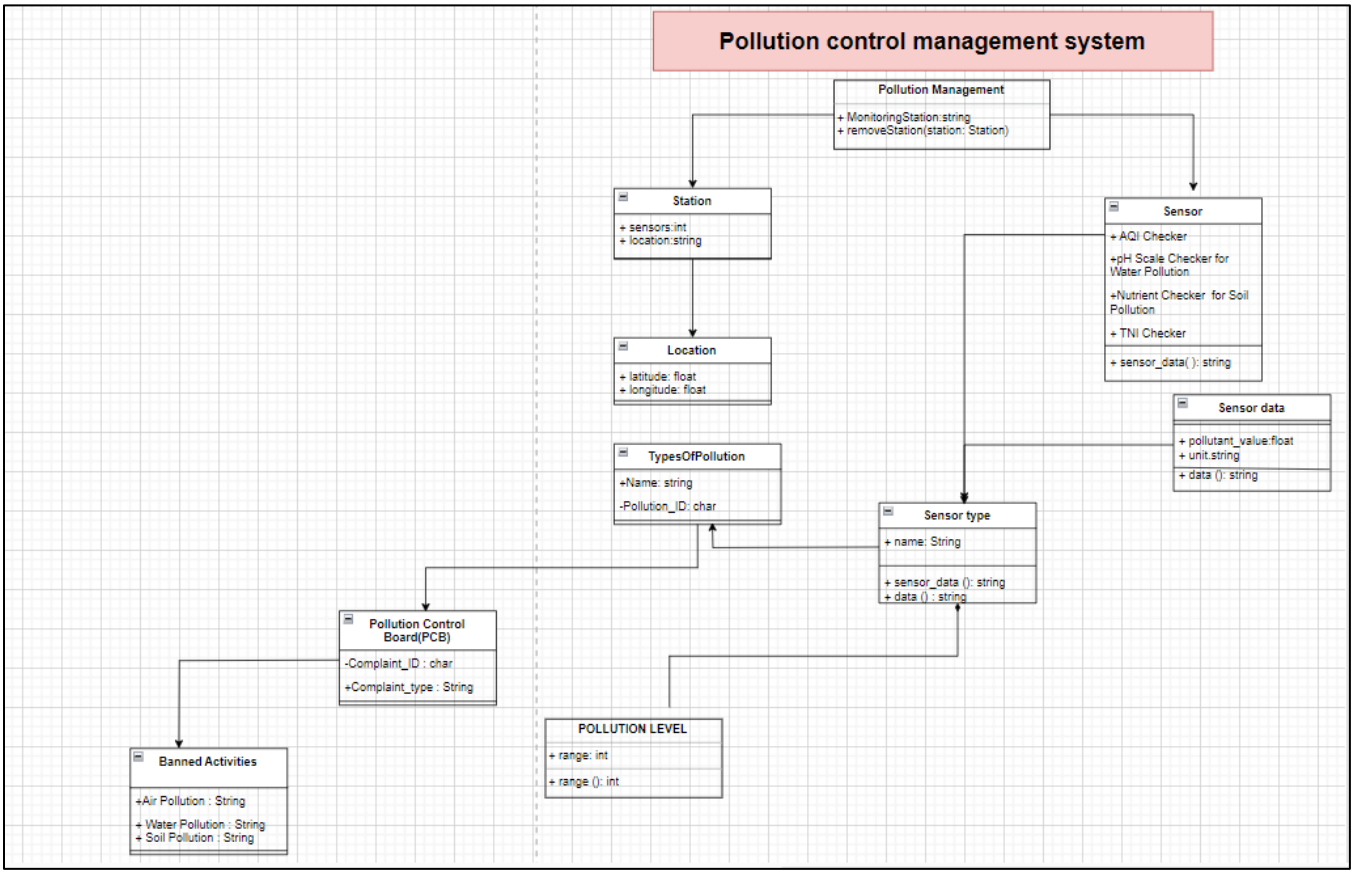
b) **Station:** This class represents the monitoring system in pollution control. It consists of the attribute's sensors and location.

c) **Location:** This class represents the geographical location. It consists of the attribute's latitude and longitude.

d) **Types of Pollution:** This class describes the various types of pollution. It consists of the attribute's name and pollution id.

e) **Sensor:** This class represents a sensor installed at a monitoring station to measure pollution levels. It consists of the attribute's AQI checker, pH, nutrient and TNI checker.
f) **Sensor Data:** This class represents the data collected by a sensor. It consists of the attribute's pollutant value and the unit.
g) **Sensor Type:** This class represents the different types of sensors that are used. It consists of the attributes name and methods sensor_data and data.
h) **PCB:** This class represents the Pollution control Board to file complaints against pollution causing activities. It consists of the attribute's complaint id and complaint type.
i) **Pollution Level:** This class represents the Pollution levels. It consists of the attributes range and the method range that returns the pollution range.
j) **Banned Activities:** This class represents the activities that cause pollution. It consists of the attributes such as activities that cause air, noise, water and soil pollution.

CLASS DIAGRAM:



EXPERIMENT-4

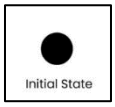
AIM: To draw State Machine Diagram for Automated Parking Management System.

INTRODUCTION TO THE DIAGRAM: A State Machine Diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions.

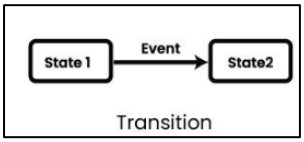
- State Machine diagrams are also referred to as State Machines Diagrams and State-Chart Diagrams.
- These terms are often used interchangeably. So simply, a state machine diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.
- We can say that each class has a state, but we don't model every class using State Machine diagrams.
- We prefer to model the states with three or more states.

Components of a State Machine Diagram:

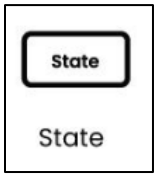
1. **Initial State**: A black filled circle is used to represent the initial state of a System or a Class.



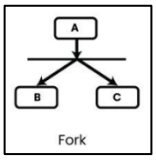
2. **Transition:** A solid arrow is used to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.



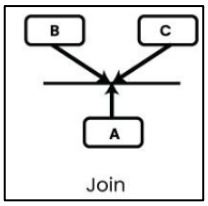
3. **State:** A rounded rectangle is used to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.



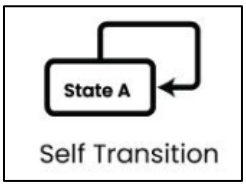
4. **Fork:** The fork notation is used to represent a state splitting into two or more concurrent states.



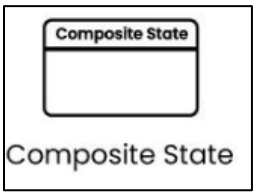
5. **Join:** The join notation is used when two or more states concurrently converge into one on the occurrence of an event or events.



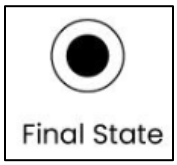
6. **Self-Transition:** It is used in scenarios when the state of the object does not change upon the occurrence of an event.



7. **Composite State:** We represent a state with internal activities using a composite state.



8. **Final State:** A filled circle within a circle notation is used to represent the final state in a state machine diagram.



<u>INTRODUCTION TO THE PROBLEM STATEMENT</u>: The problem statement consists of the following:

1. **Initial State ("Vehicle arrives")**:

- Represents the starting point of the process.

- When a vehicle arrives at the parking area, it enters this state.

2. **State: (“Checking the type of vehicle”)**:
    - This state checks whether the arriving vehicle is a car or a bike.
    - It branches into two paths based on the vehicle type.

3. **Transitions**:
    - If the vehicle is classified as a **car**, it proceeds to the next state.
    - If the vehicle is classified as a **bike**, it also proceeds to the next state.

4. **State: (“Availability of the spot”)**:
    - In this state, the system looks for available parking spots based on the vehicle type.
    - If an available spot is found, it proceeds to the next state.
    - If no available spot is found, it loops back to recheck.

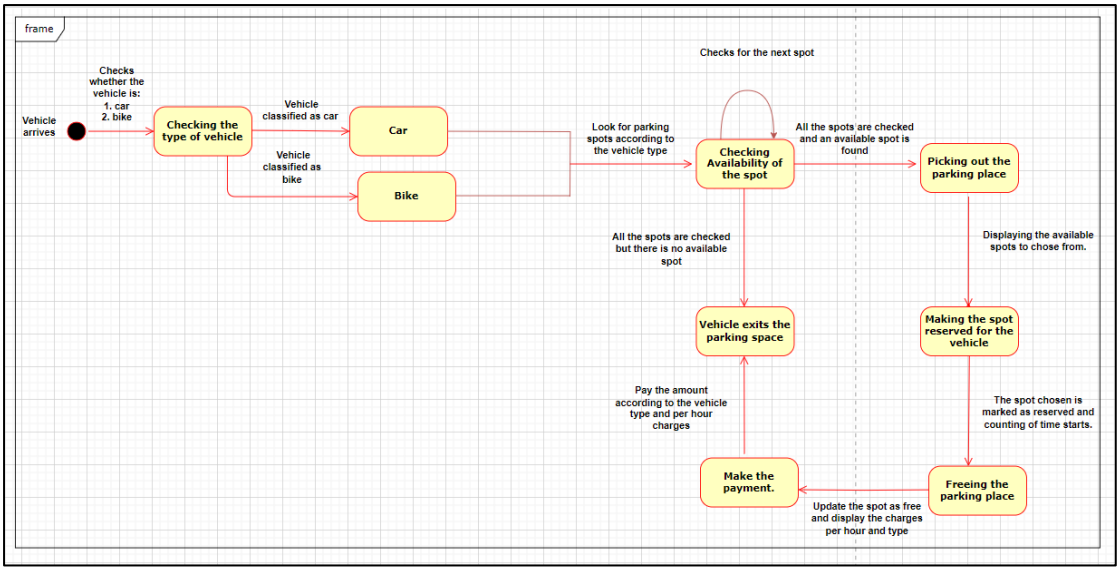
5. **State: (“Picking out the parking place”)**:
    - Once an available spot is identified, the system selects a parking place for the vehicle.

6. **State: (“Making the spot reserved for the vehicle”)**:
    - The chosen spot is marked as reserved for the arriving vehicle.
    - The countdown for parking time starts.

7. **State: (“Make payment”)**:
    - The user makes payment according to the vehicle type and hourly charges.

8. **State: (“Freeing up parking place”)**:
    - After payment, the vehicle owner frees up the parking space.
    - The spot becomes available for other vehicles.

STATE MACHINE DIAGRAM:



EXPERIMENT-5

AIM: To draw Sequence Diagram for Smart Agriculture Management System.

INTRODUCTION TO THE DIAGRAM: UML Sequence Diagrams are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent the time what messages are sent and when.

Sequence Diagrams captures:

- the interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams)
- high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)

Components of the Sequence Diagram:

- Lifeline
- Activations
- Call Message
- Return Message
- Self-Message
- Recursive Message
- Create Message
- Destroy Message
- Duration Message

INTRODUCTION TO THE PROBLEM STATEMENT: The given problem statement consists of the following:

1. **Lifelines**:
    - **Farmer/User**: Represents the individual interacting with the system. Their lifeline initiates actions and communicates with other components.
    - **Server**: Handles requests from the user and processes them.
    - **User Dashboard**: Provides options and actions for the farmer.
    - **Sensor**: Monitors environmental factors.
    - **Motor**: Controls physical processes.
    - **Government Portal for Farmers**: Assists in market research.

2. **Interactions**:

- **Login/Create Account Interaction**:
  - **Farmer/User** sends a request to the **Server** to log in or create an account.
  - The **Server** processes the request, validates credentials, and responds with account status.

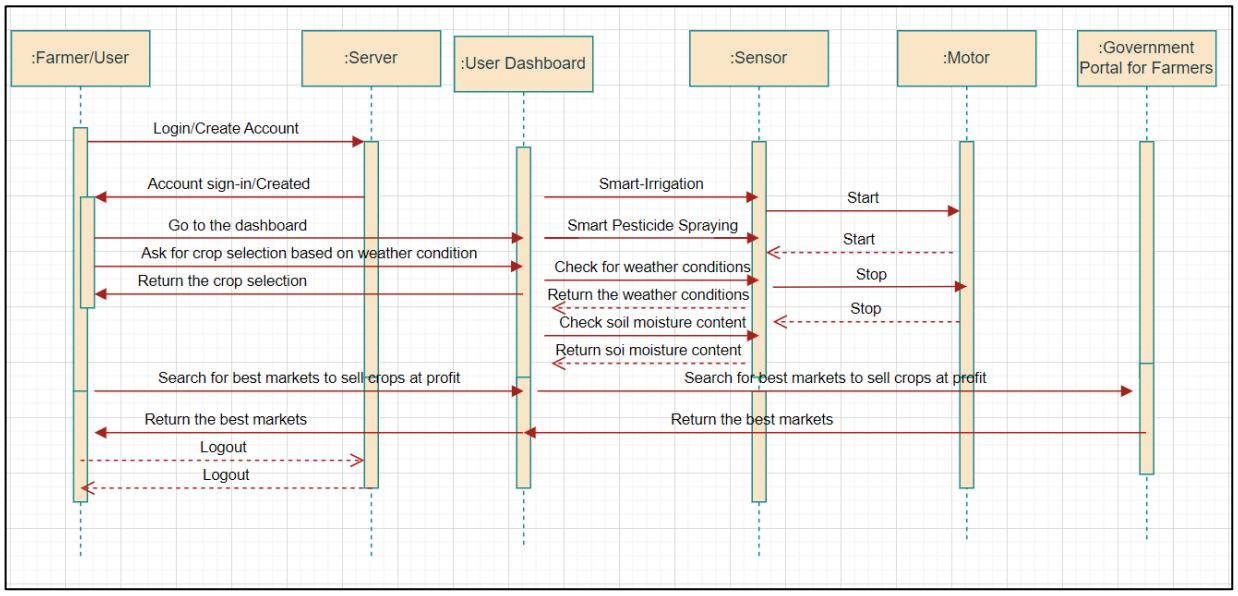
- **Crop Selection Interaction**:
  - **Farmer/User** interacts with the **User Dashboard** to select crops based on weather conditions.
  - The **User Dashboard** communicates with the **Sensor** to check weather conditions.
  - The **Sensor** provides weather information back to the **User Dashboard**.

- **Market Search Interaction**:
  - **Farmer/User** requests the **User Dashboard** to search for the best markets to sell crops.
  - The **User Dashboard** communicates with the **Government Portal for Farmers**.
  - The **Government Portal** responds with market information.

- **Smart Irrigation Interaction**:
  - **User Dashboard** initiates smart irrigation.
  - The **Sensor** monitors soil moisture content.
  - The **Motor** controls irrigation based on sensor inputs.

- **Smart Pesticide Spraying Interaction**:
  - **User Dashboard** activates smart pesticide spraying.
  - The **Sensor** checks weather conditions.
  - The **Motor** controls pesticide spraying based on sensor data.

SEQUENCE DIAGRAM:



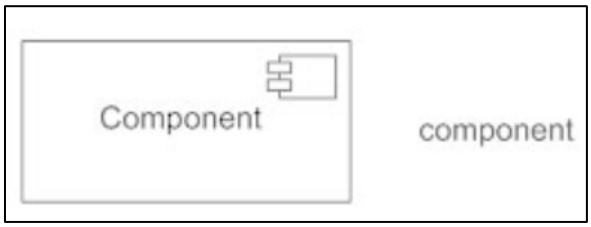
EXPERIMENT-6

AIM: To draw Component Diagram for Smart Agriculture Management System.

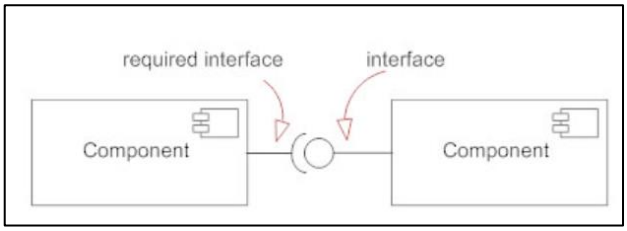
INTRODUCTION TO THE DIAGRAM: A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by planned development.

Basic Component Diagram Symbols and Notations:

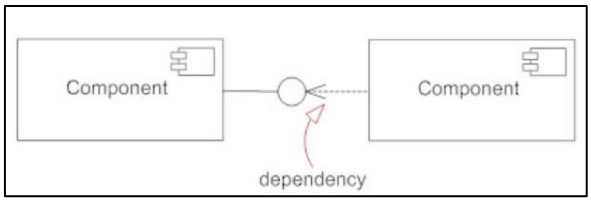
a) **Component:** A component is a logical unit block of the system, a slightly higher abstraction than classes. It is represented as a rectangle with a smaller rectangle in the upper right corner with tabs or the word written above the name of the component to help distinguish it from a class.



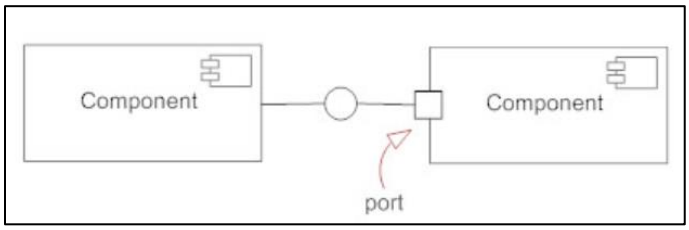
b) **Interface:** An interface (small circle or semi-circle on a stick) describes a group of operations used (required) or created (provided) by components. A full circle represents an interface created or provided by the component. A semi-circle represents a required interface, like a person's input.



c) **Dependencies:** Draw dependencies among components using dashed arrows.



d) **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.



INTRODUCTION TO THE PROBLEM STATEMENT: The problem statement consists of the following:

1. **Components**:
    - **User/Farmer**: Represents the end users (farmers) interacting with the system.
    - **Smart Agriculture System**: The central system that encompasses all other components.
    - **Server**: Handles user authentication and processes requests.
    - **User Dashboard**: The interface for users post-login or sign-up.
    - **Data Access**: Represents different data access points within the system.
    - **Sensor**: Likely collects data related to weather, soil conditions, etc.
    - **Motor**: Controls physical processes (e.g., irrigation).
    - **Government Portal for Farmers**: An external component providing information.

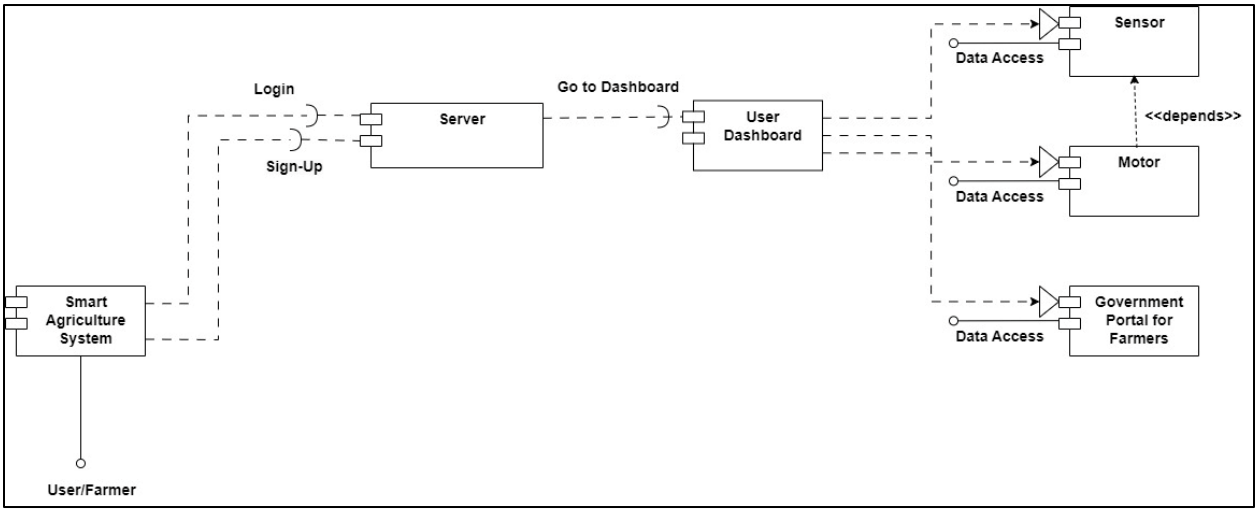
2. **Interfaces**:
    - **User/Farmer Interface**:
        - Provides interaction capabilities for end users.
        - Includes functionalities like login and sign-up.
    - **Server Interface**:
        - Handles communication between the User/Farmer and the system.
        - Offers methods for login, sign-up, and data retrieval.
    - **User Dashboard Interface**:
        - Allows users to navigate the system.
        - Provides access to functionalities and data.
    - **Data Access Interfaces**:
        - Connects to the Sensor, Motor, and Government Portal for Farmers.

- Facilitates data exchange.

- **Sensor Interface**:
  - Collects weather and soil data.
  - Communicates with the Data Access component.
- **Motor Interface**:
  - Controls physical processes (e.g., irrigation).
  - Linked to the User Dashboard.
- **Government Portal Interface**:
  - External interface accessible via the User Dashboard.
  - Provides relevant information.

3. **Dependencies and Relationships**:
   - **Sensor Dependency on Data Access**:
     - Indicates that the Sensor relies on Data Access for data retrieval.
   - **Motor Association with User Dashboard**:
     - The Motor is associated with the User Dashboard for control.
   - **Government Portal Access via Data Access**:
     - The Government Portal for Farmers is accessed through Data Access.

<u>COMPONENT DIAGRAM</u>:



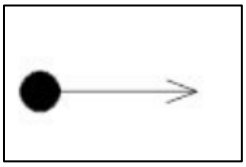
EXPERIMENT-7

AIM: To draw Activity Diagram for University Admission System.

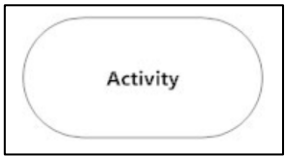
INTRODUCTION TO THE DIAGRAM: An activity diagram visually presents a series of actions or flow of control in a system like a flowchart or a data flow diagram. Activity diagrams are often used in business process modeling. They can also describe the steps in a use case diagram. Activities modeled can be sequential and concurrent. In both cases an activity diagram will have a beginning (an initial state) and an end (a final state).

Basic Activity Diagram Notations and Symbols:

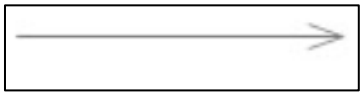
a) **Initial State or Start Point:** A small-filled circle followed by an arrow represents the initial action state or the start point for any activity diagram.



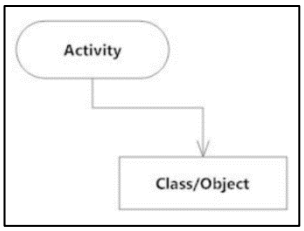
b) **Activity or Action State:** An action state represents the non-interruptible action of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.



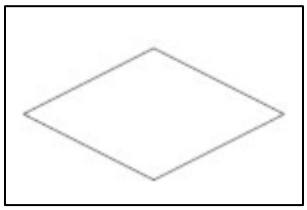
c) **Action Flow:** Action flows, also called edges and paths, illustrate the transitions from one action state to another. They are usually drawn with an arrowed line.



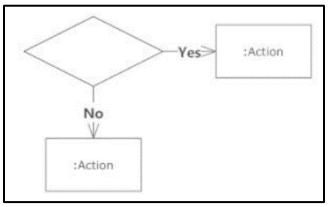
d) **Object Flow:** Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



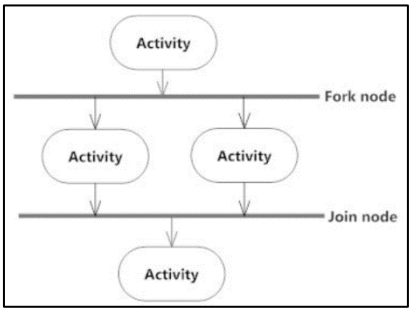
e) **Decisions and Branching:** A diamond represents a decision with alternate paths. When an activity requires a decision prior to moving on to the next activity, add a diamond between the two activities. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



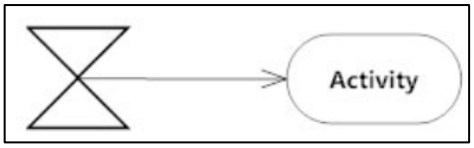
f) **Guards:** In UML, guards are a statement written next to a decision diamond that must be true before moving next to the next activity. These are not essential, but are useful when a specific answer, such as "Yes, three labels are printed," is needed before moving forward.



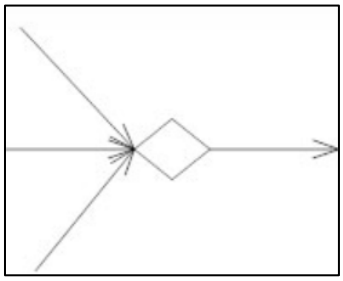
g) **Synchronization:** A fork node is used to split a single incoming flow into multiple concurrent flows. It is represented as a straight, slightly thicker line in an activity diagram. A join node joins multiple concurrent flows back into a single outgoing flow. A fork and join mode used together are often referred to as synchronization.



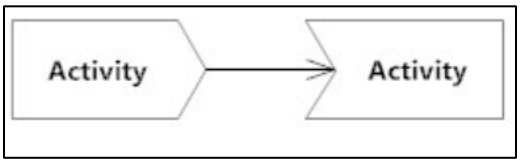
h) **Time Event:** This refers to an event that stops the flow for a time; an hourglass depicts it.



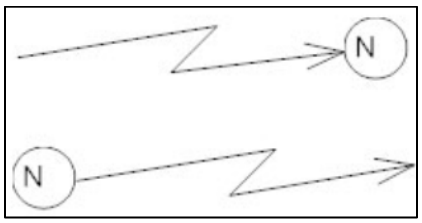
i) **Merge Event:** A merge event brings together multiple flows that are not concurrent.



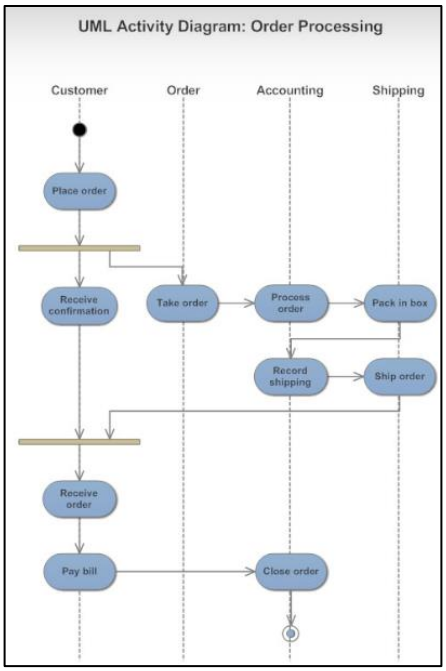
j) **Sent and Received Signals:** Signals represent how activities can be modified from outside the system. They usually appear in pairs of sent and received signals, because the state can't change until a response is received, much like synchronous messages in a sequence diagram. For example, an authorization of payment is needed before an order can be completed.



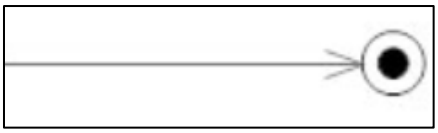
k) **Interrupting Edge:** An event, such as a cancellation, that interrupts the flow denoted with a lightning bolt.



l) **Swimlanes:** Swimlanes group related activities into one column.



m) **Final State or End Point:** An arrow pointing to a filled circle nested inside another circle represents the final action state.



<u>INTRODUCTION TO THE PROBLEM STATEMENT</u>: The problem statement visually represents the flow of actions within the University Admission Management System, guiding users through login, application, and administrative processes.

It consists of the following:

1. **Start Node**: Represents the beginning of the process. It's where the flow starts.

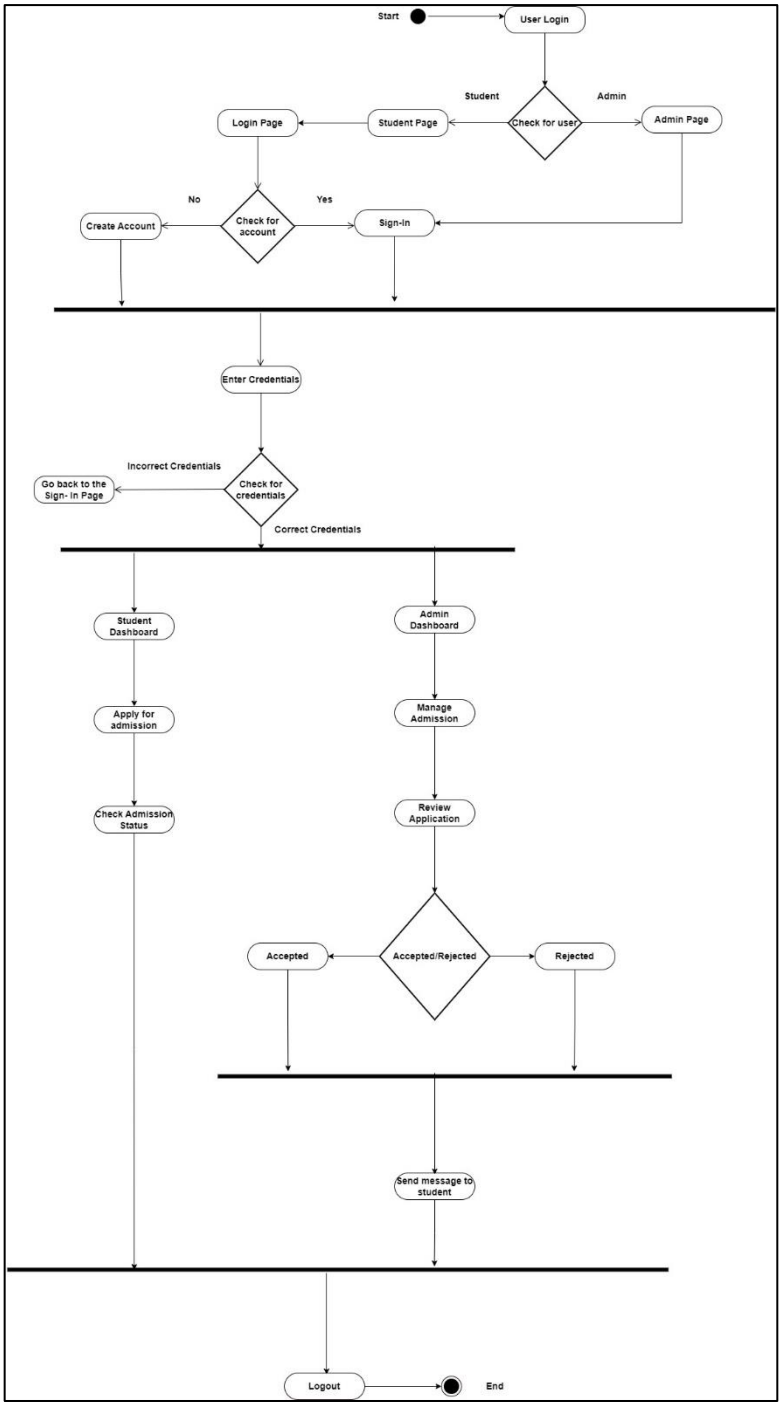
2. **User Login**:
   - This is a decision point where users provide their credentials.
   - Three possible paths emerge:
     - **Student**: If the user is a student, they proceed to the student-related activities.
     - **Admin**: If the user is an admin, they access admin-specific functionalities.

3. **Student Path**:

- After successful login, students reach the “Student Dashboard.”
- Key activities include:
  - **Apply for Admission**: Students submit their applications.
  - **Finish Admission Exam**: If required, students complete any necessary assessments.
- The application outcome is either **Accepted** or **Rejected**.
- If accepted, a message is sent to the student, and they can log out.

4. **Admin Path**:
   - Admins log in and access the “Admin Dashboard.”
   - Their responsibilities involve:
     - **Manage Admissions**: Handling the overall admission process.
     - **Review Applications**: Evaluating student applications.
   - The final decision on application status (Accepted/Rejected) lies with the admin.

ACTIVITY DIAGRAM:



EXPERIMENT-8

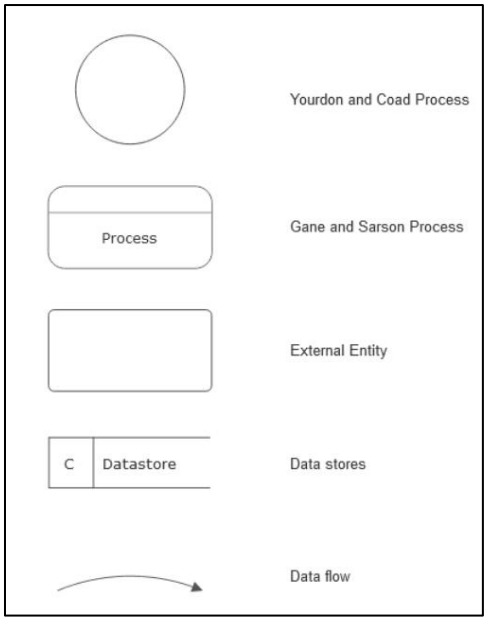
AIM: To draw Data Flow Diagram for Election Voting System.

INTRODUCTION TO THE DIAGRAM: A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates, its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

Data Flow Diagrams Symbols:

There are essentially two different types of notations for data flow diagrams (Yourdon & Coad or Gane & Sarson) defining different visual representations for processes, data stores, data flow and external entities.

- Yourdon and Coad type data flow diagrams are usually used for system analysis and design, while Gane and Sarson type DFDs are more common for visualizing information systems.
- Visually, the biggest difference between the two ways of drawing data flow diagrams is how processes look. In the Yourdon and Coad way, processes are depicted as circles, while in the Gane and Sarson diagram the processes are squares with rounded corners.



Process Notations. A process transforms incoming data flow into outgoing data flow.
Datastore Notations. Datastores are repositories of data in the system. They are sometimes also referred to as files.
Dataflow Notations. Dataflows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.

External Entity Notations. External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.

Data Flow Diagram Levels: A context diagram is a top level (also known as "Level 0") data flow diagram. It only contains one process node ("Process 0") that generalizes the function of the entire system in relationship to external entities.

- **DFD Layers.** Draw data flow diagrams can be made in several nested layers. A single process node on a high-level diagram can be expanded to show a more detailed data flow diagram. Draw the context diagram first, followed by various layers of data flow diagrams.
- **DFD Levels.** The first level DFD shows the main processes within the system. Each of these processes can be broken into further processes until you reach pseudo code.

INTRODUCTION TO THE PROBLEM STATEMENT: The problem statement makes a DFD that provides an overview of how information flows within the system, helping manage the election process efficiently.

It consists of the following:

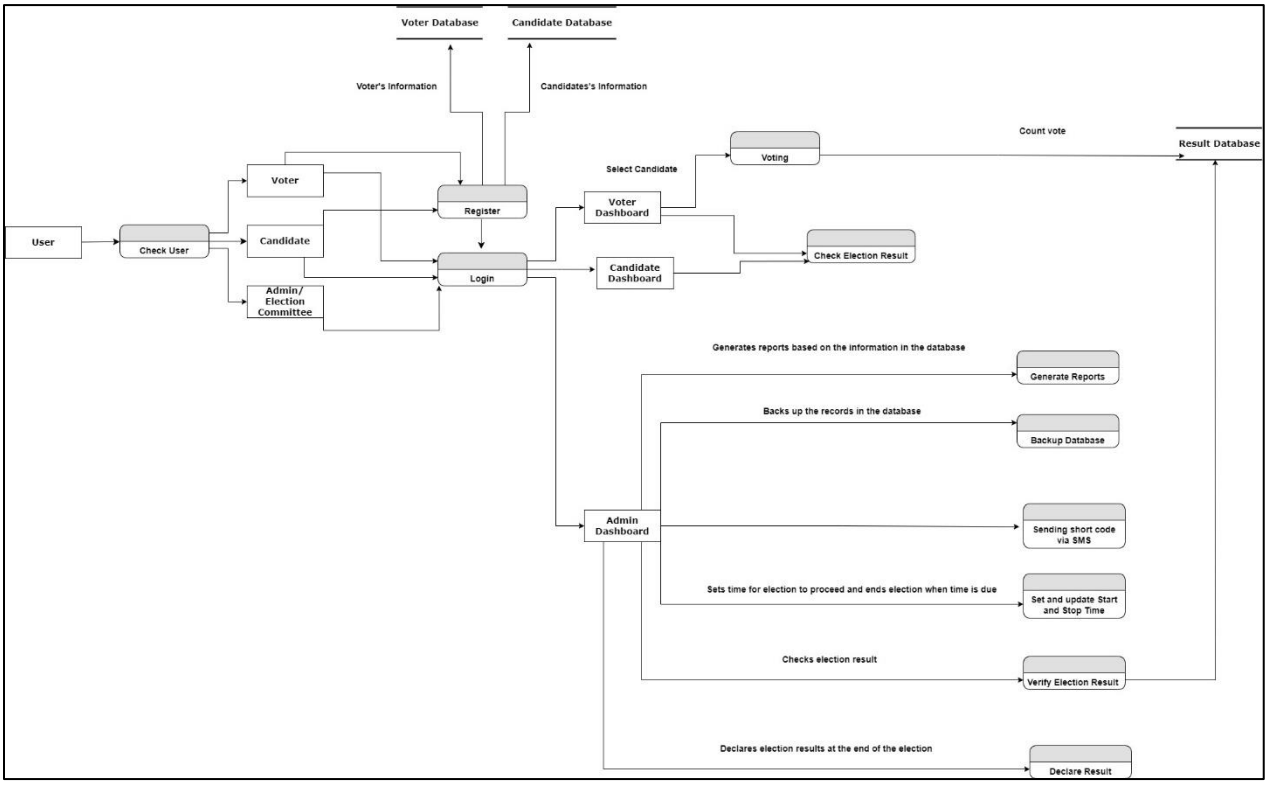
1. **Processes:**
    - **Check User**: This process verifies the validity of the user attempting to access the system.
    - **Login**: Users (voters, candidates, admin, or election committee) log in to the system.
    - **Voting**: Voters can cast their votes.
    - **Check Election Result**: Admin or election committee members can verify election results.
    - **Generate Reports**: This process creates reports based on information from the system.

2. **Data Stores:**
    - **Voter Database**: Contains information about registered voters.
    - **Candidate Database**: Stores details of candidates.
    - **Result Database**: Holds election results.

3. **Data Flows:**
    - **User**:
        - After being checked for validity, users are categorized into voters or candidates/admin/election committee based on data from the Voter Database.
        - Voters can register and then log in to their dashboard.
        - Candidates/Admin/Election Committee members also log in to their respective dashboards.

- **Voter Dashboard**:
  - Voters can select candidates from the Candidate Database.
  - Voting takes place.
- **Candidates Dashboard**:
  - Candidates/Admin/Election Committee members can check election results.
  - Reports can be generated based on information from all three databases.
- **Backup Database**:
  - Reports are backed up.
- **Admin Dashboard**:
  - Election results are verified before being declared.

DATA FLOW DIAGRAM:



EXPERIMENT-9

(OPEN ENDED)

AIM: Create a UML Activity Diagram for processing an order. Once an order has been finalized, four parties are involved in processing it: Online Sales, Accounting, Shipping, and Printing. Online Sales sends the order to Printing, where the associated PDF file is inspected. If the file is not suitable, a new file is requested from the customer. Once the file is suitable, Accounting is informed to charge the credit card. While this is done, Printing carries out the actual printing, and sends the result to Shipping. When Shipping has both received the printed products and confirmation from Accounting that payment was successful, the products are shipped to the customer.

INTRODUCTION TO THE PROBLEM STATEMENT: The given problem visually captures the flow of actions during the order handling process. Each component plays a crucial role in ensuring a smooth workflow.

The following are the components of the Activity Diagram:

1. **Start State:** Represents the beginning of the process. It's where the flow starts when an order is placed by the customer.

2. **Activities:** Key activities include:
    - **Receive Order**: The system receives the order from the customer.
    - **Inspect PDF**: The system checks whether the received file (PDF) is suitable.
    - **Print Order**: If the file is suitable, the order is printed.
    - **Enter Credit Card Details**: Under the order processing phase, credit card details are entered.
    - **Ship Order to Customer**: The order is shipped to the customer.

3. **Decision Node ("Inspect PDF"):**
    - This diamond-shaped node represents a decision point.
    - Depending on whether the PDF file is suitable or not, the flow diverges:
        - If suitable, the process continues to printing.
        - If not suitable, there's a feedback loop back to receiving a new file.

4. **End Node ("Order Received by Customer"):**
    - Represents the completion of the process.
    - The order has been successfully received by the customer.

ACTIVITY DIAGRAM: