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

THEORY –

Introduction to Entity-Relationship Modelling

Entity-Relationship (ER) modelling is a fundamental concept in database design and software engineering that serves as a cornerstone for creating robust and well-structured databases. As a high-level data modelling approach, it provides software engineers and database designers with a powerful tool to conceptualize and represent complex data relationships in a systematic manner.

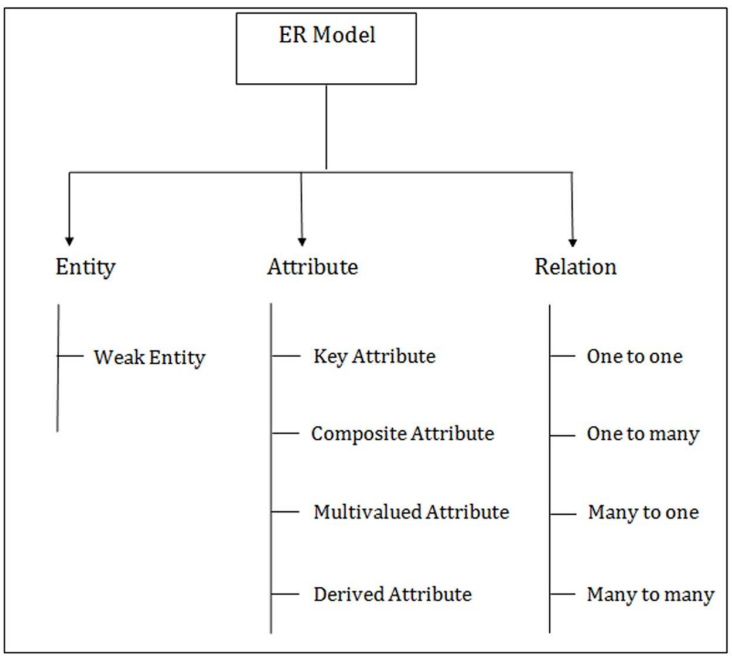
Benefits of ER Modelling

- Simplification of Complex Systems: ER models break down complex data structures into manageable, understandable components.
- Communication Tool: They serve as an effective communication medium between database designers, developers, and other stakeholders.
- Design Validation: The visual nature of ER diagrams helps in identifying potential design flaws or missing elements early in the development process.
- Documentation: ER models provide valuable documentation that can be referenced throughout the system's lifecycle.

Role in Software Engineering

In the context of software engineering, ER modelling plays a crucial role in the database design phase of system development. It bridges the gap between requirements analysis and database implementation, ensuring that the resulting database structure accurately reflects the system's data requirements while maintaining efficiency and integrity.

Components of ER-Diagram:



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), full name, email_id, info_id(Foreign Key), salary_id(Foreign Key).

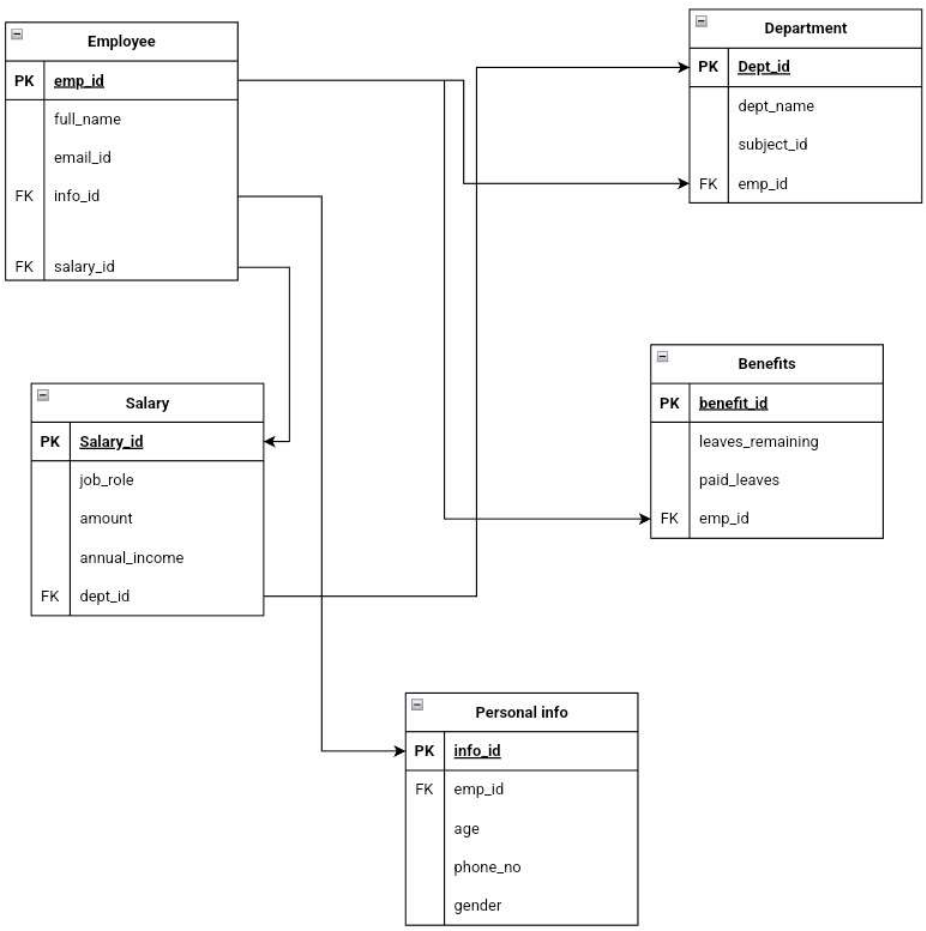
2. Personal Info: It represents the personal info an employee. Attributes include payroll id (Primary Key), emp_id (Foreign Key), age, gender, phone_no.

3. Salary: It represents the actual salary paid to the employees. Attributes include salary id (Primary Key), department id (Foreign Key), amount, annual, job_role.

4. Department: It represents the department to which the employee belongs. Attributes include department id (Primary Key), employee id (Foreign Key), subject id, dept_name.

5. Benefits: It represents the facilities provided by the University. Attributes include benefit id (Primary Key), paid_leaves, emp_id(Foreign Kepaid_leaves, emp_id (Foreign Key), leaves_remaining.

ER-DIAGRAM:



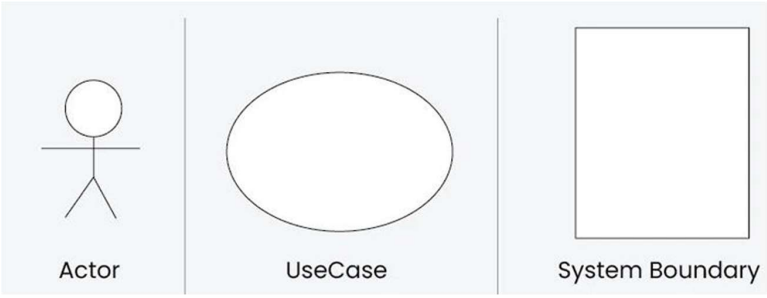
AIM: To draw Use Case diagram for Smart Agriculture Management System.

INTRODUCTION TO THE DIAGRAM: Use Case Diagrams are a fundamental component of the Unified Modelling Language (UML), representing a powerful tool for visualizing and documenting system behaviour from an external perspective. These diagrams serve as a crucial bridge between technical and non-technical stakeholders by providing a clear, visual representation of system functionality and user interactions.

Understanding Use Case Diagrams

Use Case Diagrams belong to the behavioural diagram family within UML and are primarily employed during the requirements gathering and analysis phases of software development. They illustrate the interactions between users (actors) and the system, focusing on what the system does rather than how it does it. This high-level view makes them particularly valuable for understanding system requirements and planning development strategies.

Core Components of Use Case Diagrams



Role in Software Engineering - In the software development lifecycle, Use Case Diagrams play a vital role in:

- Requirements Engineering: Capturing and documenting user requirements
- System Analysis: Understanding system behaviour and interactions
- Project Planning: Defining development phases and milestones
- Testing: Developing test cases and scenarios
- Documentation: Creating user manuals and system documentation

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

Use Cases:

Login/SignUp

- Authentication and authorization management
- Secure access to system features

Farmer Dashboard

- Contains all the necessary details for the farmer to check on regular basis.

Farm

- **Weather Forecast** – Access real-time weather predictions and alerts to optimize farming activities.

- **Fertilizers & Pesticides** – Schedule and track applications to ensure proper usage and effectiveness.
- **Soil pH** – Continuously monitor and maintain soil and water pH levels for optimal crop growth.
- **Irrigation Timing** – Set, adjust, and monitor irrigation schedules to ensure efficient water distribution.

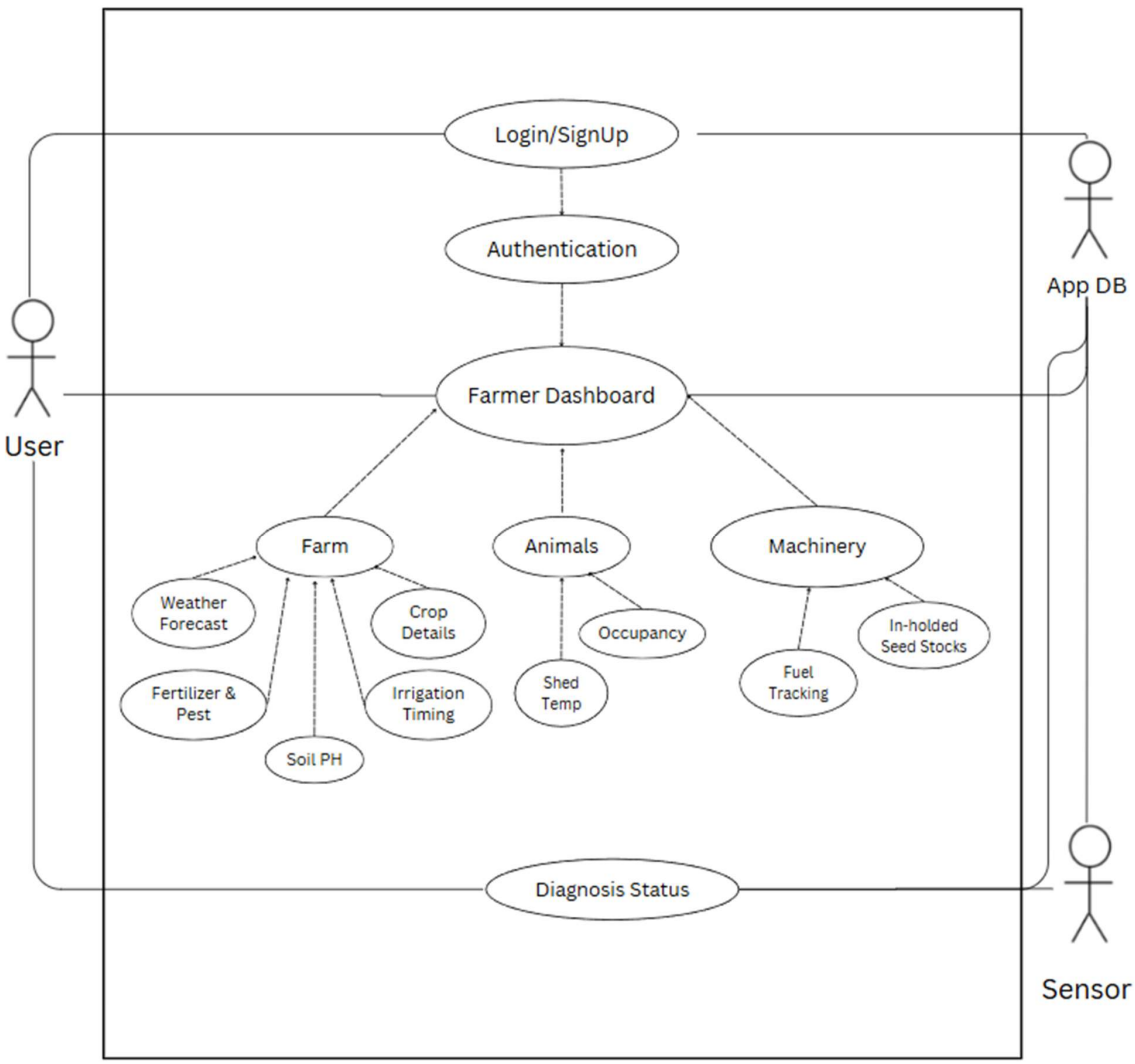
Animals

- **Shed Temperature** – Track and regulate the temperature inside animal sheds for better livestock health.
- **Occupancy** – Monitor real-time animal count and movement within the shed for efficient space management.

Machinery

- **Fuel Tracking** – Keep records of fuel consumption and efficiency to optimize machinery usage.
- **In-holded Seed Stocks** – Manage inventory of stored seeds and track availability for future use.

USE CASE DIAGRAM



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

Introduction to Class Diagrams:

Class diagrams are a type of Unified Modelling Language (UML) diagram utilized in software engineering to visually depict the structure and relationships among classes in a system. These diagrams serve as essential tools for constructing and understanding object-oriented systems.

Structure of Class Diagrams

- Classes: Represented as boxes containing three compartments:
    - Class Name: The top compartment displays the name, typically centered and bold.
    - Attributes: The second compartment lists attributes (data members) with visibility (e.g., public, private) and data types.
    - Methods: The third compartment outlines methods (functions) with visibility, return types, and parameters.
- Associations: Lines connecting classes illustrate relationships (e.g., one-to-one, one-to-many), offering a high-level overview of the system's design.

Visibility Notation

- + for public (accessible to all classes)
- - for private (accessible only within the class)
- # for protected (accessible to subclasses)
- ~ for package or default visibility (accessible within the same package)

Introduction to the Problem Statement

The following classes represent a Pollution Control system:

- **Authority** (authorityID, name, jurisdiction, contactInfo)
- **PollutionControlSensor** (deviceID, deviceType, installationDate, status, efficiency)
- **PollutionSource** (sourceID, location, category, emissionLevel)
- **MonitoringStation** (stationID, location, monitoredPollutants, status)
- **Inspection** (inspectionID, inspectorName, date, findings)
- **PollutionReport** (reportID, reportDate, recordedLevels, affectedArea)
- **Violation** (violationID, sourceID, detectedBy, severity, status)
- **ControlMeasure** (measureID, targetPollutant, effectiveness, cost)

Relationship:

• Authority oversees Inspection processes.

• PollutionSource is monitored by MonitoringStation.

• Inspection evaluates PollutionSource for compliance.

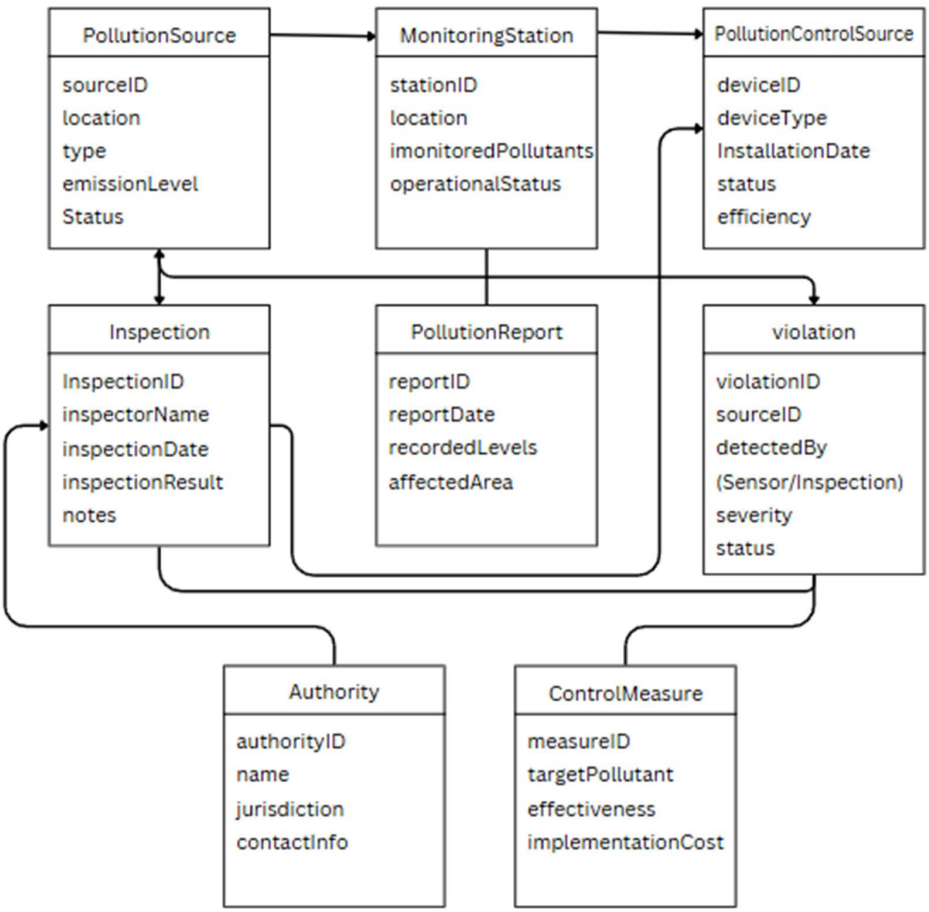
• MonitoringStation generates PollutionReport.

• Violation is linked to PollutionSource and detected by Inspection or Sensors.

• Violation is identified during Inspection.

• ControlMeasure is applied to mitigate Violation.

CLASS DIAGRAM –



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

Introduction to State Machine Diagrams:

A State Machine Diagram is a behavioural diagram used to represent the conditions of a system or a part of a system at specific moments in time. It models the dynamic behaviour of a class in response to time and external stimuli through finite state transitions. These diagrams are also known as State Machines Diagrams or State-Chart Diagrams, and the terms are often used interchangeably.

State Machine Diagrams are particularly useful for modelling classes with three or more states.

Components of a State Machine Diagram

1. Initial State: Represented by a filled black circle, indicating the starting point of the process.
2. Transition: Shown as a solid arrow, indicating a change of control from one state to another, labelled with the event causing the change.
3. State: Depicted as a rounded rectangle, representing the conditions of an object at a specific moment.
4. Fork: Indicates a state splitting into two or more concurrent states.
5. Join: Represents the convergence of two or more states into one upon the occurrence of an event.
6. Self-Transition: Occurs when the state of an object remains unchanged despite an event.
7. Composite State: Represents a state with internal activities.
8. Final State: Illustrated as a filled circle within a circle, marking the end of the state machine.

Introduction to the Problem Statement

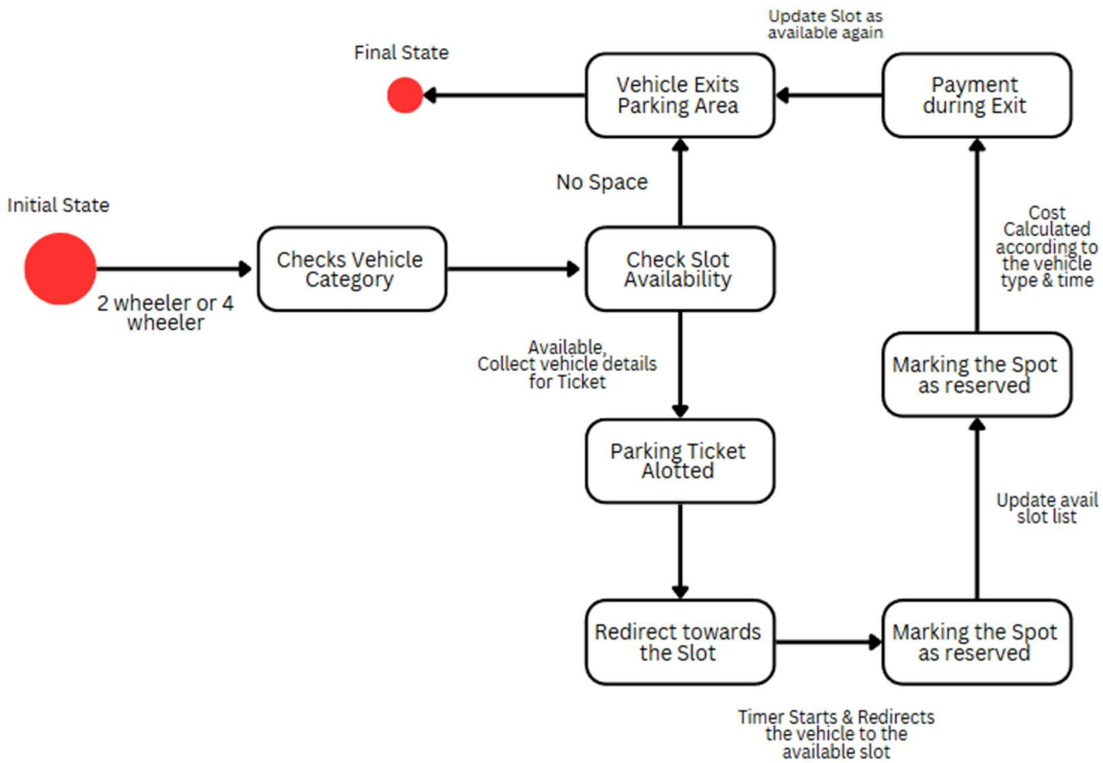
The following states represent the parking management process:

1. Initial State: Vehicle Arrives
    - Marks the beginning of the process when a vehicle enters the parking area.
2. State: Checking Vehicle Type
    - The system checks whether the vehicle is a four-wheeler or a two-wheeler, branching into appropriate paths based on the type.
3. State: Checking Availability
    - The system checks for available parking spots. If no spots are available, the vehicle must leave the parking space.
4. State: Parking Ticket Issued
    - Once the vehicle is classified and a spot is available, a parking ticket is issued, and the timer begins.
5. State: Redirecting Based on Available Spots
    - Vehicles are redirected to available spots based on their type.

- The selected parking spot is marked as reserved for the vehicle.

7. State: Payment at Exit Gate
    - The user makes a payment based on the vehicle type and the duration of parking.
8. State: Vehicle Exits the Parking Space
    - After payment, the vehicle owner frees up the parking space, which then becomes available for other vehicles.
9. Final State: Leaving the Parking Space
    - The process concludes as the vehicle leaves the parking area, and the available spots are updated accordingly.

STATE MACHINE DIAGRAM –



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

Introduction to UML Sequence Diagrams

UML Sequence Diagrams are interaction diagrams that illustrate how operations are performed within a system. They capture the interactions between objects in the context of a collaboration, focusing on the order of messages exchanged over time. The vertical axis represents time, allowing for a clear visualization of the sequence of interactions.

Key Features of Sequence Diagrams

- Collaboration Representation: They detail interactions that realize a use case or operation, showcasing high-level interactions between users, systems, or subsystems.
- Time Focus: Sequence Diagrams emphasize the timing and order of message exchanges.

Components of a Sequence Diagram

- Lifeline: Represents an object or participant in the interaction.
- Activations: Indicate periods during which an object is active and processing.
- Call Message: Represents a call to an operation on another object.
- Return Message: Indicates a response from a called operation.
- Self-Message: A message sent from an object to itself.
- Recursive Message: A message that leads to a recursive call.
- Create Message: Indicates the creation of a new object.
- Destroy Message: Represents the destruction of an object.
- Duration Message: Indicates the time duration of an operation.

Introduction to the Problem Statement

The following interactions illustrate the sequence of operations in the agricultural management system:

1) **Lifelines:**

    - Farmer/User: Initiates interactions like login, data requests, and account management.
    - Mobile App/Dashboard: Acts as the interface for user interactions and communicates with the cloud server.
    - Cloud Server: Processes requests, fetches data, and controls IoT devices.
    - IoT Gateway: Bridges communication between the cloud and IoT sensors.
    - IoT Sensor: Collects environmental data (e.g., soil moisture) and sends it to the cloud.
    - Weather API: Provides real-time weather updates to the cloud server.
    - Irrigation System: Automates watering based on data received from sensors and weather conditions.

- Login & Authentication

- The Farmer/User initiates login or sign-up via the Mobile App/Dashboard.
- The Mobile App/Dashboard forwards the authentication request to the Cloud Server.
- The Cloud Server verifies credentials and responds with account status.

- Data Request & Fetching

  - The Farmer/User requests agricultural data from the Mobile App/Dashboard.
  - The Mobile App/Dashboard forwards the request to the Cloud Server.
  - The Cloud Server fetches the required data and sends it back to the Mobile App/Dashboard.

- Soil Moisture Monitoring & Weather Update

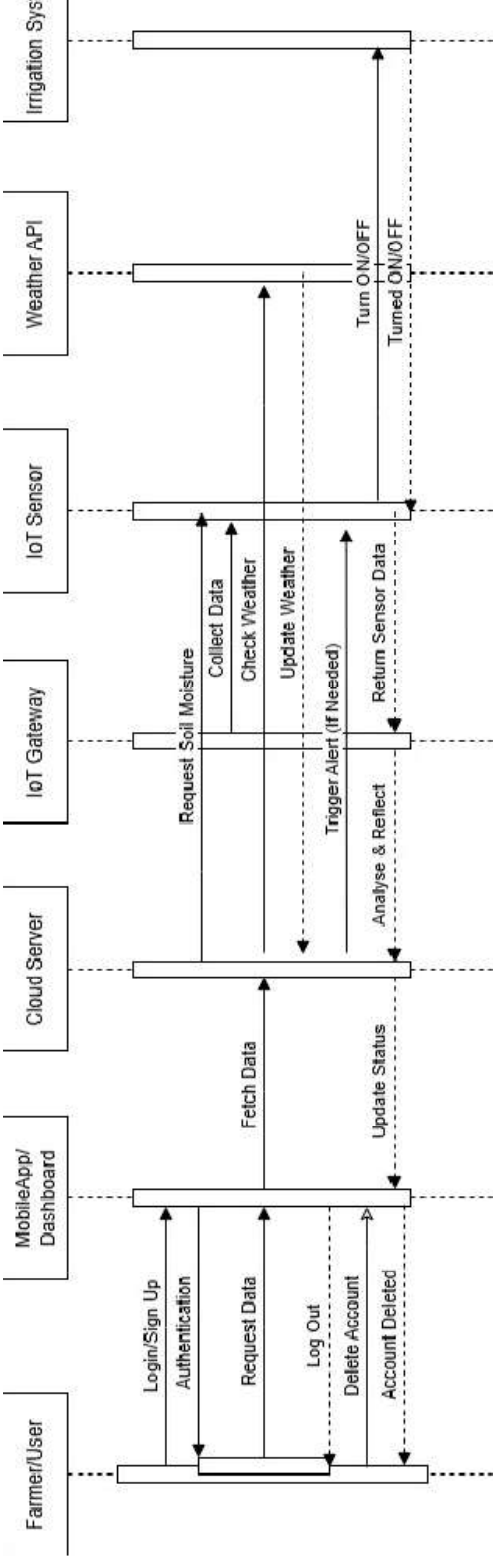
  - The Cloud Server requests soil moisture data from the IoT Gateway.
  - The IoT Gateway forwards the request to the IoT Sensor.
  - The IoT Sensor collects soil data and sends it back via the IoT Gateway.
  - The Cloud Server requests a weather update from the Weather API.
  - The Weather API returns weather data to the Cloud Server.

- Irrigation Automation & Alert Handling

  - The Cloud Server analyzes sensor and weather data.
  - If necessary, it triggers an alert via the IoT Gateway.
  - The Cloud Server decides whether to turn the irrigation system ON/OFF.
  - The Irrigation System receives the command and updates the status.

- Account Management (Logout & Delete Account)

  - The Farmer/User logs out or deletes their account via the Mobile App/Dashboard.
  - The Cloud Server processes the request and updates the system status accordingly.



SEQUENCE DIAGRAM OF SMART AGRICULTURE MANGAGEMENT SYSTEM

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.

Components of the Smart Agriculture System Diagram

1. Smart Agriculture System:
    - Represents the complete system that integrates various components for efficient farming.
    - Facilitates login and sign-up interactions for farmers.
2. User/Farmer:
    - The end-user interacting with the system through a Dashboard/Mobile App.
    - Can log in and access agricultural data.
3. Dashboard/Mobile App:
    - Acts as the user interface for farmers.
    - Sends login and sign-up requests to the Cloud Server.
    - Fetches and displays sensor data and weather updates.
4. Cloud Server:
    - Handles authentication requests from the Dashboard/Mobile App.
    - Manages and processes sensor data.
    - Communicates with the IoT Gateway for data collection.

- Acts as an intermediary between the Cloud Server and Sensors.
- Sends collected environmental data to the Cloud Server.

6. Sensors:

- Collect real-time environmental data such as soil moisture.
- Send data through the IoT Gateway to the Cloud Server.

7. Weather API:

- Provides real-time weather updates.
- Shares data with the Cloud Server for better irrigation management.

8. Soil Sensors:

- Measure soil moisture levels and send data through the Sensors.
- Help optimize irrigation decisions.

9. Motor & Water Sprinkler:

- Controlled based on data from the Cloud Server.
- Automatically start or stop irrigation based on soil and weather conditions.

Interaction Flows

Login/Sign-Up Flow:

- The User/Farmer logs in via the Dashboard/Mobile App.
- The Dashboard/Mobile App sends authentication requests to the Cloud Server.

IoT Gateway & Data Collection Flow:

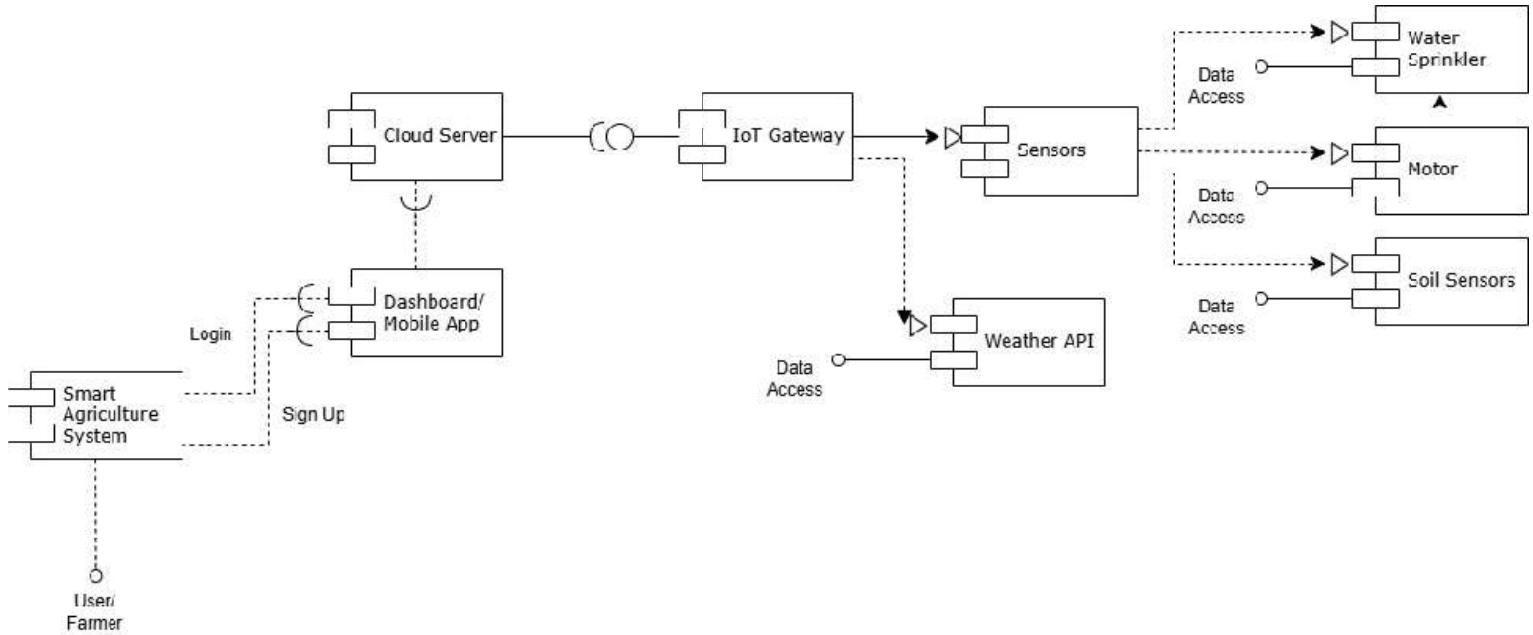
- The Sensors collect real-time data and send it to the IoT Gateway.
- The IoT Gateway forwards this data to the Cloud Server for processing.

Weather Data Access Flow:

- The Cloud Server requests real-time weather updates from the Weather API.

Irrigation Control Flow:

- The Cloud Server analyzes data from Sensors and the Weather API.
- The Motor and Water Sprinkler adjust irrigation accordingly.


Cloud Server
IoT Gateway
Sensors
Water Sprinkler
Motor
Soil Sensors
Data Access
Data Access
Data Access
Dashboard/ Mobile App
Login
Sign Up
Smart Agriculture System
Weather API
Data Access
User/ Farmer

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

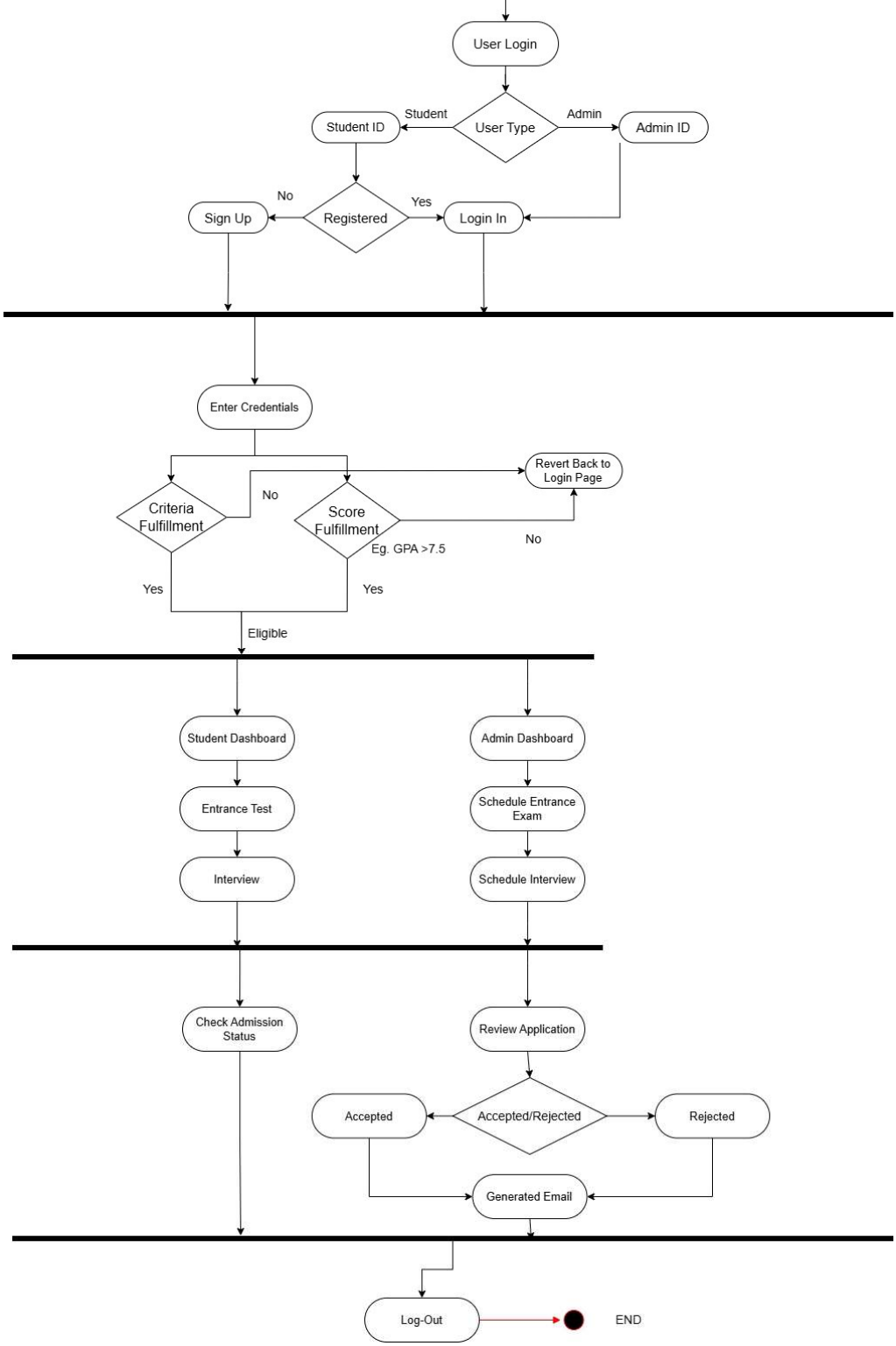
- **Initial State:** A small filled circle with an arrow represents the starting point.
- **Action State:** A rounded rectangle represents a non-interruptible action.
- **Action Flow:** Arrows indicate transitions between actions.
- **Object Flow:** Arrows from actions to objects show creation/modification; from objects to actions indicate usage.
- **Decision & Branching:** A diamond represents a decision with alternate paths.
- **Guards:** Conditions next to a decision diamond that must be true to proceed.
- **Synchronization:** A fork splits a flow into multiple concurrent flows; a join merges them back.
- **Time Event:** An hourglass symbol indicates a delay.
- **Merge Event:** Combines multiple flows that are not concurrent.
- **Signals:** Represent external modifications; sent and received in pairs.
- **Interrupting Edge:** A lightning bolt represents an event that halts the flow.
- **Swimlanes:** Columns that group related activities.
- **Final State:** A filled circle inside another circle marks the end.

Activity Diagram Explanation

1. **Start Node:**
    - The process begins with a **User Login** action.
2. **User Login Decision:**
    - The user is classified as either **Student** or **Admin** based on their credentials.
    - **Student:** Requires a **Student ID** for verification.
    - **Admin:** Requires an **Admin ID** for access.
3. **Student Path:**
    - If the student is **not registered**, they must **Sign Up** before proceeding.
    - If already registered, they proceed to **Login In** and enter their credentials.
    - Their eligibility is checked based on **Criteria Fulfillment** and **Score Fulfillment (e.g., GPA > 7.5)**.

- If eligible, they proceed to the **Student Dashboard**.
- They take an **Entrance Test** and **Interview** as part of the admission process.
- They can later **Check Admission Status**.

4. **Admin Path:**
   - Upon successful login, the admin accesses the **Admin Dashboard**.
   - Admins can **Schedule Entrance Exams** and **Schedule Interviews** for students.
   - They **Review Applications** and make the final decision (**Accepted/Rejected**).
5. **Admission Decision:**
   - If **Accepted**, an email is **Generated and Sent** to the student.
   - If **Rejected**, the process ends for that student.
6. **Logout and End Process:**
   - After completing tasks, the user can **Log Out**, marking the **End** of the process.


User Login
User Type
Student
Admin
Student ID
Admin ID
Registered
No
Yes
Sign Up
Login In
Enter Credentials
Revert Back to Login Page
Criteria Fulfillment
No
Score Fulfillment
Eg. GPA >7.5
No
Yes
Yes
Eligible
Student Dashboard
Admin Dashboard
Entrance Test
Schedule Entrance Exam
Interview
Schedule Interview
Check Admission Status
Review Application
Accepted
Accepted/Rejected
Rejected
Generated Email
Log-Out
END

AIM: To draw a Data Flow Diagram (DFD) for the Election Voting System.

INTRODUCTION TO THE DIAGRAM:

A Data Flow Diagram (DFD) visually represents the flow of information within a system. It describes how data moves between different entities, processes, and data stores without specifying technical implementation details. DFDs help in understanding the logical structure of a system.

Basic DFD Notations and Symbols:

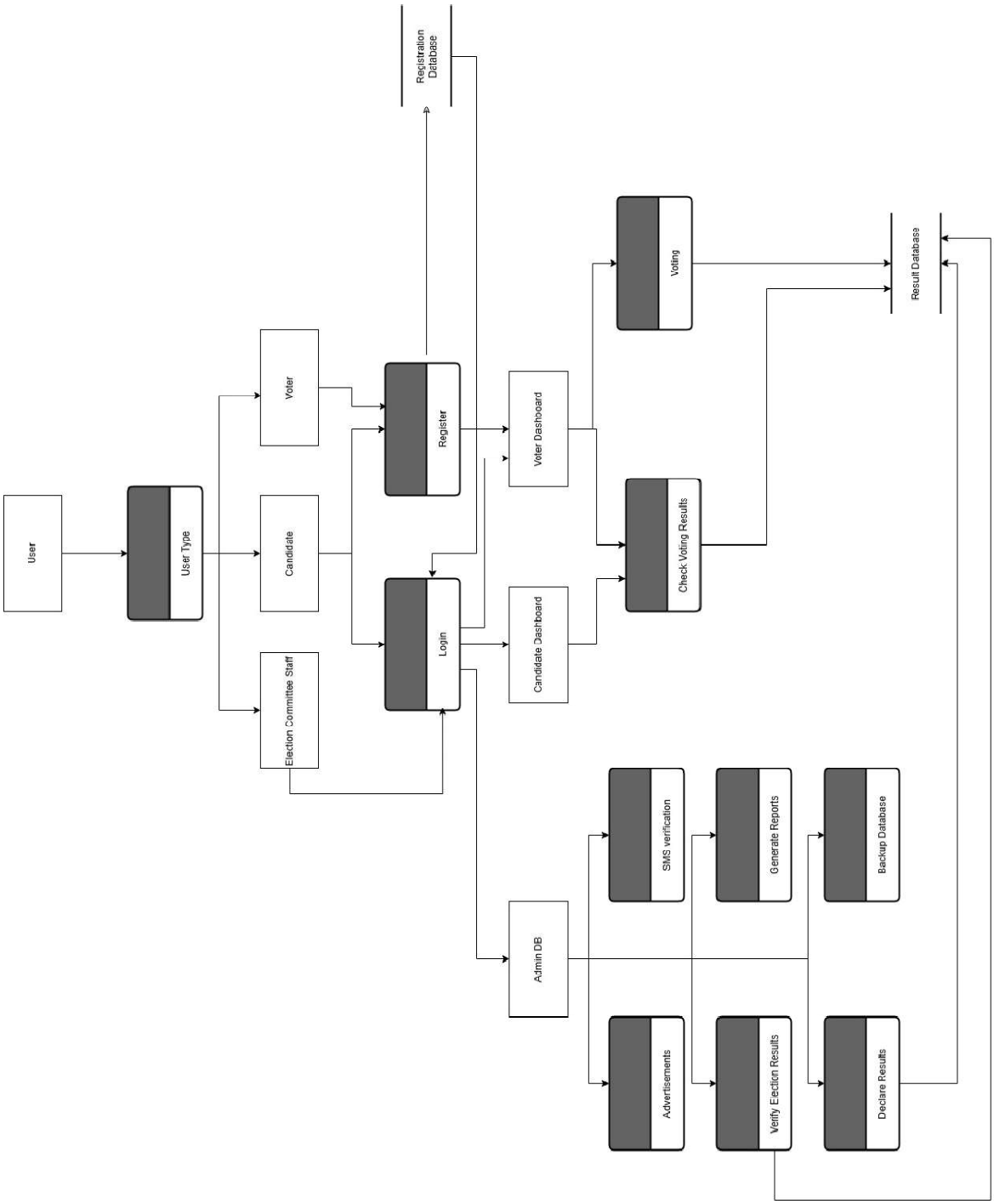
- External Entity: Represents external users or systems that interact with the system (e.g., Voter, Candidate, Election Committee Staff).
- Process: A function or operation performed on data (e.g., Login, Register, Voting, Check Results).
- Data Store: A repository where data is stored (e.g., Registration Database, Result Database, Admin DB).
- Data Flow: Arrows indicating the movement of data between entities, processes, and data stores.

Data Flow Diagram Explanation:

1. **User Identification:**
    - A User selects their type (Election Committee Staff, Candidate, or Voter).
    - Users can Register (storing data in the Registration Database) or Login to access their dashboards.
2. **Candidate and Voter Actions:**
    - Candidates access their Candidate Dashboard to monitor election activities.
    - Voters access their Voter Dashboard to participate in elections.
    - Voters can cast their votes through the Voting process, updating the Result Database.
    - They can also Check Voting Results once the process is complete.
3. **Election Committee Staff Role:**
    - Admins manage election operations through the Admin DB, including:
        - Advertisements for election promotions.
        - SMS verification to authenticate users.
        - Verify Election Results before declaring them.
        - Generate Reports and Backup the Database.
4. **Election Result Declaration:**
    - Once results are verified, they are stored in the Result Database.
    - The system allows for Result Declaration, completing the election process.

The DFD for the Election Voting System provides a clear overview of how data flows between users, processes, and databases. It ensures a structured election process, from user authentication to result declaration, maintaining transparency and efficiency.

Data Flow Diagram:



(CASE STUDY)

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: In any e-commerce system, order processing involves multiple interconnected steps. The activity diagram represents the flow of operations from the moment an order is placed until it is successfully delivered. Each phase is managed by a specific department, ensuring efficiency and accuracy.

The following four entities participate in the workflow:

1. Online Sales – Handles order reception and file validation.
2. Printing – Inspects the submitted file and prints the order if approved.
3. Accounting – Manages the payment process and verifies transactions.
4. Shipping – Dispatches and delivers the final product to the customer.

COMPONENTS OF THE DIAGRAM:

1. Start State

- The process begins when a customer places an order.
- Online Sales receives and validates the order.

2. File Inspection & Order Validation

- The associated PDF file is inspected in the Printing department.
- If the file is unsuitable, the system requests a new file from the customer.
- Once approved, the order moves forward.

3. Parallel Processing (Simultaneous Tasks)

Two key activities occur simultaneously to optimize workflow:

1. Payment Processing (Accounting Department)
    - The system charges the customer's credit card.
    - If the transaction fails, an alternate payment method is requested.
2. Order Printing (Printing Department)
    - Once the file is validated, the printing process begins.

      The order can only proceed if both steps are successfully completed.

4. Order Shipping & Final Delivery

order.

- The process concludes when the customer receives the order.

KEY FLOW CONTROL:

- **Decision Node:** Requests a new file if needed.
- **Parallel Execution:** Printing and payment happen simultaneously.
- **Final State:** Order is delivered only if all steps succeed.

Activity Diagram: