# TABLE OF CONTENT

# INTRODUCTION

The Airline Reservation System is a software application designed to facilitate the booking and management of flight tickets for passengers. Developed using the C++ programming language, this system offers a comprehensive set of features to enhance the user experience, including –login, registration, booking tickets, editing ticket details, making payments, and displaying ticket information.

## Key Features:-

1. **Login and Registration:** Users can create accounts by registering with the system using their personal information. Upon registration, they can log in securely using their credentials, ensuring a personalized experience with access to their booking history and preferences.

2. **Book Ticket:** The system enables users to search for available flights based on criteria such as date, time, and destination. Once the desired flight is selected, users can proceed to book their tickets by providing passenger details and selecting preferred seating options.

3. **Edit Ticket:** In case of any changes or modifications to the booked tickets, users have the flexibility to edit ticket details such as passenger names, flight dates, or seating preferences. This feature ensures convenience and adaptability for passengers with evolving travel plans.

4. **Payment Integration:** Seamless integration with payment gateways allows users to securely make payments for their booked tickets using various payment methods such as credit/debit cards,

net banking, or digital wallets. The system ensures the confidentiality and integrity of financial transactions for a hassle-free booking experience.

5. **Display Ticket:**  After successful booking and payment, users can conveniently access and view their e-tickets through the system. Detailed information about the booked flight, including itinerary, passenger names, seat numbers, and boarding instructions, is presented in a user-friendly format for easy reference.

## Benefits:-

- Streamlined Booking Process: The intuitive user interface and feature-rich functionality streamline the booking process, making it quick and convenient for users to reserve their flight tickets.
- Enhanced User Experience: Personalized accounts, flexible ticket editing options, and secure payment processing contribute to an enhanced user experience, fostering customer satisfaction and loyalty.
- Efficient Management: The system's comprehensive features, including ticket editing and payment tracking, enable efficient management of flight bookings, minimizing errors and maximizing operational efficiency for airline administrators.

In conclusion, the Airline Reservation System built in C++ offers a robust platform for managing flight bookings effectively while providing users with a seamless and convenient booking experience. With its array of features and user-centric design, the system plays a vital role in modernizing and optimizing the airline industry's ticket reservation process.

# PRACTICAL:- 1

Software Development Life Cycle (SDLC) is a structured process followed by software development teams to design, develop, test, and deploy high-quality software products efficiently. It consists of a series of phases that guide the development process from inception to deployment and maintenance. The main purpose of SDLC is to produce software that meets customer requirements, is delivered on time and within budget, and maintains quality standards throughout its lifecycle.

## The phases of the SDLC typically include:

1. **Requirements Gathering:** In this phase, the development team works closely with stakeholders to gather and analyze requirements for the software project. This involves understanding the needs of end-users, defining functional and non-functional requirements, and documenting them in a comprehensive manner.

2. **Design:** Once the requirements are gathered, the next phase involves designing the architecture and detailed specifications of the software system. This includes defining the system architecture, data models, user interfaces, and software components.

3. **Implementation:** In this phase, the actual coding of the software system takes place based on the design specifications. Developers write code according to coding standards, best practices, and using appropriate programming languages and frameworks.

4. **Testing:** After the implementation phase, the software undergoes rigorous testing to identify and fix defects. Various testing techniques such as unit testing, integration testing,

system testing, and acceptance testing are performed to ensure the software meets quality standards and fulfills the requirements.

5. **Deployment:** Once the software is thoroughly tested and verified, it is deployed to the production environment for end-users to use. This involves installation, configuration, and setup of the software system in the target environment.

6. **Maintenance:** After deployment, the software enters the maintenance phase where it is regularly updated, enhanced, and maintained to address issues, add new features, and adapt to changing user needs and technology advancements.

- ❖ **Now, let's discuss how the Waterfall model, which is one of the traditional SDLC methodologies, can be used in the development of an airline reservation system with features like login, registration, booking ticket, edit ticket, payment, and display ticket:**

1. **Requirements Gathering:** In the Waterfall model, this phase involves thorough documentation of all the requirements for the airline reservation system, including user requirements for features such as login, registration, booking ticket, etc. This documentation should be detailed and comprehensive to avoid misunderstandings later in the development process.

2. **Design:** Once the requirements are gathered, the design phase begins. In this phase, the system architecture, database design, and user interface design for features like login, registration, booking ticket, etc., are developed based on the requirements gathered in the previous phase. Detailed design documents are created to guide the implementation phase.

3. **Implementation:** With the design documents in hand, the development team starts coding the system according to the specifications outlined in the design phase. Each feature, such as login, registration, booking ticket, etc., is implemented sequentially, following the Waterfall model's linear approach.

4. **Testing:** After the implementation phase, the testing phase begins. Each feature of the airline reservation system undergoes testing to ensure it meets the specified requirements and functions correctly. Testing includes unit testing of individual components as well as integration testing to verify that different features work together seamlessly.

5. **Deployment:** Once testing is complete, and the system is deemed stable and ready for release, it is deployed to the production environment. This involves installing the system on servers, configuring it for use by end-users, and conducting any necessary training for administrators and users.

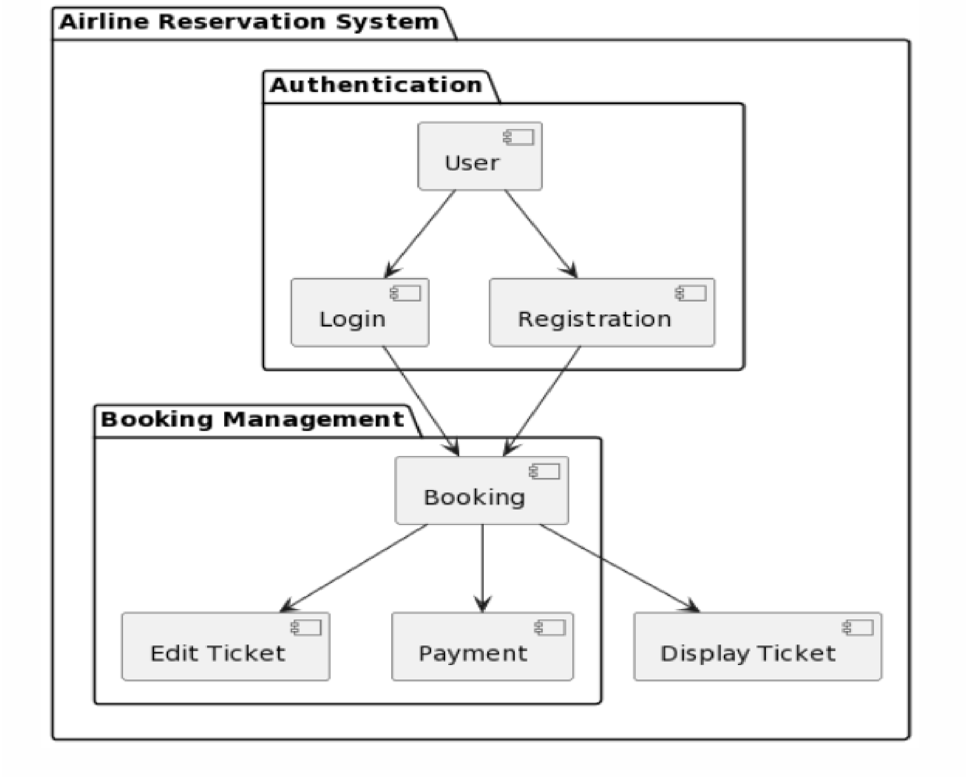
6. **Maintenance:** After deployment, the system enters the maintenance phase, where it is monitored for any issues or bugs that may arise. Updates and enhancements may be made to the system based on user feedback or changing requirements.

The Waterfall model's sequential approach works well for projects with well-defined requirements and a stable scope, such as the development of an airline reservation system. However, it may lack flexibility in accommodating changes during the development process, which could be a drawback if requirements evolve over time.

# PRACTICAL:- 2

## PROJECT BREAKDOWN

In software engineering, a project breakdown refers to the process of dividing a software development project into smaller, manageable components or tasks. This breakdown helps in organizing the project's work, assigning responsibilities to team members, estimating project duration and cost, and tracking progress effectively. Project breakdown typically involves breaking down the project's scope into work packages, tasks, and subtasks, each with its own deliverables and dependencies.



Here's a project breakdown for the Airline Reservation System with features like login, registration, booking ticket, edit ticket, payment, and display ticket:

1. **Project Scope:**

- Develop an Airline Reservation System with the following features:
  - Login
  - Registration
  - Booking Ticket
  - Edit Ticket
  - Payment
  - Display Ticket

2. **Project Breakdown:**

   a. **Login Module:**
   - Design login page UI
   - Implement user authentication logic
   - Integrate with user database
   - Test login functionality

   b. **Registration Module:**
   - Design registration page UI
   - Implement user registration logic
   - Validate user input
   - Store user information in the database
   - Test registration functionality

c. **Booking Ticket Module:**

- Design search flights page UI
- Implement flight search functionality
- Display available flights
- Allow user to select flight and enter passenger details
- Reserve selected flight
- Test booking functionality

d. **Edit Ticket Module:**

- Design edit ticket page UI
- Allow user to select ticket to edit
- Implement logic to modify ticket details
- Update ticket information in the database
- Test edit ticket functionality

e. **Payment Module:**

- Design payment page UI
- Integrate with payment gateway API
- Implement payment processing logic
- Handle payment confirmation and failure scenarios
- Test payment functionality

f. **Display Ticket Module:**

- Design ticket display page UI
- Retrieve ticket information from the database
- Display ticket details to the user
- Test ticket display functionality

3. **Dependencies:**

- Login and Registration modules must be completed before users can access other features.
- Booking Ticket, Edit Ticket, Payment, and Display Ticket modules are independent and can be developed concurrently.
- Payment module depends on successful booking of tickets.

4. **Milestones:**

- Milestone 1: Login and Registration modules completed
- Milestone 2: Booking Ticket module completed
- Milestone 3: Edit Ticket and Payment modules completed
- Milestone 4: Display Ticket module completed

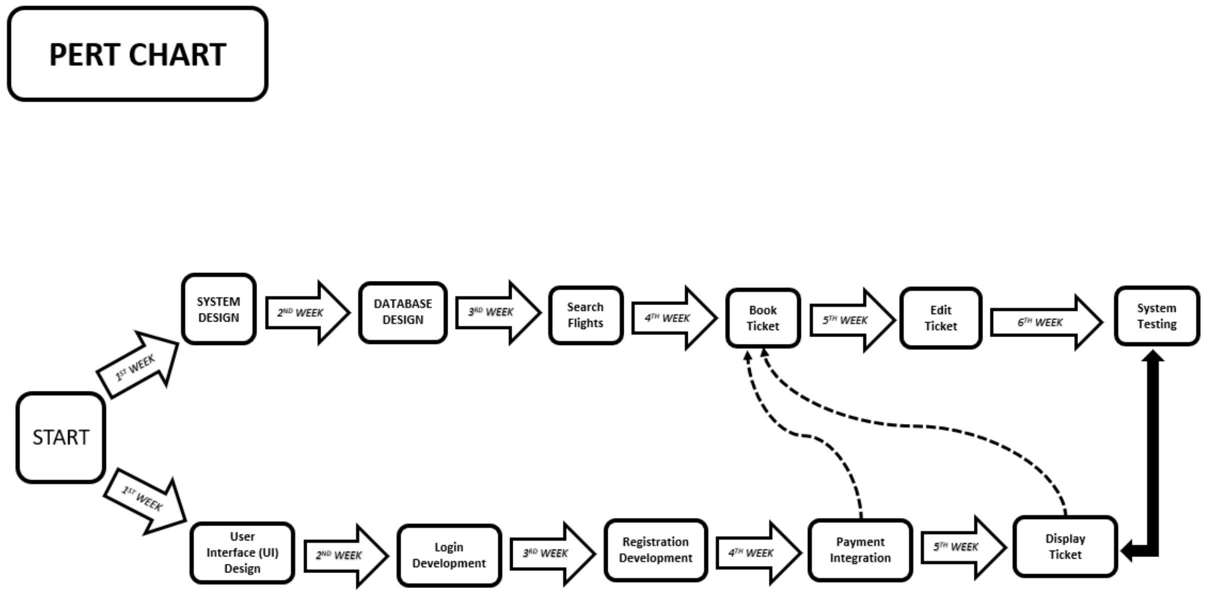
5. **Quality Assurance:**

- Conduct thorough testing at each module level and integration testing after the completion of each module.
- Ensure security, usability, and performance testing for all features.

By breaking down the project into smaller components and tasks, it becomes easier to manage, track progress, and ensure successful delivery of the Airline Reservation System.

## PERT CHART

PERT (Program Evaluation and Review Technique) chart is a project management tool used to plan, organize, and coordinate tasks within a project. It visually represents the sequence of activities, their dependencies, and the estimated time required to complete each task. PERT charts are particularly useful for complex projects with numerous interdependent tasks and uncertain durations.



Key components of a PERT chart include:

1. **Nodes:** Represent individual tasks or activities within the project. Nodes are usually depicted as circles or rectangles.

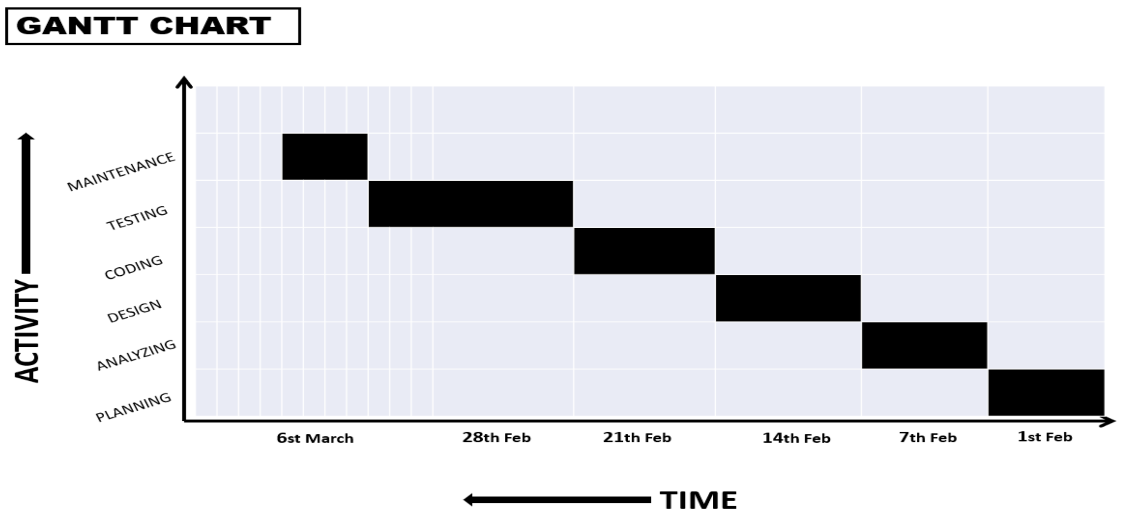
2. **Arrows:** Represent the sequence of activities and their dependencies. Arrows connect nodes to illustrate the order in which tasks must be completed.

3. **Estimates:** Each activity is assigned an estimated duration for completion. This helps in scheduling and resource allocation.

4. **Critical Path:** The longest sequence of tasks that determines the minimum duration required to complete the project is known as the critical path. Any delay in tasks on the critical path will directly impact the project's overall timeline.

## GANTT CHART

A Gantt chart is a type of bar chart used in project management to visualize and track the progress of tasks and activities over time. It provides a graphical representation of a project schedule, showing the start and end dates of individual tasks, their duration, and their dependencies. Gantt charts are widely used because they offer a clear and easy-to-understand visualization of project timelines, helping project managers and team members to plan, coordinate, and monitor project activities effectively.



Key features of a Gantt chart include:

1. **Task List:** The chart typically includes a list of tasks or activities that need to be completed as part of the project.

2. **Time Scale:** The horizontal axis of the chart represents time, usually broken down into days, weeks, or months, depending on the duration of the project.

3. **Bars:** Each task is represented by a horizontal bar on the chart, with the length of the bar indicating the duration of the task. The position of the bar on the time scale shows when the task starts and ends.

4. **Dependencies:** Gantt charts can show dependencies between tasks, indicating which tasks must be completed before others can begin. This helps in identifying critical paths and potential bottlenecks in the project schedule.

5. **Milestones:** Important project milestones, such as project kickoff, major deliverables, or project completion dates, can be marked on the chart to highlight key events in the project timeline.

7. **Progress Tracking:** As tasks are completed, their progress can be visually represented on the chart, allowing project managers to monitor progress and identify any delays or deviations from the planned schedule.

# PRACTICAL:- 3

DFD stands for Data Flow Diagram. It is a graphical representation used in software engineering to model the flow of data within a system. DFDs provide a clear visualization of how data moves through various processes within a system, from input to output.

Key components of a DFD include:

1. **Processes:** Processes represent the various functions or activities performed within the system. Each process takes input data, performs some transformation or processing on it, and produces output data.

2. **Data Stores:** Data stores represent the storage locations where data is temporarily or permanently stored within the system. They can include databases, files, or any other storage medium.

3. **Data Flows:** Data flows represent the movement of data between processes, data stores, and external entities. They show how data is transferred from one part of the system to another.

4. **External Entities:** External entities represent entities outside the system boundary that interact with the system. These can include users, other systems, or external data sources.

## ❖ 0-LEVEL DFD

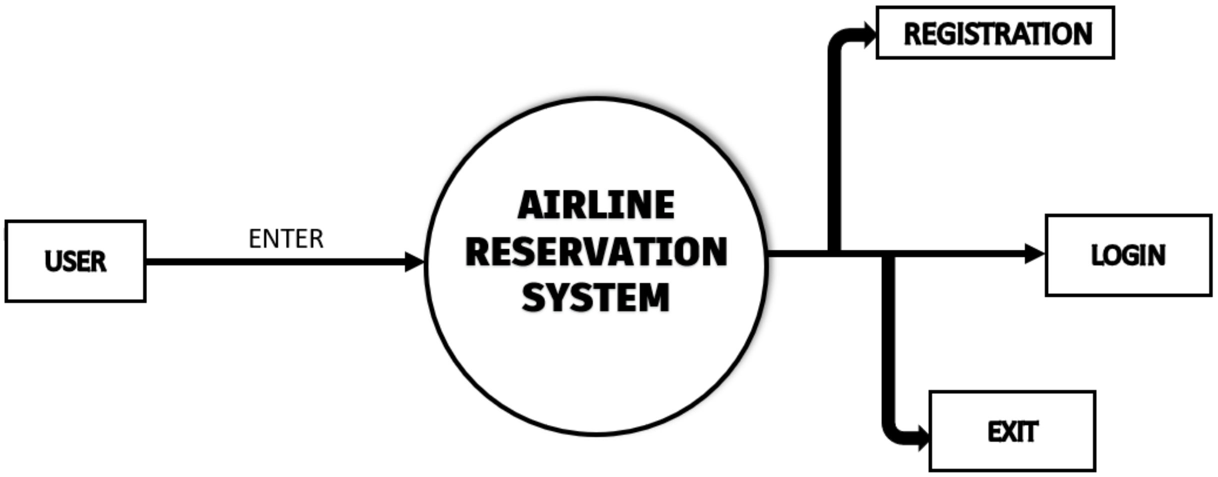
A Level-0 Data Flow Diagram (DFD), also known as a **context diagram**, provides a **high-level overview** of an entire system. It depicts the system as a single **process** and its interactions with **external entities**. These external entities are anything outside the system that interacts with it, such as users, other systems, or databases.

Key features of a Level 0 DFD include:

1. **External Entities:** These represent entities outside the system boundary that interact with the system, such as users, other systems, or data sources.

2. **Processes:** Processes represent high-level functions or activities performed within the system. Each process is depicted as a bubble and represents a major transformation or operation on data.

Level-0 DFD



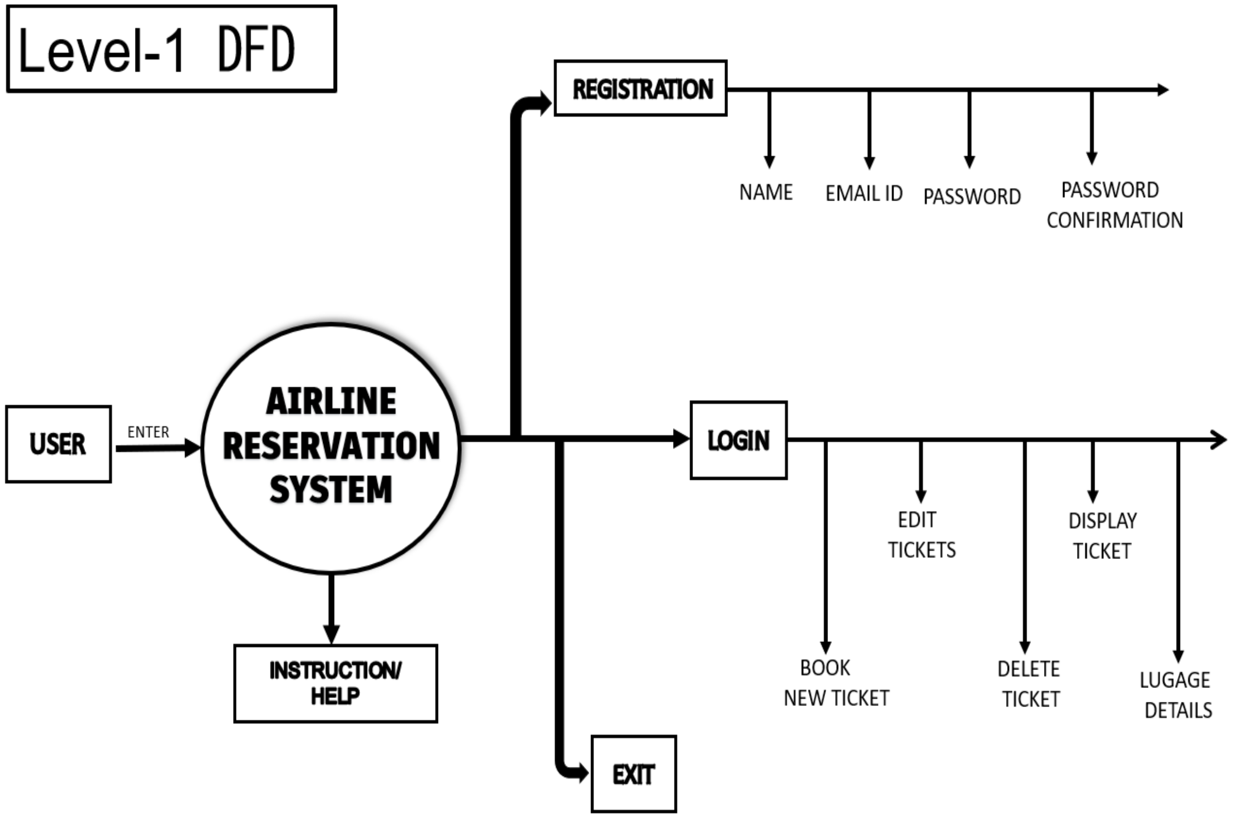
## ❖ 1-LEVEL DFD

A Level-1 Data Flow Diagram (DFD) dives a bit deeper than a Level-0 DFD. It expands on the single process shown in the context diagram by breaking it down into its **sub-processes**.

The purpose of a Level-1 DFD is to:

- Provide a more detailed view of the system's internal workings.
- Show how the sub-processes interact with each other and with external entities.
- Identify the data that is stored and used by the system.

By breaking down the system into smaller, more manageable chunks, the Level-1 DFD helps in understanding the specific tasks performed within the system and how they work together to achieve the overall functionality.



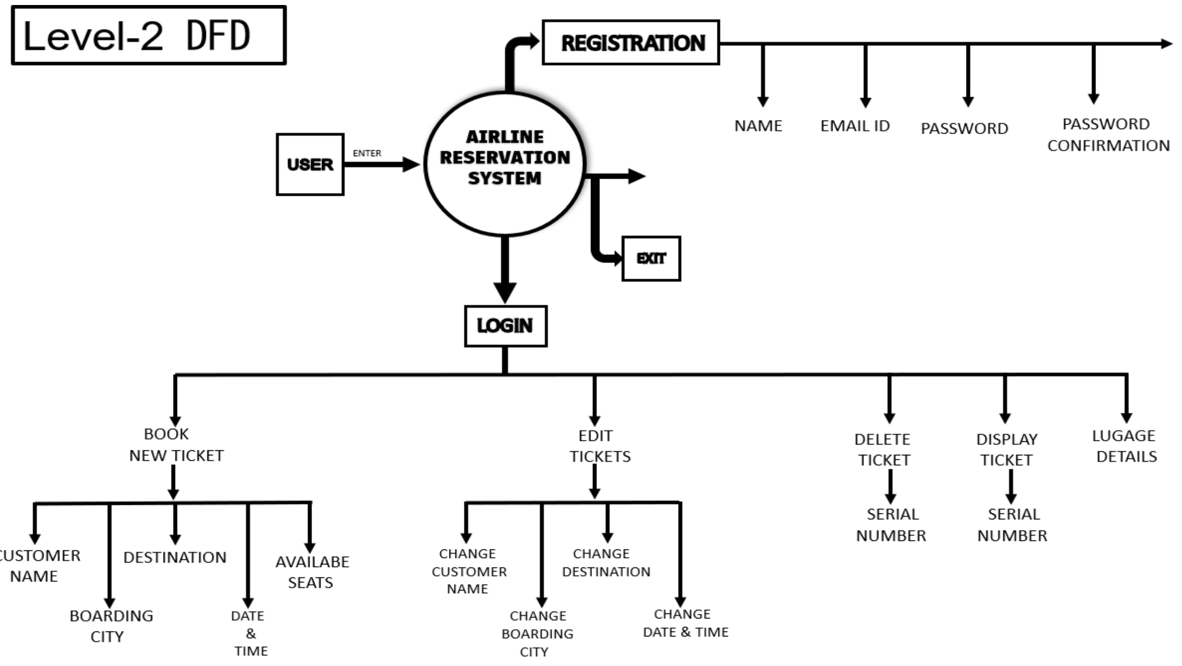
## ❖ LEVEL-2 DFD

A Level 2 Data Flow Diagram (DFD) is a more detailed version of the Level 1 DFD, which provides a deeper understanding of the system's processes and data flows. In a Level 2 DFD, the processes identified in the Level 1 diagram are broken down into smaller subprocesses, and the data flows are elaborated upon to show more specific data transformations and interactions.

Key characteristics of a Level 2 DFD include:

1. **Subprocesses:** Processes identified in the Level 1 DFD are decomposed into smaller subprocesses at the Level 2 DFD. These subprocesses represent the detailed steps or tasks required to accomplish the functions identified in the higher-level processes.

2. **Expanded Data Flows:** Data flows between processes are elaborated upon to show more specific data transformations and interactions. This includes detailing the data attributes or fields being transferred between processes and specifying any data validation or manipulation that occurs.



# PRCTICAL :- 4

An Entity-Relationship (ER) diagram is a graphical representation used in software engineering to model the data schema of a system or application. It depicts the entities (objects or concepts) in the system, their attributes, and the relationships between them. ER diagrams are commonly used in database design to visualize and communicate the structure of a database schema.

Key components of an ER diagram include:

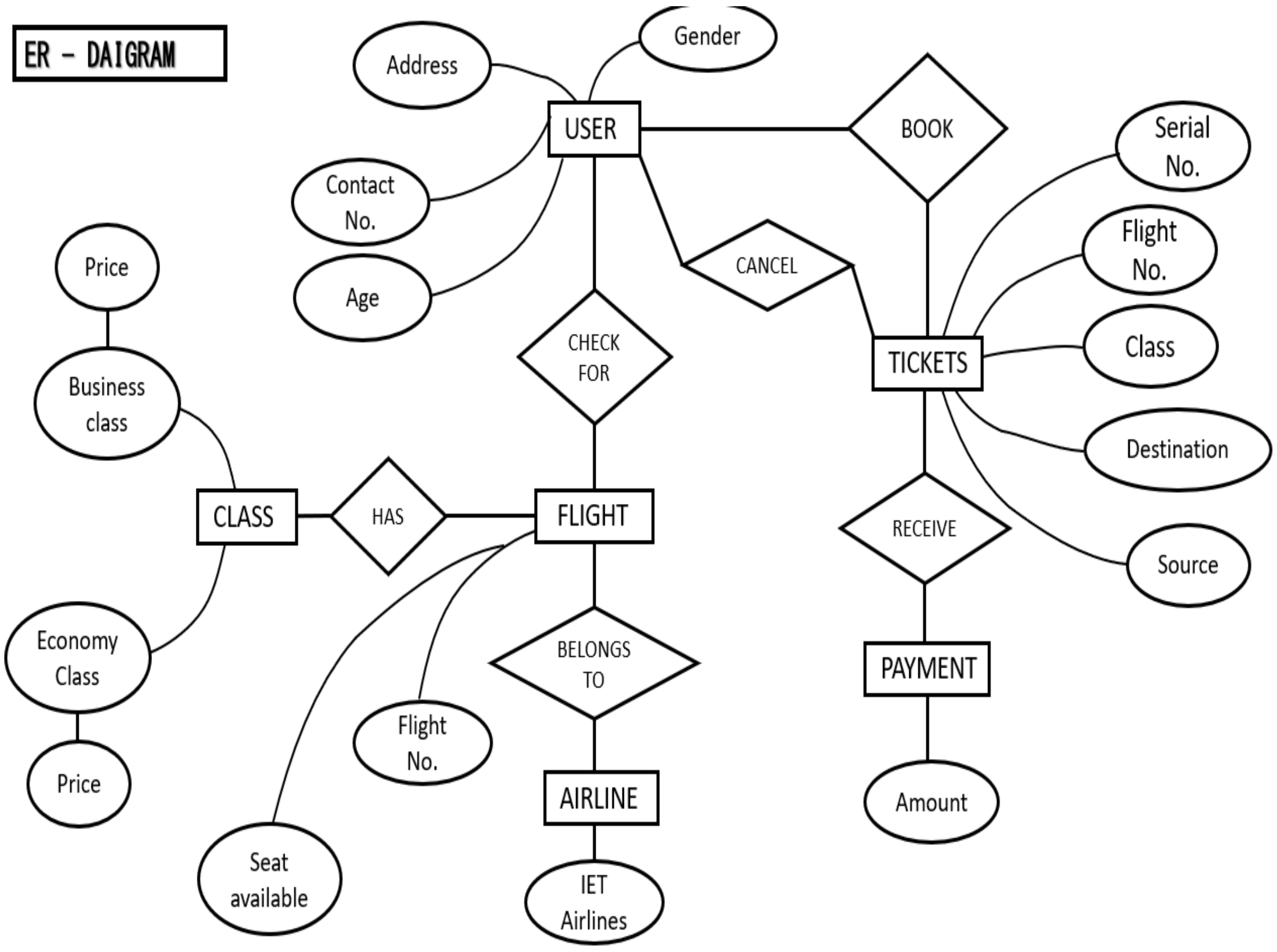
1. **Entities:** Entities represent real-world objects or concepts within the system that are to be stored in the database. Each entity is depicted as a rectangle in the diagram, with the entity's name written inside. Examples of entities could include "Customer," "Product," "Employee," etc.

2. **Attributes:** Attributes are the properties or characteristics of entities. They describe the data that is stored for each entity. Attributes are depicted as ovals connected to their respective entities by lines. For example, attributes of a "Customer" entity might include "CustomerID," "Name," "Email," etc.

3. **Relationships:** Relationships define how entities are related to each other within the system. They describe the associations or connections between entities. Relationships are depicted as diamond shapes connecting the related entities, with lines indicating the cardinality and participation constraints of the relationship

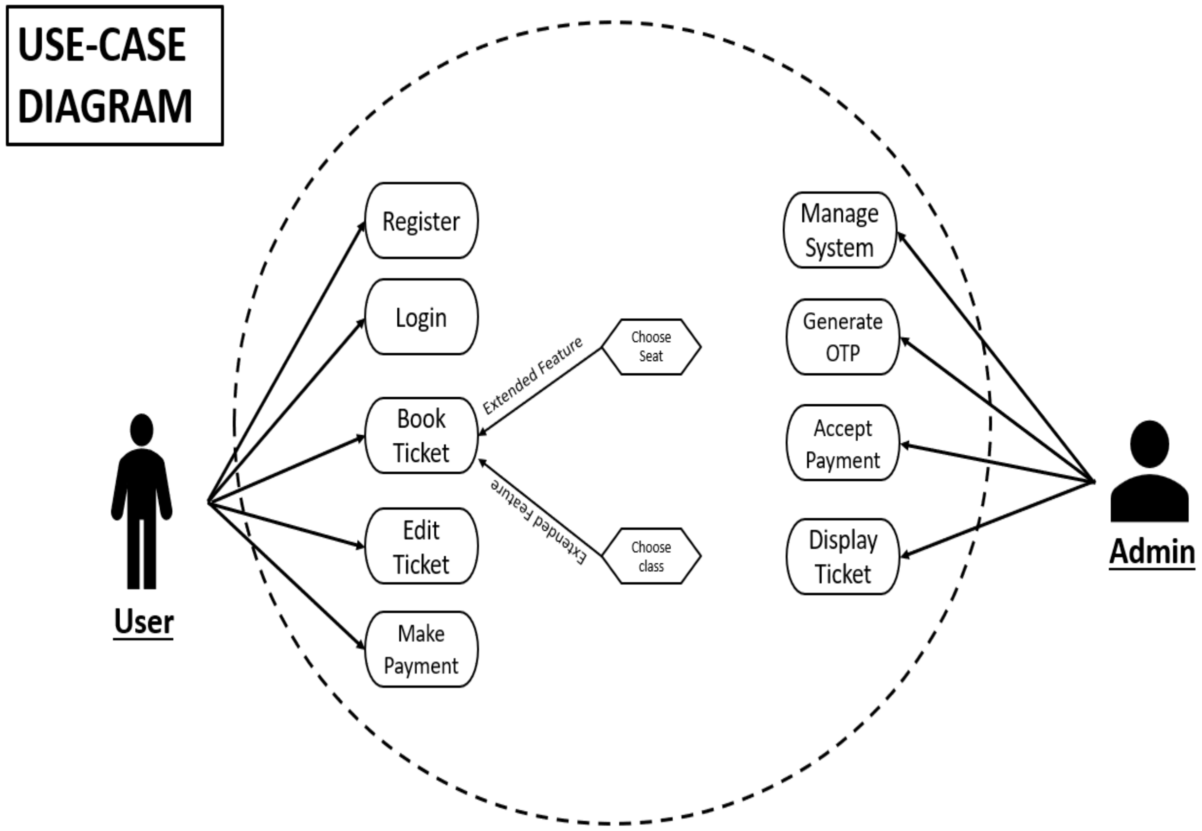
4. **Cardinality:** Cardinality specifies the number of instances of one entity that can be associated with instances of another entity through a relationship. It is represented by the symbols "1" (one) or "N" (many) on each side of the relationship line.

5. **Participation Constraints:** Participation constraints specify whether each entity in a relationship is required to participate (total participation) or optional (partial participation) in the relationship. They are represented by a solid line (total participation) or a dashed line (partial participation) connecting the entity to the relationship diamond.



# PRACTICAL:-5

A Use Case Diagram is a graphical representation used in software engineering to model the interactions between actors (users or external systems) and the system being designed or developed. It illustrates the various use cases or functionalities of the system and how actors interact with them.



Key components of a Use Case Diagram include:

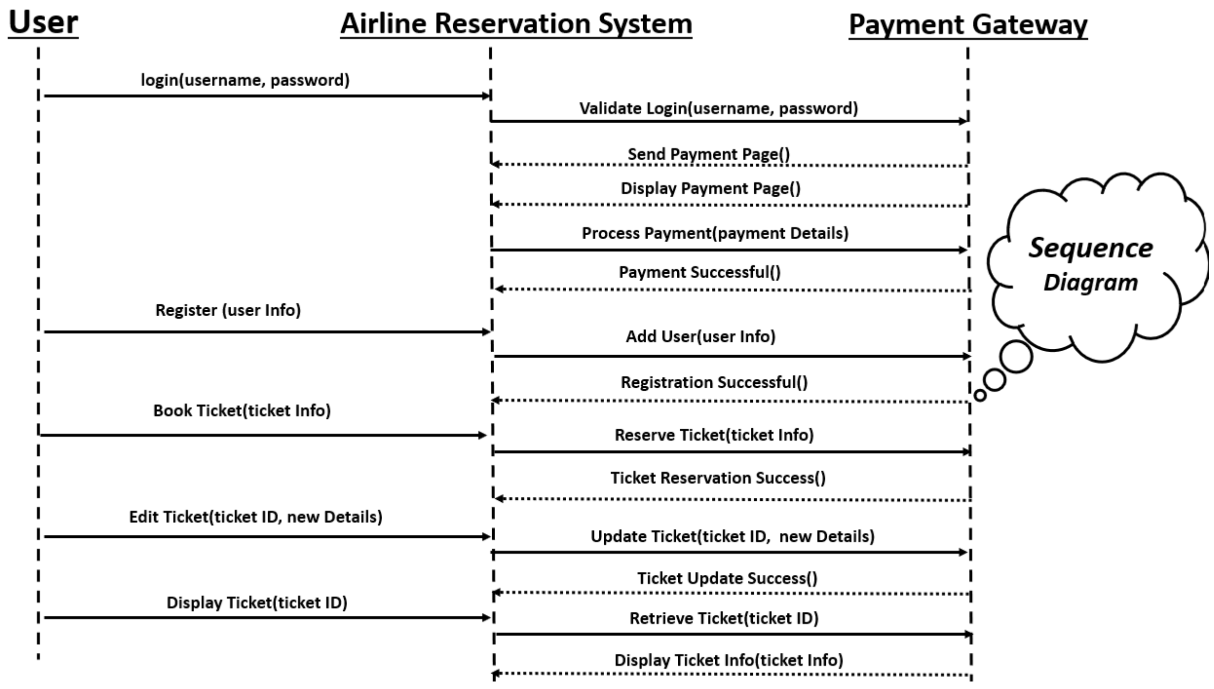
1. **Actors:** Actors represent the users or external systems that interact with the system. They are depicted as stick figures or icons outside the system boundary. Examples of actors could include "Customer," "Administrator," "External API," etc.

2. **Use Cases:** Use cases represent the specific functionalities or actions that the system performs to achieve a particular goal or deliver value to its users. Each use case describes a specific interaction between an actor and the system. Use cases are depicted as ovals within the system boundary, with the use case name written inside. Examples of use cases could include "Login," "Register," "Book Ticket," "Edit Profile," etc.

Use Case Diagrams help in visualizing the functional requirements of a system, identifying the actors involved, and understanding the interactions between actors and the system. They serve as a communication tool for stakeholders, developers, and designers to discuss and analyze the system's behavior and functionality effectively. Use Case Diagrams also provide a basis for further elaboration and refinement of system requirements and serve as a foundation for system design and development.

# PRACTICAL:-6

A Sequence Diagram is a type of interaction diagram in software engineering that illustrates how objects interact in a particular scenario or sequence of events within a system. It focuses on the chronological order of messages exchanged between objects over time.


User
Airline Reservation System
Payment Gateway
login(username, password)
Validate Login(username, password)
Send Payment Page()
Display Payment Page()
Process Payment(payment Details)
Payment Successful()
Register (user Info)
Add User(user Info)
Registration Successful()
Book Ticket(ticket Info)
Reserve Ticket(ticket Info)
Ticket Reservation Success()
Edit Ticket(ticket ID, new Details)
Update Ticket(ticket ID, new Details)
Ticket Update Success()
Display Ticket(ticket ID)
Retrieve Ticket(ticket ID)
Display Ticket Info(ticket Info)
Sequence Diagram

Key elements of a sequence diagram include:

1. **Objects:** Objects represent the various entities or components within the system that interact with each other. They are depicted as rectangles at the top of the diagram.

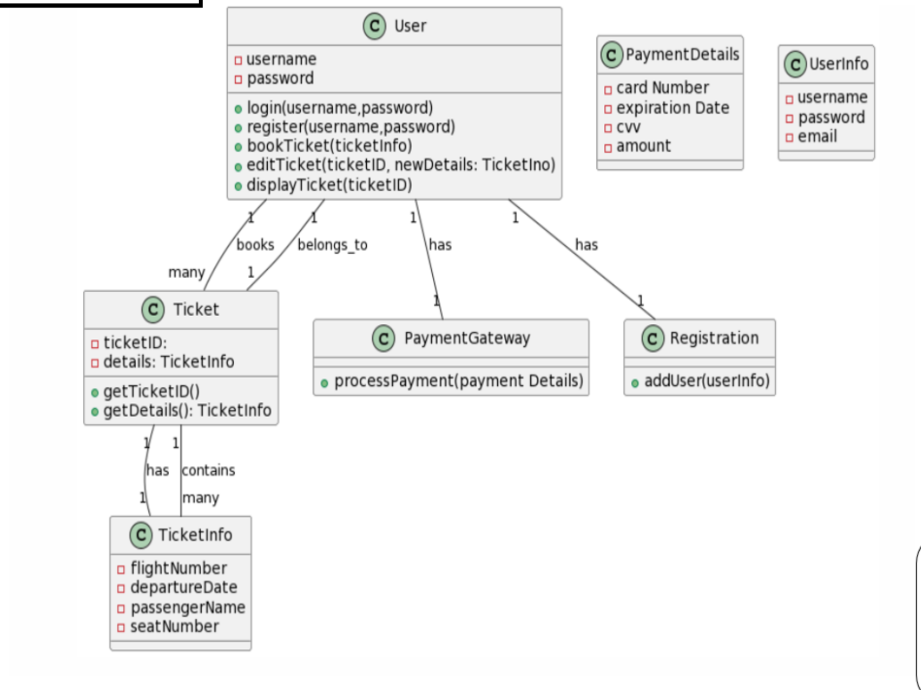
2. **Lifelines:** Lifelines represent the existence of objects over time. They are depicted as vertical lines extending downwards from the objects' rectangles.

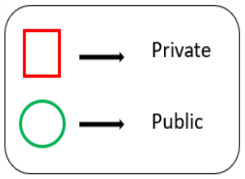
Sequence diagrams help in visualizing and understanding the flow of control and data between objects in a system, as well as in identifying the sequence of events that occur during a particular scenario. They are commonly used during the design phase of software development to model the behavior of system components and to communicate interaction scenarios among team members and stakeholders.

# PRACTICAL:- 7

A class diagram is a static structural diagram used in software engineering to depict the classes, attributes, operations, and relationships in a system. It provides a visual representation of the structure of a system's object-oriented design.

**Class Diagram**





Key elements of a class diagram include:

1. **Class:** A class represents a blueprint for creating objects in the system. It encapsulates data (attributes) and behavior (operations or methods) related to a specific concept or entity in the system.

2. **Attributes:** Attributes are the data members of a class that represent the characteristics or properties of objects belonging to that class. They describe the state of objects and are typically shown as labeled lines inside the class box.

3. **Operations:** Operations, also known as methods, represent the behaviors or functionalities that objects belonging to a class can perform. They define the actions that objects can take and are typically shown as labeled lines inside the class box.

4. **Relationships:** Relationships depict the associations and dependencies between classes in the system. They indicate how classes interact with each other and can include associations (e.g., one-to-one, one-to-many), generalization (inheritance), aggregation (whole-part relationships), and composition (stronger form of aggregation).

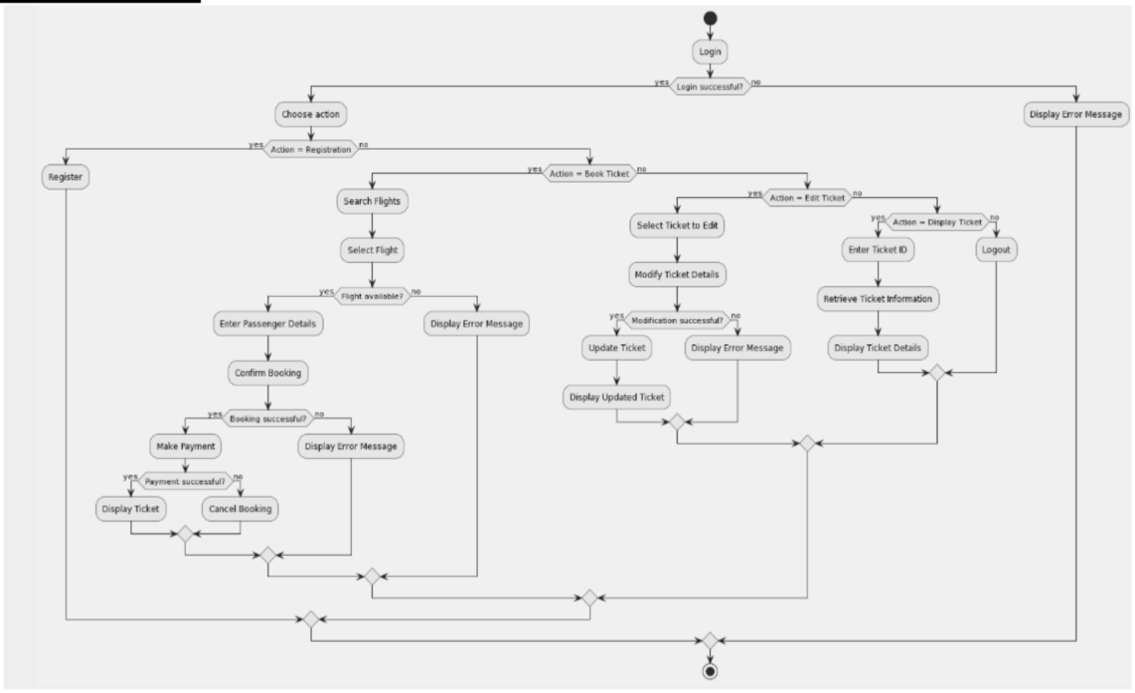
5. **Multiplicity:** Multiplicity specifies the number of instances of one class that can be associated with instances of another class in a relationship. It is represented by numerical values or ranges near the association lines between classes.

Class diagrams help in visualizing the structure and organization of classes in a system, identifying relationships between classes, and understanding the inheritance hierarchy. They are an essential tool for software developers, designers, and stakeholders to communicate and analyze the object-oriented design of a system effectively.

# PRACTICAL:-8

An Activity Diagram is a behavioral diagram used in software engineering to visualize the flow of activities or actions within a system. It depicts the sequence of actions or steps performed by a system, a user, or any other actor to achieve a specific goal or accomplish a task.

**Activity Diagram**



Key components of an Activity Diagram include:

1. **Activities:** Activities represent the tasks, actions, or operations performed within the system. They are depicted as rounded rectangles and are connected by arrows to show the flow of control between them.

2. **Transitions:** Transitions represent the flow of control from one activity to another. They are depicted as arrows and show the sequence in which activities are executed. Transitions can include conditions or triggers that determine when the transition occurs.

3. **Decision Points:** Decision points, also known as decision nodes, represent points in the process where a decision must be made. They are depicted as diamonds and contain conditions or expressions that determine which path the flow of control will take based on the evaluation of the condition.

4. **Start and End Nodes:** Start nodes represent the beginning of the activity diagram, indicating where the process starts. End nodes represent the end of the activity diagram, indicating where the process terminates.

Activity diagrams are useful for modeling the flow of activities in various scenarios, such as business processes, use cases, system workflows, and software behavior. They help stakeholders understand the sequence of actions required to accomplish a task, identify potential bottlenecks or inefficiencies in the process, and ensure clarity and alignment among project team members regarding system behavior and functionality.

------------------------------------------------------------**THE END**------------------------------------------------