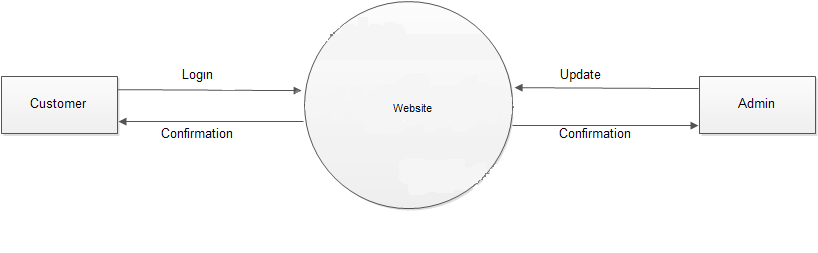
**DFD :**

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

0-level DFD It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows.

1-level DFD In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into sub-processes.

2-Level DFD 2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning. Steps in developing DFDs 1. List business activities to identify processes, external entities, data flows, and data stores 2. Create a context diagram 3. Create the next level diagram 4. Create child diagrams.



**USE CASE :**

**Use case diagrams belong to the category of behavioral diagram** of UML diagrams. They present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform one or more actors, and dependencies among them.

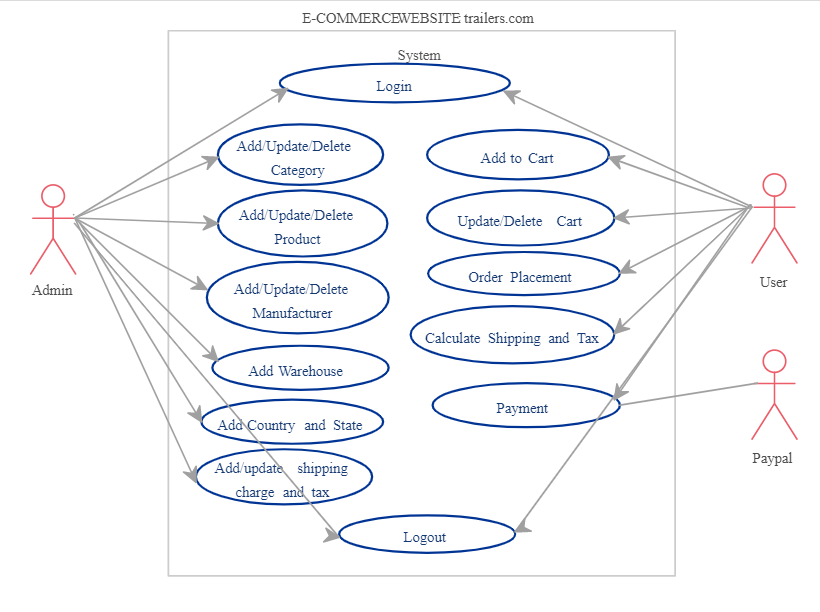
**Actor**

An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.

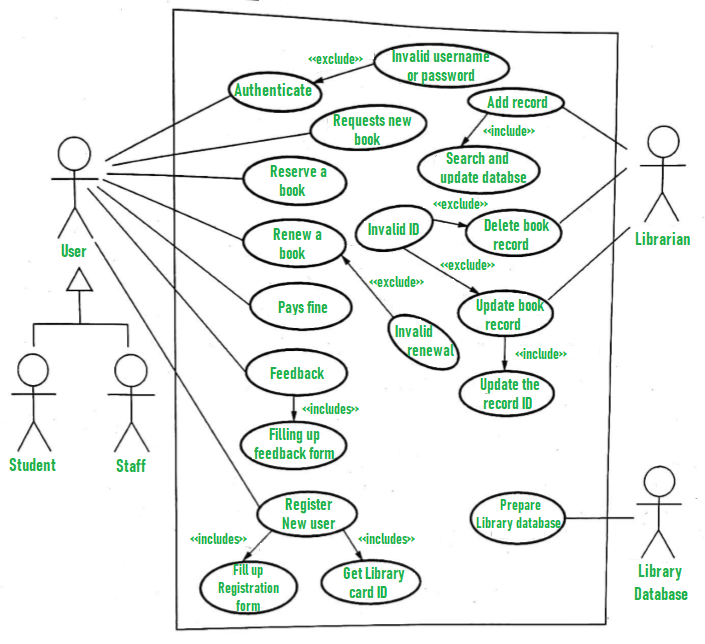
For example, consider the case where a customer *withdraws cash* from an ATM. Here, customer is a human actor.

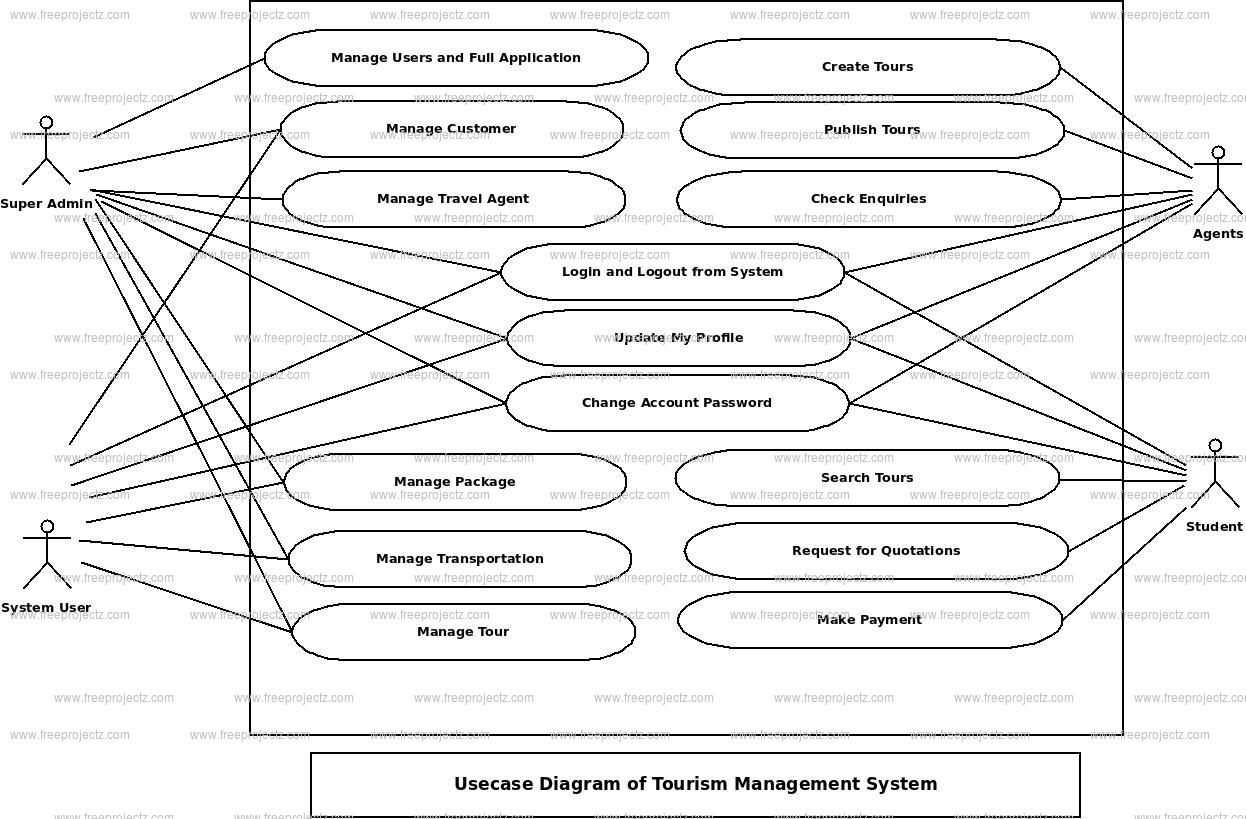
Actors can be classified as below:

* **Primary actor**: They are principal users of the system, who fulfill their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.
* **Supporting actor**: They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.



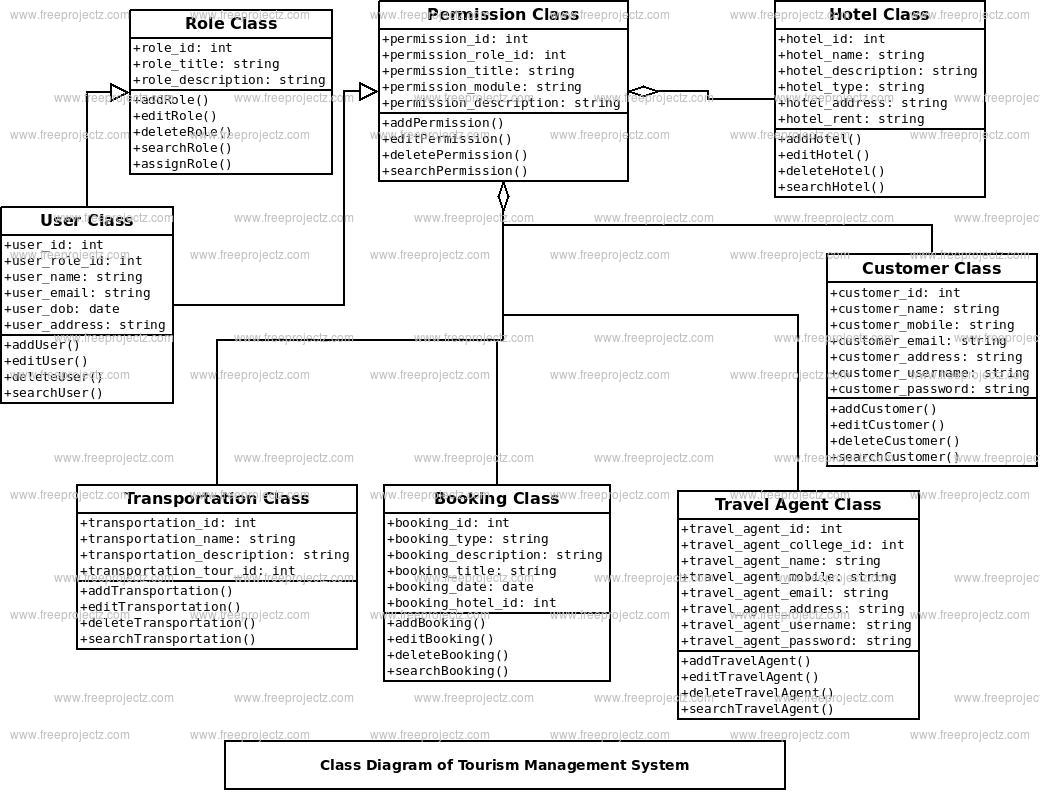


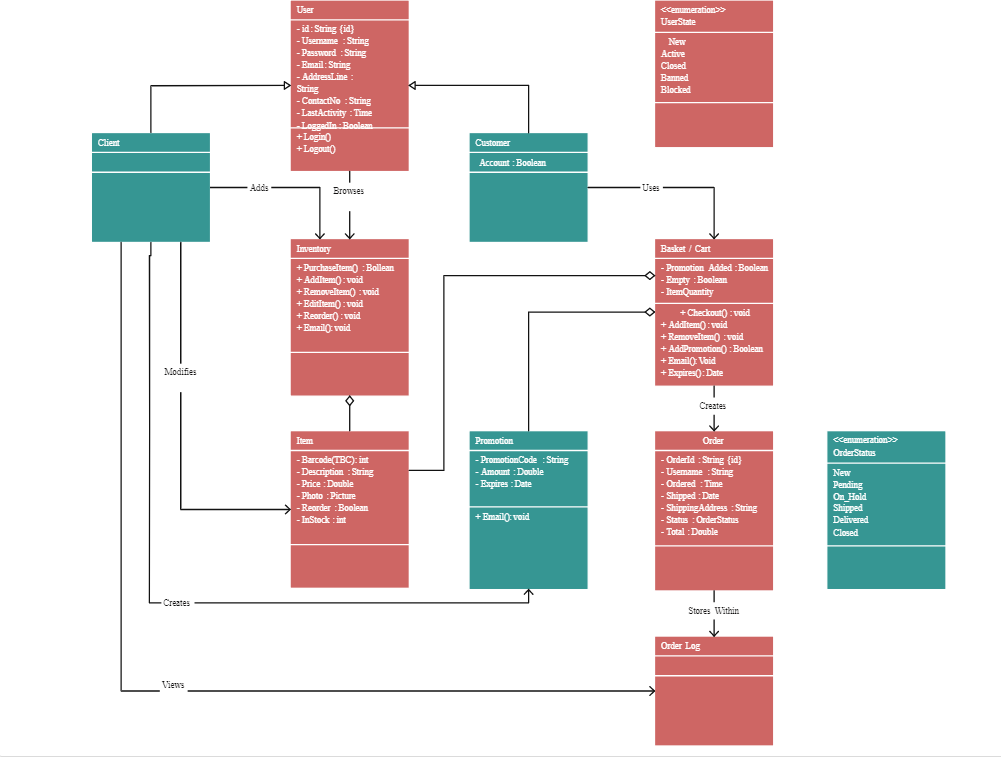




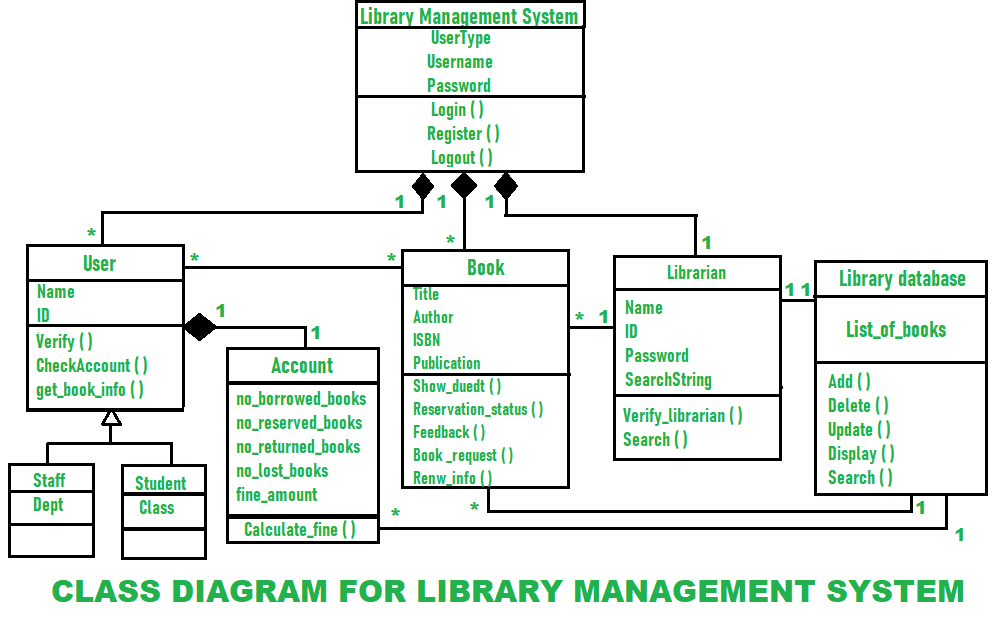
**Class Diagram :**

Class Diagrams: Classes are the structural units in object oriented system design approach, so it is essential to know all the relationships that exist between the classes, in a system. All objects in a system are also interacting to each other by means of passing messages from one object to another. Elements in class diagram Class diagram contains the system classes with its data members, operations and relationships between classes. • Class: A set of objects containing similar data members and member functions is described by a class. In UML syntax, class is identified by solid outline rectangle with three compartments which contain • Class name: A class is uniquely identified in a system by its name. A textual string [2]is taken as class name. It lies in the first compartment in class rectangle. • Attributes: Property shared by all instances of a class. It lies in the second compartment in class rectangle. • Operations: An execution of an action can be performed for any object of a class. It lies in the last compartment in class rectangle • Relationships: Existing relationships in a system describe legitimate connections between the classes in that system. • Association: It is an instance level relationship that allows exchanging messages among the objects of both ends of association. A simple straight line connecting two class boxes represent an association. We can give a name to association and also at the both end we may indicate role names and multiplicity of the adjacent classes. Association may be uni-directional. • Aggregation: It is a special form of association which describes a part-whole relationship between a pair of classes. It means, in a relationship, when a class holds some instances of related class, then that relationship can be designed as an aggregation • Multiplicity: It describes how many numbers of instances of one class is related to the number of instances of another class in an association Class diagrams are the most popular UML diagrams used for construction of software applications. It is very important to learn the drawing procedure of class diagram.Class diagrams have a lot of properties to consider while drawing but here the diagram will be considered from a top level view.





**E-COMMERCE**



**State / Activity Diagram :**

**State-chart Diagrams**

In case of Object Oriented Analysis and Design, a system is often abstracted by one or more classes with some well-defined behaviour and states. A *state chart diagram* is a pictorial representation of such a system, with all its states, and different events that lead transition from one state to another.

To illustrate this, consider a computer. Some possible states that it could have are running, shutdown, hibernate. A transition from running state to shutdown state occur when user presses the "Power off" switch or clicks on the "Shut down" button as displayed by the OS. Here, clicking on the shutdown button, or pressing the power off switch act as external events causing the transition.

State-chart diagrams are normally drawn to model the behavior of a complex system. For simple systems this is optional.

**State**

A state is any "distinct" stage that an object (system) passes through in it's lifetime. An object remains in a given state for finite time until "something" happens, which makes it to move to another state.  All such states can be broadly categorized into following three types:

* Initial: The state in which an object remains when created
* Final: The state from which an object does not move to any other state [optional]
* Intermediate: Any state, which is neither initial, nor final

**Transition**

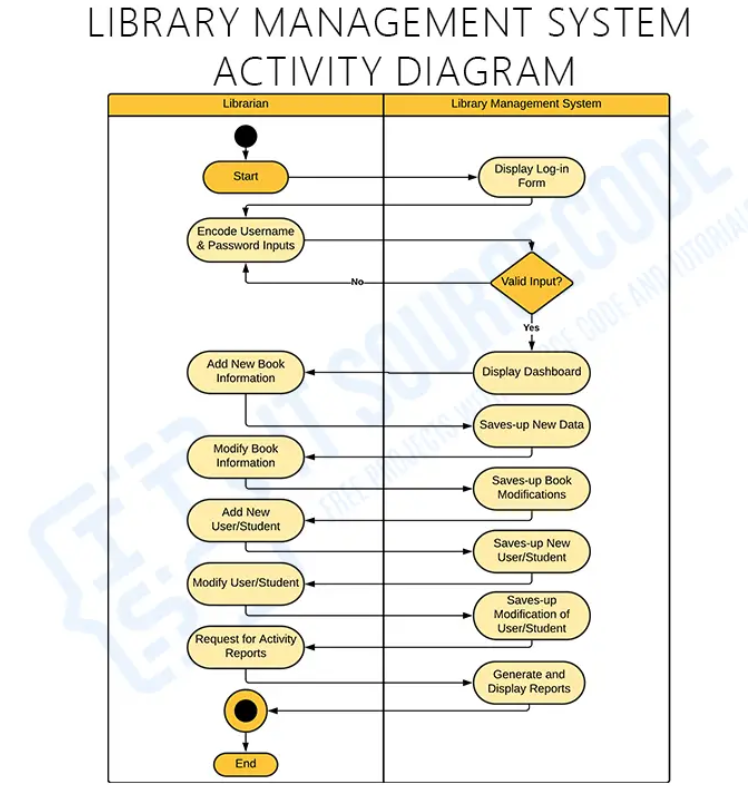
* Transition is movement from one state to another state in response to an external stimulus (or any internal event). A transition is represented by a solid arrow from the current state to the next state.

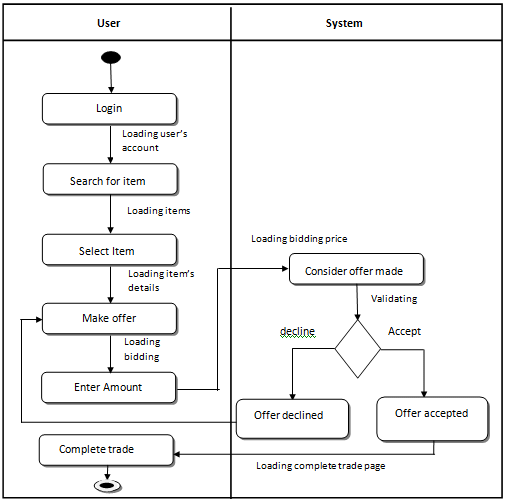
**Action**

* An action represents behavior of the system. While the system is performing any action for the current event, it doesn't accept or process any new event.

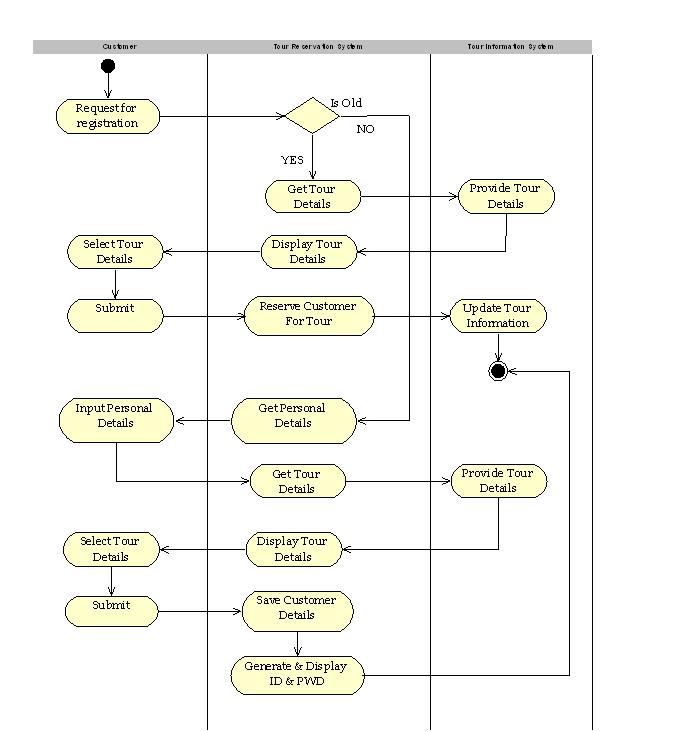
**Activity Diagrams**

* Activity diagrams fall under the category of behavioral diagrams in Unified Modeling Language. It is a high level diagram used to visually represent the flow of control in a system. It has similarities with traditional flow charts. However, it is more powerful than a simple flow chart since it can represent various other concepts like concurrent activities, their joining, and so on.
* Activity diagrams, however, cannot depict the message passing among related objects. As such, it can't be directly translated into code. These kinds of diagrams are suitable for confirming the logic to be implemented with the business users. These diagrams are typically used when the business logic is complex. In simple scenarios it can be avoided entirely.

****



**E-COMMERCE**



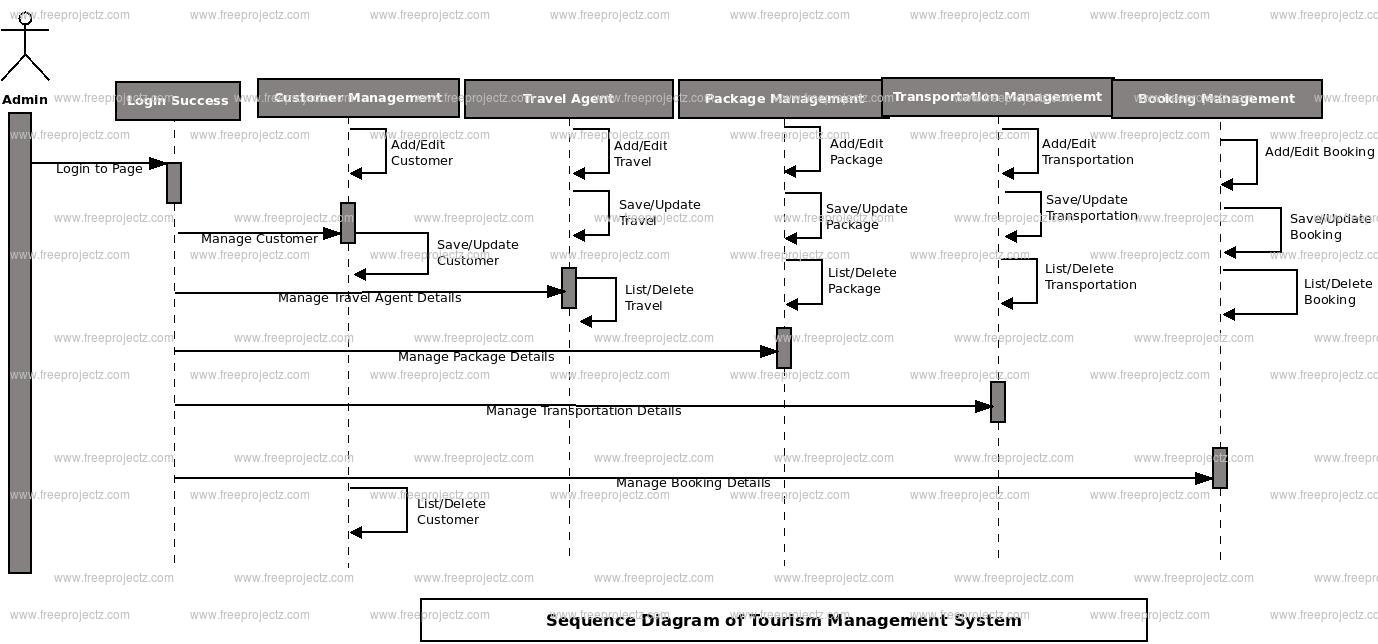
**SEQUENCE DIAGRAM:**

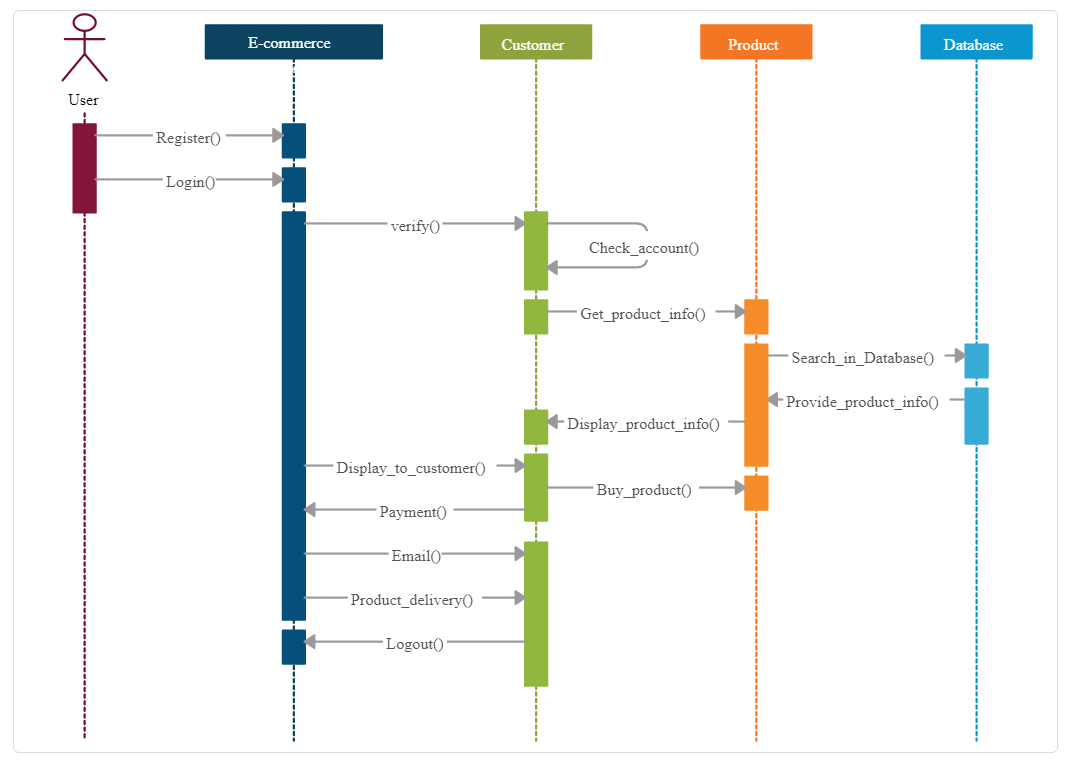
A **sequence diagram** is the most commonly used **interaction** diagram. **Interaction diagram –** An interaction diagram is used to show the **interactive behavior** of a system. Since visualizing the interactions in a system can be a cumbersome task, we use different types of interaction diagrams to capture various features and aspects of interaction in a system. **Sequence Diagrams –** A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

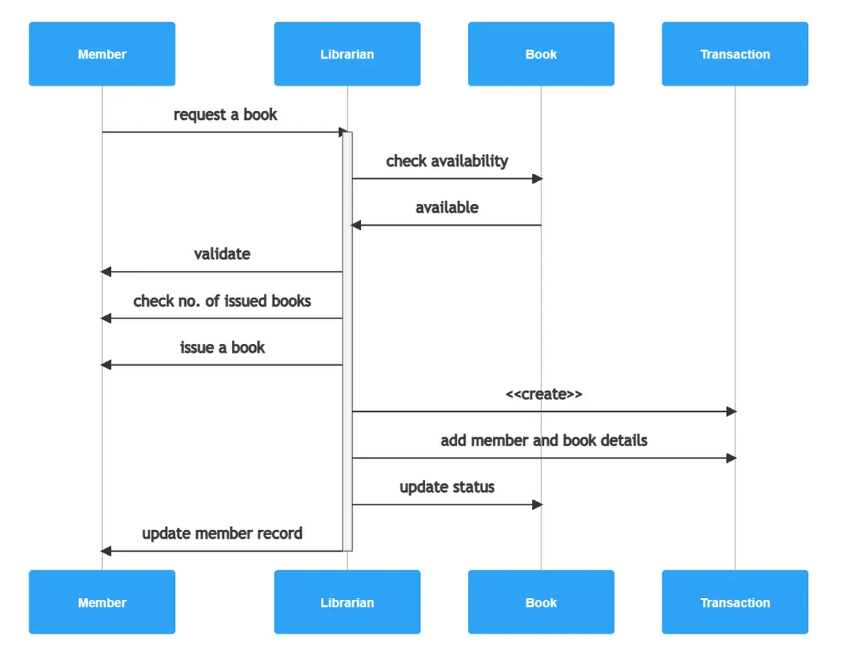
**Benefits of sequence diagrams**

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

* Represent the details of a UML use case.
* Model the logic of a sophisticated procedure, function, or operation.
* See how objects and components interact with each other to complete a process.
* Plan and understand the detailed functionality of an existing or future scenario



****

****

**PROJECT MANAGEMENT TOOL :**

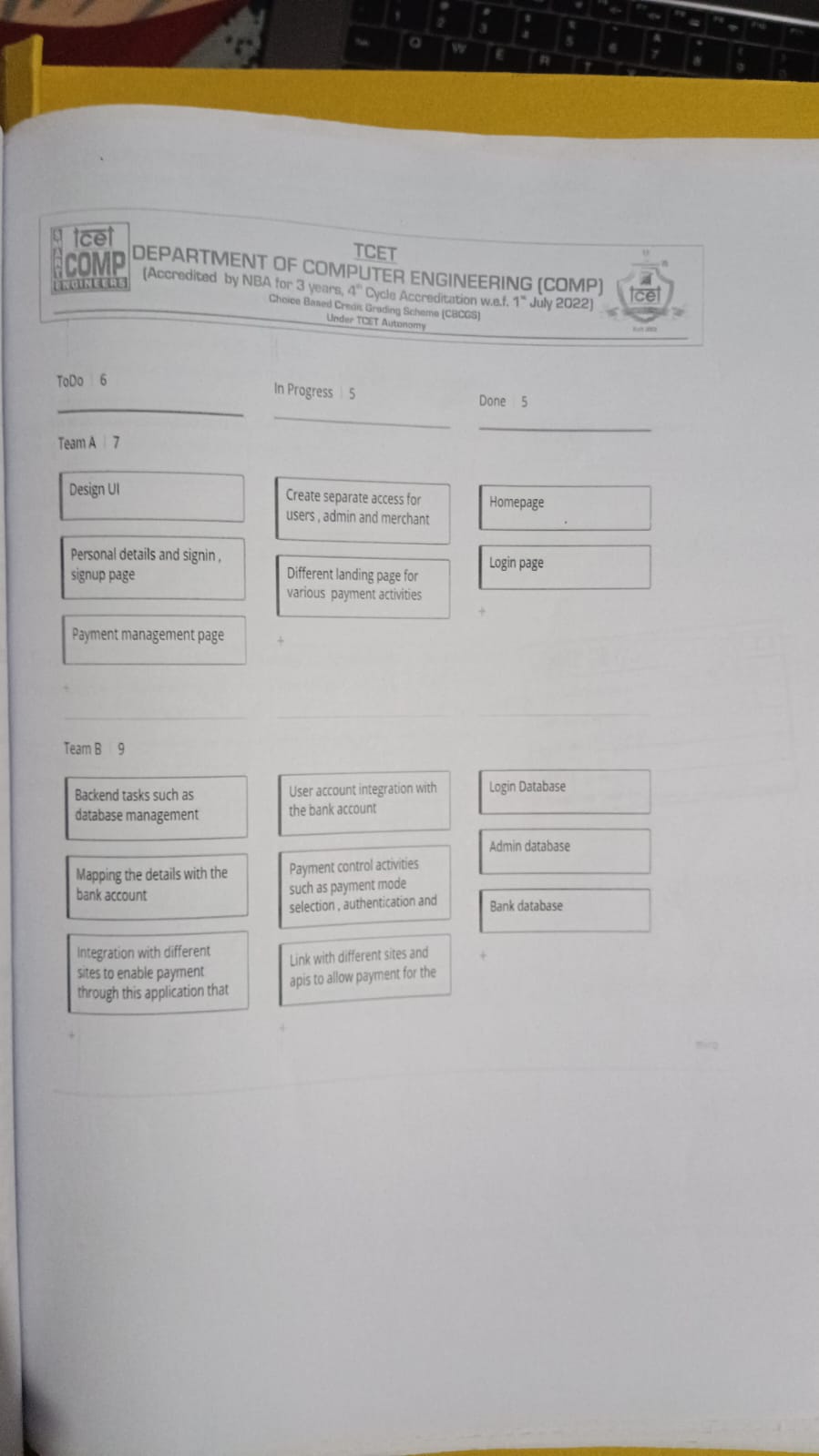
The main aim of PROJECT SCHEDULING AND TRACKING is to get the project completed on time. Program evaluation and review technique (PERT) and Gantt chart are two project scheduling methods that can be applied to software Split the project into tasks and estimate time and resources required to complete each task. Organize tasks concurrently to make optimal use of workforce. Minimize task dependencies to avoid delays caused by one task waiting for another to complete.

**Gantt chart:**

A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity.

This allows you to see at a glance:

* What the various activities are
* When each activity begins and ends
* How long each activity is scheduled to last
* Where activities overlap with other activities, and by how much

****