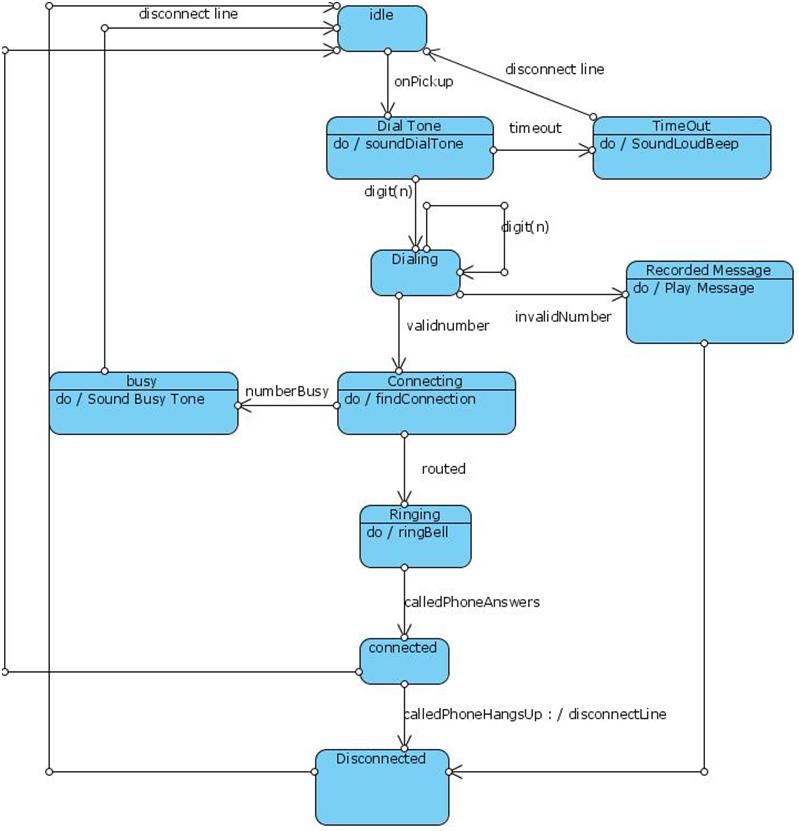
Draw state model for telephone line, with various activities.

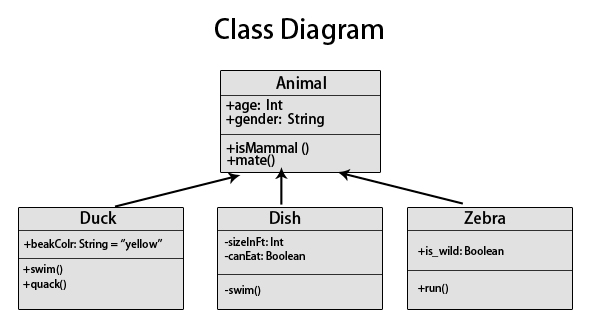


Draw basic class diagrams to identify and describe key concepts like classes, types in your system and their relationships.

<https://www.educba.com/class-diagram/>

## Introduction to Class Diagram

The class diagram is one of the types of UML diagrams which is used to represent the static diagram by mapping the structure of the systems using classes, attributes, relations, and operations between the various objects. A class diagram has various classes; each has three-part; the first partition contains a Class name which is the name of the class or entity which is participated in the activity, the Second partition contains class attributes that show the various properties of the class, the third partition contains class operations which shows various operations performed by the class, relationships shows the relation between two classes.



### Relationships

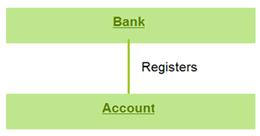
In a class diagram, it is necessary that there exists a relationship between the classes. Unfortunately, the similarity of various relationships often makes it difficult to understand them.

Below are the relationships which exist in a class diagram.

#### 1. Association

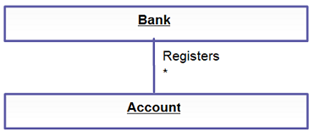
Between two other classes in an association relationship, an association class forms a part of it. Additional information about the relationship could be obtained by attaching the association relationship with the association class. Various operations, attributes, etc., are present in the association class.

The below diagram shows an association between bank and account.



#### 2. Multiplicity

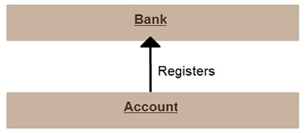
The number of elements or cardinality could be defined by multiplicity. It is one of the most misunderstood relationships which describes the number of instances allowed for a particular element by providing an inclusive non-negative integers interval. It has both lower and upper bound. For example, a bank would have many accounts registered to it. Thus near the account class, a star sign is present.



#### 3. Directed Association

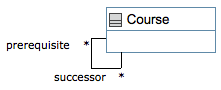
This is a one-directional relationship in a class diagram that ensures the flow of control from one to another classifier. The navigability is specified by one of the association ends. The relationship between two classifiers could be described by naming any association. An arrow indicates the direction of navigation.

The below example shows an arrowhead relationship between the container and the contained.



#### 4. Reflexive Association

The association of a class to itself is known as Reflexive association, which could be divided into Symmetric and Asymmetric type associations. In Symmetric reflexive association, the semantics of each association end has no logical difference, whereas, in Asymmetric Reflexive Association, the associated class is the same, but there is a semantic difference between the ends of the association.



#### 5. Aggregation

In this type of relationship, a more complex object is created by assembling different objects together. The interaction within the different groups of objects is defined by Aggregation. The integrity of the objects is protected, and the response of the assembled objects is decided by the control object. In aggregation, the classes nurture the ‘has a relationship.

Aggregation

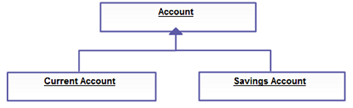
#### 6. Composition

It is a form of aggregation which represents the whole-part relationship. Here, the part classifier lifetime is dependent on the whole classifier lifetime. In a class, a strong life-cycle is represented by the composition relationship. There is usually a one-direction flow of data here. It is generally indicated by a solid line.

Composition

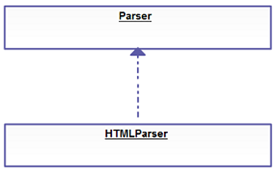
#### 7. Generalization

In this kind of relationship, the child model is based on the parent model. The relationship is used to describe various use-case diagrams and ensures that the child class receives the properties present in the parent. The child model could reuse the attributes of the parent model with the help of the generalization relationship. Hence the distinct attributes need to be defined only in the child; the rest it would inherit from the parent. There could be single parents, multiple children, or multiple parents, single child characteristics in this relationship. There are no names in the generalization relationships. It is also known as the ‘is a relationship.



#### 8. Realization

The behavior of one model element is realized by the specified behavior of another model element. This type of relationship doesn’t have any names.



### Why Should we Use Class Diagram?

The structure of a system is defined by a Class Diagram by showing its attributes, relationships among objects, and so on. It is the backbone of object-oriented modeling and could also be [used for Data modeling](https://www.educba.com/what-is-data-modeling/). Class Diagrams help in making pre-plans which eases the programming process. Moreover, you could always make changes to the Class Diagram as it’s kind of annoying to code different functionality after facts. It is a design plan based on which a system is built. It is easy to understand without much technical knowledge required.

Class Diagram provides a static view of the application, and its mapping ability with object-oriented language makes it ready to be used in construction. Unlike the sequence diagram, activity diagram, etc., the class diagram is the most popular UML diagram.

Below is the purpose of a Class diagram.

* The static view of an application is designed and analyzed.
* A system’s responsibilities are described by it.
* The components and the deployment diagram’s base is the class diagram.
* The forward and reverse engineering is influenced by the Class Diagram.

### Types of Class Diagram

Class Diagram could be divided into three components:

* The Upper Section consists of the class name and is a mandatory component.
* The middle section described the class qualities and used them while describing a class’s specific instance.
* The bottom section describes class interaction with the data.

Moreover, a UML is divided into Behavioural and Structural Diagram with Class Diagram falling under the Structural diagram.

### Advantages of Class Diagram

A class diagram could be implemented in different phases of a project and is the heart of the UML. A representation of reality is created by the class diagram by appearing on the domain model during analysis. The software modeling is done during the design phase, whereas the code is generated during the implementation phase. The foundation of software products is the class diagrams which are an essential part of any project.

A sense of orientation is given by the class diagrams. The structure of the system is analyzed in detail by the class diagram, and also the synergy among different elements is overviewed by them along with their properties. It is fast and easy to read and could be created easily if the right software is in place. Any system that needs to be created, the class diagrams form the foundation for that.

### Benefits

* Any simple or complex data model could be illustrated using the class diagram to gain maximum information.
* The schematics of an application could be understood with the help of it.
* Any system need could be visualized and passed across the business for specific action to be taken.
* Any requirement to implement a specific code could be highlighted through charts and programmed to the described structure.
* A description that is implementation-independent could be provided and passed on to the components.

### Disadvantages of Class Diagram

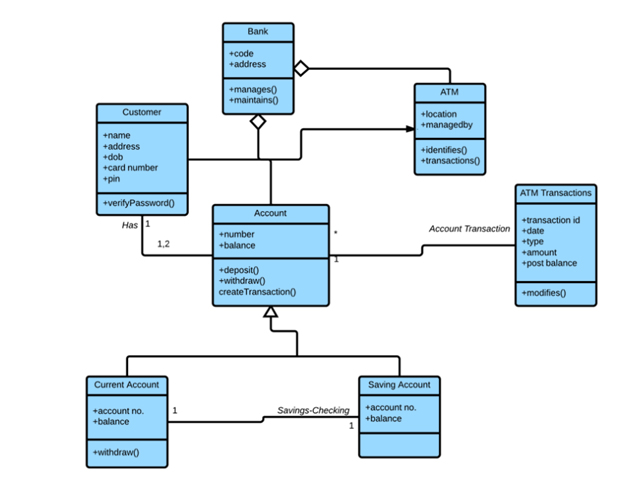
Though Class Diagram is the first thing to consider in a production environment to build a flawless system, it certainly has its fair share of cons as well.

* The class diagrams might often take a longer time to manage, and maintain which is sometimes annoying for a developer. It requires time for the synchronization with the software code to set it up and maintain. Often developers or small companies find it difficult to synchronize the code as it required an added amount of work.
* A lack of clarity in understanding the beneficiary of the diagram is also a disadvantage. As software developers work with code, sometimes the class diagrams are not that helped much. However, project managers could be benefited from the diagrams as it gives an overview of the workflow of a particular tool. Hence, there is often an argument to not waste time on the class diagrams and focus rather on using a whiteboard or paper to draw the diagram.
* An overcomplicated or overwhelming diagram doesn’t help software developers in their work. There could be situations when the developers are frustrated due to the structure of the class diagrams. Mapping out every single scenario could make the diagram messy and hard to work with. Using high-level information could somehow help to combat such issues.
* Putting overemphasis on the design could cause a hindrance to the developers and companies. The stakeholders could easily overanalyze the problems after looking into the class diagram, and putting too much effort into the features of software might lead to a loss of focus. People need to get down on the actual work rather than spending time looking into the diagram and solving issues.

As you can see, despite the importance of the Class Diagram in the [software development life cycle](https://www.educba.com/what-is-sdlc/), it is certainly not without any shortcomings and could make life difficult for the developers and companies if not used wisely.

### Example of Class Diagram

Without the fuss of technical constraints, a diagram is fairly easy to create. To use an ATM, it is only required for a customer to press a few buttons to get their cash. Despite the ease with which the cash flows out, the backend system has multiple layers of security which needed to be passed to prevention in fraud, money laundering, and so on.



As seen over here, there are several entities that follow the properties of different relationships as described earlier. These relationships describe the structure in which an ATM system is built and the layers of security it has to pass through to ensure transparency and integrity in the transaction.

There are three perspectives in which the class diagram could be divided:

* First is the conceptual perspective which the real-world objects are described with the help of conceptual diagrams. The domain under study is represented by the diagram. It is independent of language and is class-related.
* The software components are described by the Specification perspective with interfaces and specifications. In the case of the specific implementation, however, no commitment is given.
* A specific language implementation could be done with the Implementation perspective class diagrams.

### Working with Class Diagram

For software development, the most important UML diagram is the Class Diagram.

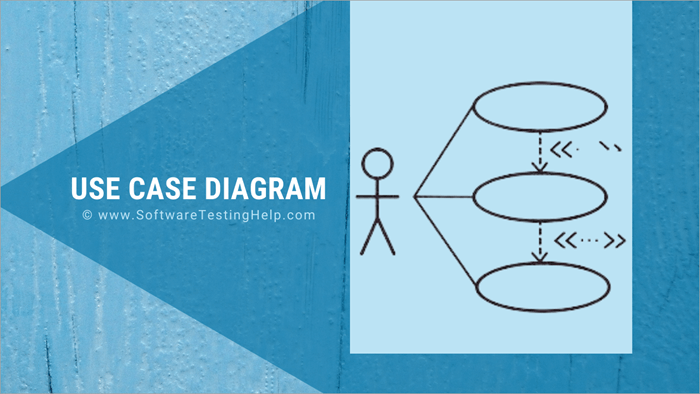
To draw a class diagram representing various aspects of an application, a few of the properties which needs to be considered are:

* A meaningful name should be given to a class diagram describing a system’s real aspect.
* It is necessary that in advance, one understands the relationship between each element.
* To develop a better product, the responsibility among the classes needs to be recognized.
* To avoid making the diagram complicated, the specific properties of a class should be specified.
* Documentation is a good practice in any software development project. Thus defining any aspect in a diagram needs proper documentation or notes for others to comprehend. A software development team at the end should understand what has been configured in the diagram.
* Drawing on a whiteboard or plain paper is needed before the creation of the final version. However, one needs to ensure that only the diagram that is ready should be submitted, which might include several reworks.

Draw one or more Use Case diagrams for capturing and representing requirements of the system. Use case diagrams must include template showing description and steps of the Use Case for various scenarios.

<https://www.softwaretestinghelp.com/use-case-diagram-tutorial/>

**What Is Use Case Diagram**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/Use-Case-Diagram-2.png)

Use Case diagram represents the system’s functionality connecting all four perspectives, i.e. design, implementation, process, and deployment. For every single functionality representation, a fresh diagram is used. Hence multiple use case diagrams represent the complete system.

**Objective Of UML Use Case Diagrams**

The main purpose is to present all functional requirements of the system diagrammatically to all the users who can access the functionality. The presentation is from the perspective of all users giving a high-level design and basic flow of events of the system.

It represented the collaboration and interdependence of the functionality and users in a very easy and understandable manner. The observable outcome of the functionality to the actor and other stakeholders of the system is shown with clarity.

It also presents the functionality’s exceptions, pre-condition, and post-condition. The diagrams do not give the details of deployment, the trigger of the event, etc.

**Benefits**

**The benefits are as follows:**

1. Using a Case diagram is a functional requirement documentation technique. It elicits the functionality as a black box with all the users who have access or a role in it.
2. They are presented in a simple and non-technical way, easy to understand by all technical and business users.
3. They bring customers, and all other users on the same page, making communication easy.
4. It presents a large complex project as a set of small functionalities.
5. It is presented from the end user’s perspective, making it easy for the developers to understand the business purpose.
6. The association presented between actors and other external applications brings clarity to the validations and checking required for the wholesome verification of the system.
7. Using Case driven project development and tracking approach help in assessing the progress of the project from a functionality readiness point of view. The key development activity status enables the project heads to present the readiness from a customer deliverable point of view.
8. The project development can be prioritized as per key deliverable functionalities facilitating better control and management of project revenue.

**Components**

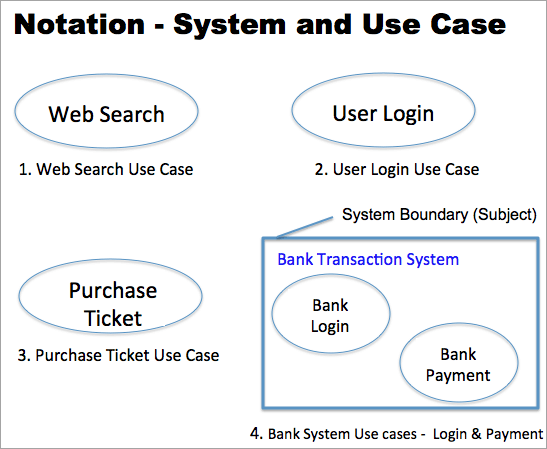
**Listed below are some important components of Use Case diagrams:**

**#1) System:**It is also referred to as scenario or functionality. It details a set of actions between actors and the data consumed and produced if any. Notation of System Boundary (Subject) is a rectangle with the System’s name on top of the rectangle.

All use cases or functionality of the specific system are located inside the rectangle. The actors accessing the system are placed outside the system boundary.

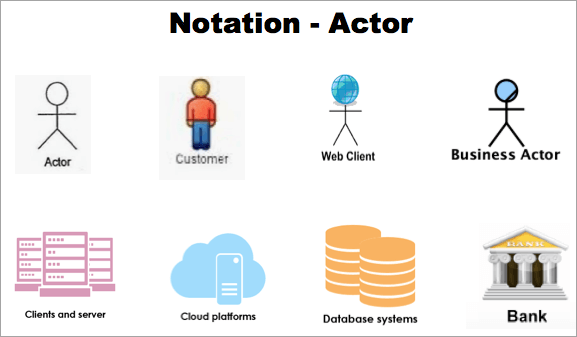
**#2) Use Case:**It represents a functional unit of a large application. Notation is horizontally shaped oval and is located inside the System boundary rectangle indicating that the use case applies to the mentioned subject. A specific use case can be referred to by other systems as well.

So the system is not the owner of the use case. The interactions and actions between events, actors, and the data lead to the end result which is the Use Case goal.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/1Use-Case.png)

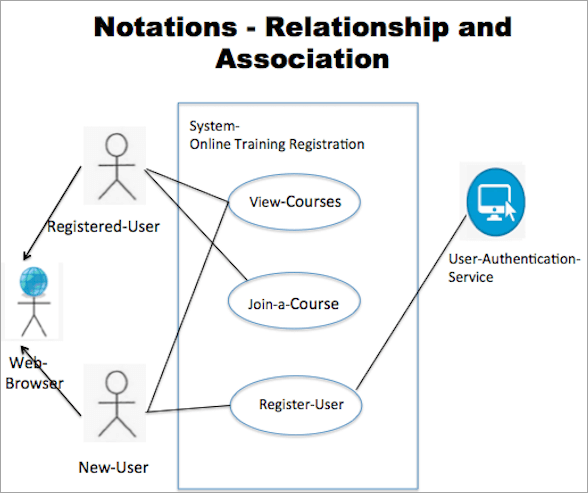
**#3) Actor: The actor** is the entity that interacts with the subject. The actor is external to the subject and hence lies outside the system’s boundary. Actors’ naming should represent the role they play in the system, e.g. Customer, Student, Web-User, etc. Notation is the “**stick man**” icon with the actor’s name above or below the icon.

Custom icons can also be used to denote actors to represent the actor with more clarity. The actor using the use case services is called the primary actor and the actor maintaining or providing services to the use case is called the supporting actor.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/2Actor.png)

**#4) Relationship and Associations:**The actors and use cases have an association with each other. The notation, a line with an arrow, shows a generalized relationship between the two components. In the example below ‘Registered-User’ and ‘New-User’ are generalized to ‘Web-Browser’.

A line between the use case and an actor denotes a communication link between them. Association between actors and use cases can only be binary. A use case can be linked to multiple actors and an actor could also be associated with multiple use cases.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/3Relationship-and-Associations.png)

**Suggested Reading =>>**[**Entity-Relationship Diagram tutorial**](https://www.softwaretestinghelp.com/er-diagram-tutorial/)

**Multiplicity Of Use Case And Actor**

**The multiplicity of Use Case:**

When a use case can be associated with multiple Actors, then it’s a case of multiplicity of a use case. **For example,** as shown in the above image “Notation- Relationship And Association”, View-Courses’ is associated with two actors–‘New-User’ and ‘Registered-User’.

**The multiplicity of an Actor**

**#1)** Multiplicity of an Actor is an association represented by a number and can be zero to any number.

**#2)** Multiplicity zero – It means the use case may have an instance of no actor.

**#3)** Multiplicity One – It means one actor is a must for the use case.

**#4)** Refer to the diagram of the ‘Online Training Website’ explained below:

* When the course payment use case is processed through cash payment, the bank payment service will not be required. Hence the multiplicity of actor ‘Bank-Payment-Service’ can be 0.
* For accessing ‘View-Course’ one actor ‘New-User’ is a must hence multiplicity of this association is 1.

**#5)** Multiplicity greater than 1 – means there can be multiple actors involved in a use case instance. Multiple actors can be associated concurrently or at different points of time or sequentially.

* The multiplicity of an actor more than one is rare. Consider a use case diagram of a marathon-race game where multiple players run concurrently in a given instance of race. So Multiplicity of the actor (player) will be greater than 1 and concurrent.
* Consider a use case diagram of a chess game. Two players will be associated but sequentially as the steps taken by each player are not in parallel but in sequence in an instance of a chess game.
* In a use case diagram depicting the activity of a single relay-race team, multiple players will be associated but at different points in time. In an instance of race, all team members of one team are active at a different point in time.

**Relationship: Exclude And Include**

**Relationship Extend**

1. Extend is a relationship between two use cases. One is called the extended use case and the other extending use case.
2. It is a directed relationship from the extending to the extended use case.
3. The extended use case is independent and complete on its own and is the owner of the extended relationship.
4. The extended use case has no relevance independently, and it just adds value to the extended use case.
5. Notation is a dashed line with an open arrowhead labeled with the keyword «extend».
6. The Extended Use Case name can have names of all its extending use cases as well.
7. A specific use case can be extended by more than one use case.
8. The extending use case can be extended further also.
9. The condition which triggers the extension use case and the detail of the extension point is mentioned in a comment note and are optional

**Relationship Include**

1. Include the relationship between use cases denotes that the behavior of the included use case is part of the base use case
2. Include helps in breaking a large use case into smaller manageable use cases. A base use case can have multiple included use cases.
3. Include also helps in not repeating a specific behavior, which is commonly referred to by different use cases.
4. The common part is depicted in the included use case and is associated with all the use cases where it is referred.
5. The included use case needs the included use case for completion. So Include cannot be depicted alone.
6. Notation is a dashed arrow with an arrowhead from the included base use case to the included common part use case. The relationship notation is labeled with the keyword «include»
7. An included use case can include another use case. Refer to Example 3 shown below in this tutorial, where Search doc includes Preview doc, which includes Browse docs.

**Refer to the diagram of the ‘Online Training Website’ explained below:**

* For joining a course, the user needs to search the course, select it and make payment. Hence the two use cases ‘View-Courses’ and ‘Course-payment’ are included in the ‘Join-a-Course’ use case.
* ‘View-Courses’ can be accessed by actor ‘New-User’ and also ‘Registered-User’. Hence the use case is separated to enable access to two actors.
* ‘Course-payment’ is separated to make the base use of ‘Join-a-Course’ less complex.

For a better understanding of all the components, please refer to the section “Step by step Guideline to Draw Use Case Diagram”.

**To-do List Before Drawing Use-Case Diagram**

**Listed below are some readiness points before starting to draw a use case diagram to represent a System:**

**#1)** Project broken down into multiple small functionalities

* Understand the complex large project and break it down into multiple functionalities and start documenting the detail of each functionality.

**#2)** Identify the goal and prioritize

* Start listing each functionality identified with the goal to be achieved by the functionality.
* Prioritize the identified functionality as per the business deliverable plan.

**#3)** Functionality Scope

* Understand the scope of the functionality and draw the system boundary.
* Identify all the use cases that need to be part of the system to achieve the goal.
* List all the actors (users and services) that have a role in the system. An actor can be a human, internal, and external application that can interact with the functionality.

**#4)** Identify relationship and association

* Have clarity in the relationships and interdependency between use cases and actors.

**#5)** Identify Extension and Inclusion Use cases

* List all the use cases with extension or Include a use case for it.

**#6)** Identify Multiplicity

* Find multiplicity of Use cases and Actors, if any.

**#7)** Naming Use Case and actors

* Follow a standard in naming the use cases and actors. The name should be self-explanatory.
* The name referred to for a specific user/use case should be the same across the whole project.
* A brief detail of use case functionality and the actors with access to the use case should be summarized under a specific section in the document.

**#8)** Important note points

* Clarify and highlight important points using Notes without overburdening the use case with notes.

**#9)** Review

* Review and validate the document before starting the drawing of the use cases.

The drawing of a specific system Use Case diagram should start only after the above details are documented and approved. An approved system’s drawing can be started while the overall project’s details are still being gathered and documentation is in progress.

**Project Document Sample**

**Refer to the Sample document prepared which is a deliverable.**

* The document helps in preparing for the Use Case depiction of the system, scheduling the Use case drawing, tracking the progress of the development, etc.
* The ‘List of System’ enables to schedule of the System that can be picked for Use Case drawing, i.e. one whose status is approved.
* The ‘List of Use Cases‘ and ‘List of Actors’ detail the use cases and actors in the scope of the system.

**Document Sample**

**Project Name:** Online Training Website

**List of Actors of the Project**

| **Actor Name / User Name** | **Actor Category** | **Role Brief** | **Standard icon** |
| --- | --- | --- | --- |
| **New-User** | Web User | Any Web browser | [new](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/new.png) |
| **Registered-User** | Web User | Customers who have registered (student / ex student / Browsers interested in joining a course) | [new](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/new.png) |
| **Web-User** | Category |  |  |
| **Course-Coordinator** | Internal User |  | [Course-Coordinator](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/Course-Coordinator.png) |
| **Employee-Cashier** | Internal User |  | [Course-Coordinator](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/Course-Coordinator.png) |
| **Bank-Payment-Service** | Service / application |  | [Bank-Payment-Service](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/Bank-Payment-Service.png) |
| **User-Authentication-Service** | Service / application |  | [User-Authentication-Service](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/User-Authentication-Service.png) |

**List of Use Cases/Activities**

| **Use Case Name** | **Brief detail** | **Allowed Actors / Multiplicity number of Actor** | **Extension / Include Use Case** | **Use case Included** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| **Register-User** | Register User details like name, city , contact etc. and provide an Id | 1. New-User / 1 2. User-Authentication-Service / 1 |  |  | Extension point - Registration -help Location-Search-help |
| **View-Courses** | Ability to see latest available courses | 1. New-User / 1 2. Instructors / 1 3.User-Authentication-Service / 1 |  |  |  |
| **Course-payment** |  | 1. Bank-Payment-Service / 0 2. Cashier / 0 |  |  |  |
| **Join-a-Course** |  | 1. Registered-User / 1 | Include | 1. View-Courses 2. Course-payment |  |
| **Registration help** |  | None | Exclude |  | Condition - On click of help link |
| **Location-Search-help** |  | None | Exclude |  | Condition – On click of City help link |
| **Edit Registered User details** |  | 1. Registered-User / 1 2. User-Authentication-Service / 1 |  |  | Extension point – Registration- help |

**List of System (Functionality list)**

| **Functionality / System Name** | **Brief detail of the System** | **Business Priority** | **Approval Status** | **Progress Status** | **Use case Names** | **Allowed Actors** |
| --- | --- | --- | --- | --- | --- | --- |
| **Online Training Registration** | The functionality covers three tasks 1.New user looking at all the available courses 2.Registering user to get notifications etc. 3. Join a course by making payment | 1 | Y | Use Case Diagram to be initiated | 1.View-Courses 2. Register-User 3. Join-a-Course | 1. New-User 2. Registered-User 3. Employee-Cashier 4. User-Authentication-Service 5. Bank-Payment-Service |
| **Course Management** |  | 2 | N | Functional Detail sent for approval |  |  |
| **Instructors Management** |  | 2 | N | Functional Documentation in progress |  |  |

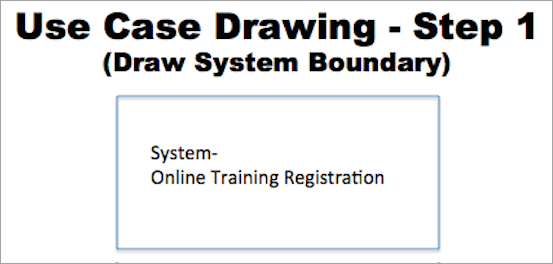
**Draw Use Case Diagram: Step-by-Step Guideline**

The current section explains the step-by-step approach to drawing a Use Case diagram. Refer to the ‘Document Sample’ and select the ‘System’ with the status – Approved i.e. ‘Online Training Registration. Change the status to Use Case Diagram ‘started’ to facilitate progress tracking of each System.

Understand the system by referring to the brief and scope of the System detailed in the ‘List of System’ section of the document.

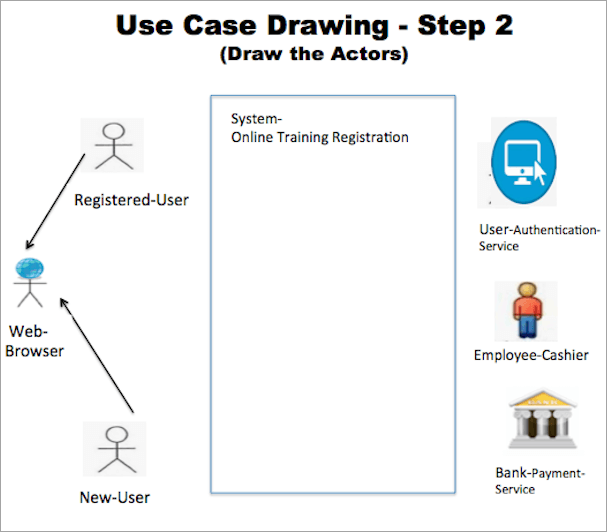
**Step 1:**

* Draw the System Boundary and name the system

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/4Draw-the-System-Boundary-and-name-the-system.png)

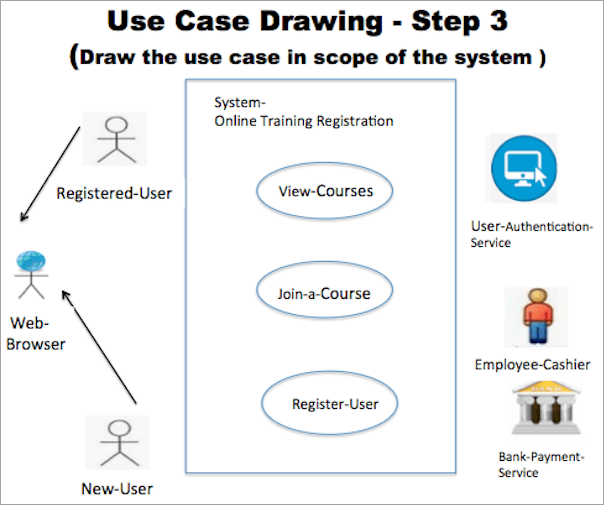
**Step 2:**

* Draw the actors by referring to the column ‘Allowed actors’ in the ‘List of System’ section and name them as per the project standard icon and names as described in the ‘List of Actors’ section of the document.
* The actors ‘New-User’, ‘Registered-User’, and ‘Employee–Cashier’ are the primary actors of the system.
* The other two support service actors, i.e. the ‘Bank-Payment-Service’ and the ‘User-Authentication-Service’ are the supporting actors.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/5supporting-actors.png)

**Step 3:**

Draw the use case in the scope of the system by referring to the column ‘Use Case names’ in the ‘List of System’ section and name the use cases as mentioned in the ‘List of Use Cases‘ section of the document.

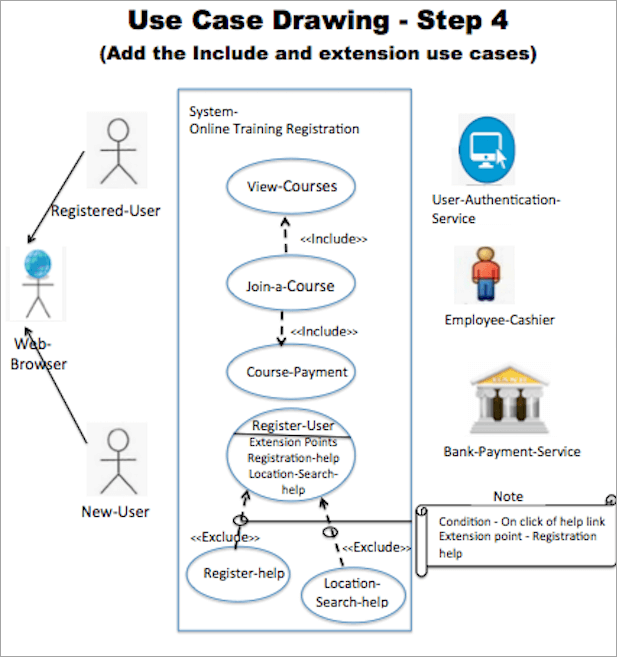
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/6Use-Case-names.png)

**Step 4:**

Add the Include and extension use cases for the in-scope use cases by referring to the ‘List of Use Cases‘ section of the document. ‘Join-a-Course’ includes two Use cases–‘Course-payment’ and ‘View-Courses’. Establish the association with a dash-line starting from the base use case with an arrow pointing to the included two use cases.

Depict ‘Register-User’ with its two extension points with ‘Register-help’ and ‘Location-Search-help’ and associate it with a dashed line and an arrow pointing to ‘Register-User’.

The Note feature can be added as shown in the diagram to give details.

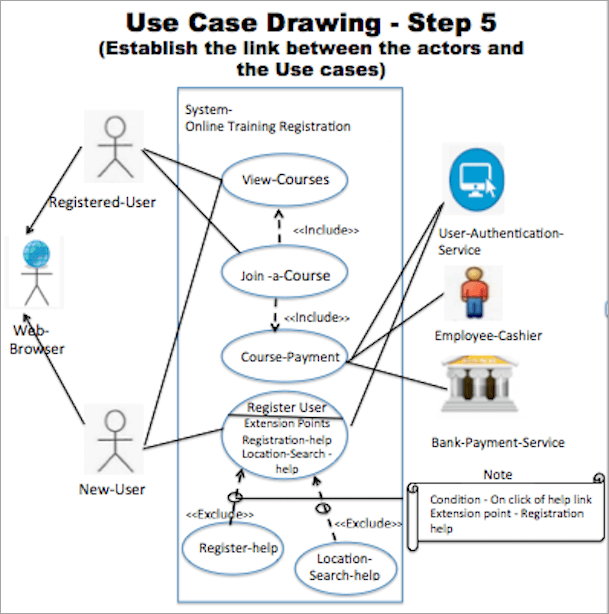
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/7Note-feature.png)

**Step 5:**

Establish the link between the actors and the Use cases. The column ‘Allowed Actors/Multiplicity number of Actor’ in the ‘List of Use Cases‘ section of the document gives all the actors to Use case association.

There can be some actor that is allowed by the Use case but they do not have any role in the current system being depicted. Like the actor ‘Instructor’ that can access use case ‘View-Courses’ but does not have a role in the current system being depicted.

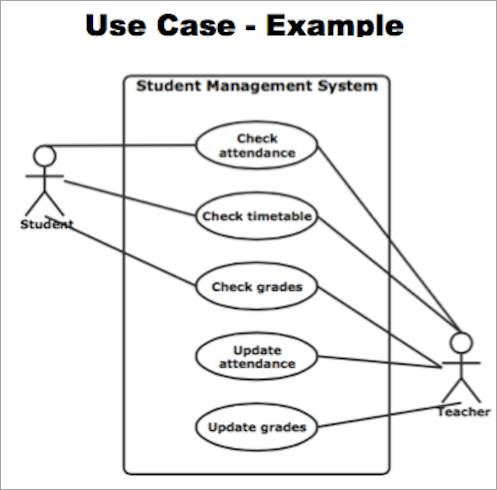
**This completes the ‘Online Training Registration’ system depiction.**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/8online-Training-Registration.png)

**Use Case Diagram Examples**

**Example 1:** This diagram represents a system named Student Management System that has five functionalities in scope.

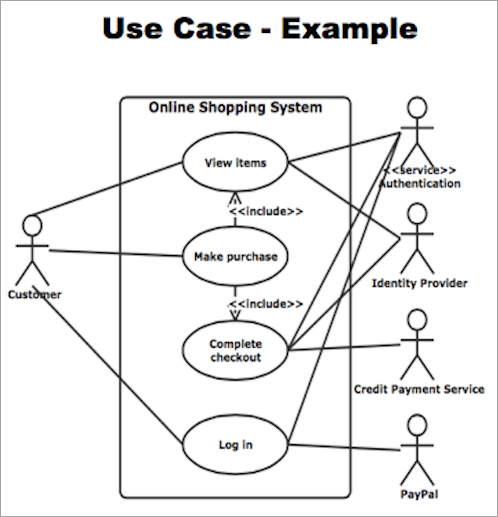
There are two user roles, i.e. Actor who have access to the system. Actors, Teachers, and students have access to functionalities to check timetables, check grades, and check attendance. The access to functionalities update attendance and update grades are only for actor Teachers.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/10Use-Case-Diagram.png)

*[image*[*source*](https://www.edrawmax.com/)*]*

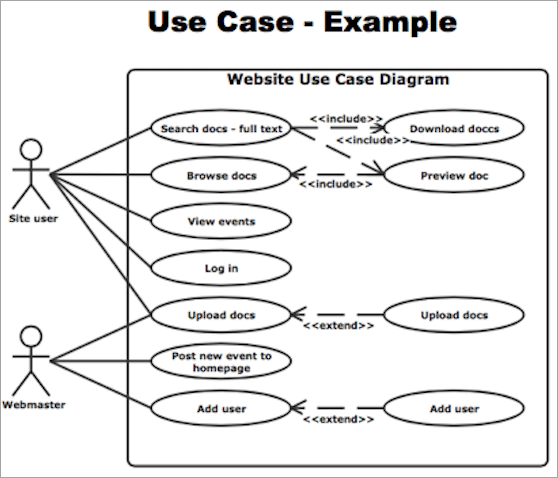
**Example 2:** This diagram represents Online Shopping System that has three independent functionalities in scope. Complete checkout and view items are two included functionality of Make purchase.

The primary actor is the Customer and there are four supporting actors which are services like identity providers, service authentication, and external applications like PayPal, Credit payment services.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/usecase-diagram-ex-3.png)

**Example 3:** This diagram represents a system Website that has 7 functionalities in scope. There are two Actors Webmaster and the Site user. The Search Doc functionality has two included functionalities Preview doc and Download doc.

The Preview doc includes Browse doc functionality. There are two extension points one for each use case Upload doc and Add user.

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2021/07/usecase-diagram-ex-2.png)

Draw activity diagrams to display either business flows or like flow charts

<https://creately.com/blog/diagrams/activity-diagram-tutorial/>

Activity diagrams can be used in all stages of software development and for various purposes. And because they are a lot similar to flowcharts, they are generally more popular than [other UML diagram types](https://creately.com/blog/diagrams/uml-diagram-types-examples/).

In this activity diagram tutorial, we hope to cover everything you need to know about activity diagrams to learn and master it. Scroll down to find;

### What is an Activity diagram?

A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioral diagram that illustrates the flow of activities through a system.

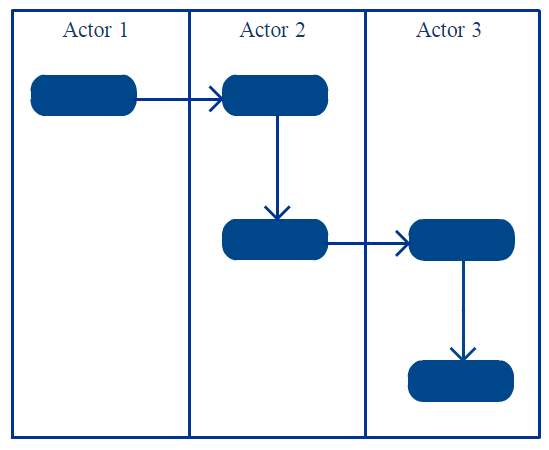
UML activity diagrams can also be used to depict a flow of events in a business process. They can be used to examine business processes in order to identify its flow and requirements.

### Activity Diagram Symbols

UML has specified a set of symbols and rules for drawing [activity diagrams.](https://creately.com/diagram-type/template/gqynfxwy3/activity-chart) Following are the commonly used activity diagram symbols with explanations.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Use** |
| Start | Start/ Initial Node | Used to represent the starting point or the initial state of an activity |
| Activity | Activity / Action State | Used to represent the activities of the process |
| Action | Action | Used to represent the executable sub-areas of an activity |
| Control flow | Control Flow / Edge | Used to represent the flow of control from one action to the other |
| Object flow | Object Flow / Control Edge | Used to represent the path of objects moving through the activity |
| Activity final node | Activity Final Node | Used to mark the end of all control flows within the activity |
| Flow final node | Flow Final Node | Used  to mark the end of a single control flow |
| Decision node | Decision Node | Used to represent a conditional branch point with one input and multiple outputs |
| Merge node | Merge Node | Used to represent the merging of flows. It has several inputs, but one output. |
| Fork | Fork | Used to represent a flow that may branch into two or more parallel flows |
| Merge | Merge | Used to represent two inputs that merge into one output |
| Signal sending | Signal Sending | Used to represent the action of sending a signal to an accepting activity |
| Signal receipt | Signal Receipt | Used to represent that the signal is received |
| Note or comment | Note/ Comment | Used to add relevant comments to elements |

**Activity Diagrams with Swimlanes**



In activity diagrams [swimlanes](https://creately.com/lp/swimlane-diagram-software/" \t "_blank) – also known as partitions – are used to represent or group actions carried out by different actors in a single thread. Here are a few tips you can follow when using swimlanes.

* Add swimlanes to linear processes. It makes it easy to read.
* Don’t add more than 5 swimlanes.
* Arrange swimlanes in a logical manner.

Create and visualize complex activity flows, coordinate efforts with your team, and develop a shared understanding of how systems behave with Creately.

[Create an Activity Diagram](https://creately.com/demo-start/?tempId=jlg33xw41)

### How to Draw an Activity Diagram

Activity diagrams can be used to model business requirements, create a high-level view of a system’s functionalities, analyze use cases and for various other purposes. In each of these cases, here’s how to draw an activity diagram from the beginning.

#### Step 1: Figure out the action steps from the use case

Here you need to identify the various activities and actions your business process or system is made up of.

#### Step 2: Identify the actors who are involved

If you already have figured out who the actors are, then it’s easier to discern each action they are responsible for.

#### Step 3: Find a flow among the activities

Figure out in which order the actions are processed. Mark down the conditions that have to be met in order to carry out certain processes, which actions occur at the same time and whether you need to add any branches in the diagram. And do you have to complete some actions before you can proceed to others?

#### Step 4: Add swimlanes

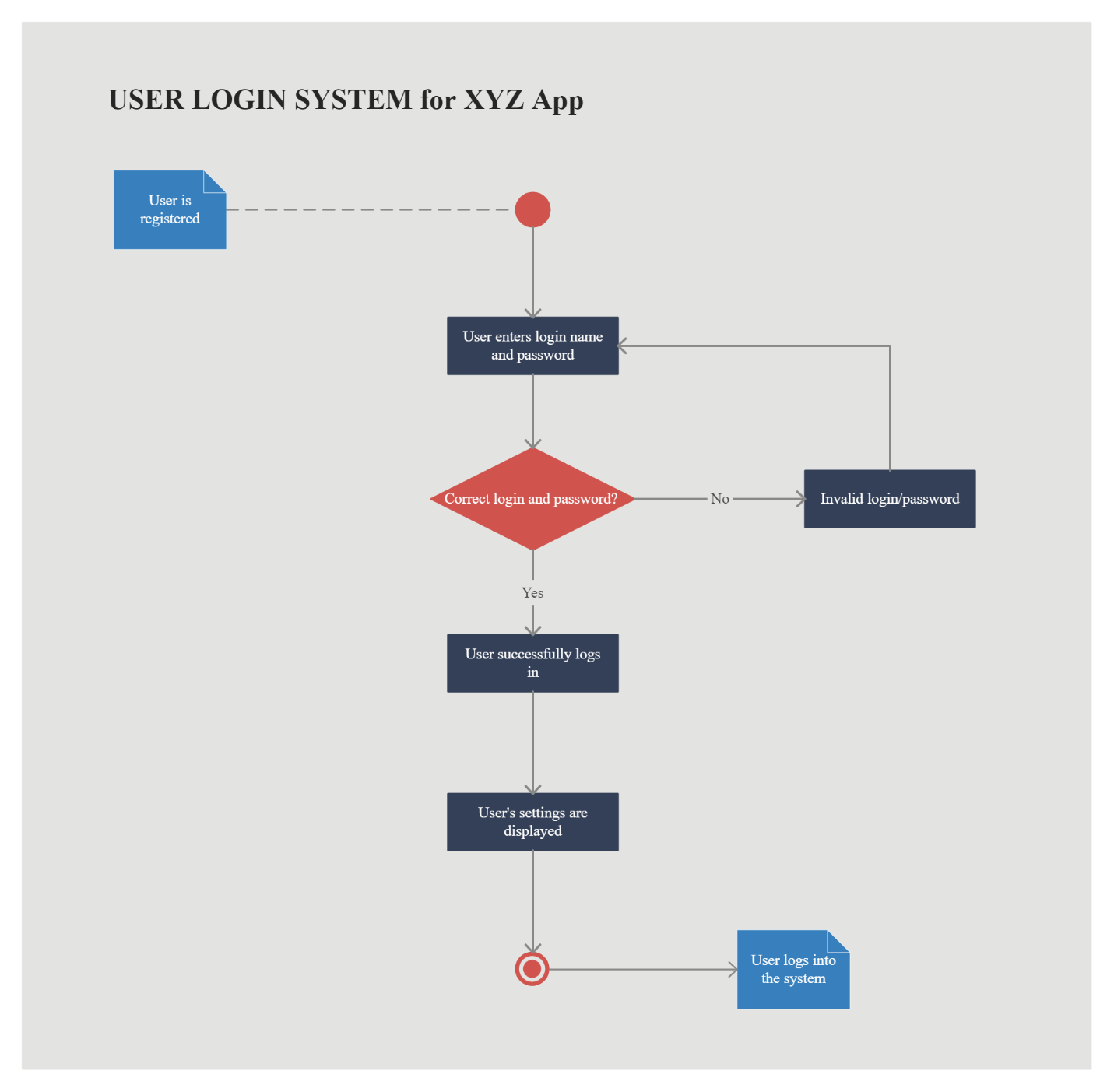
You have already figured out who is responsible for each action. Now it’s time to assign them a swimlane and group each action they are responsible for under them.

Click here to start drawing your [activity diagram](https://creately.com/lp/activity-diagram-tool/).

### Activity Diagram Examples

Following are activity diagram templates that are instantly editable. Click on the image to open them in the editor so you can make the changes online.

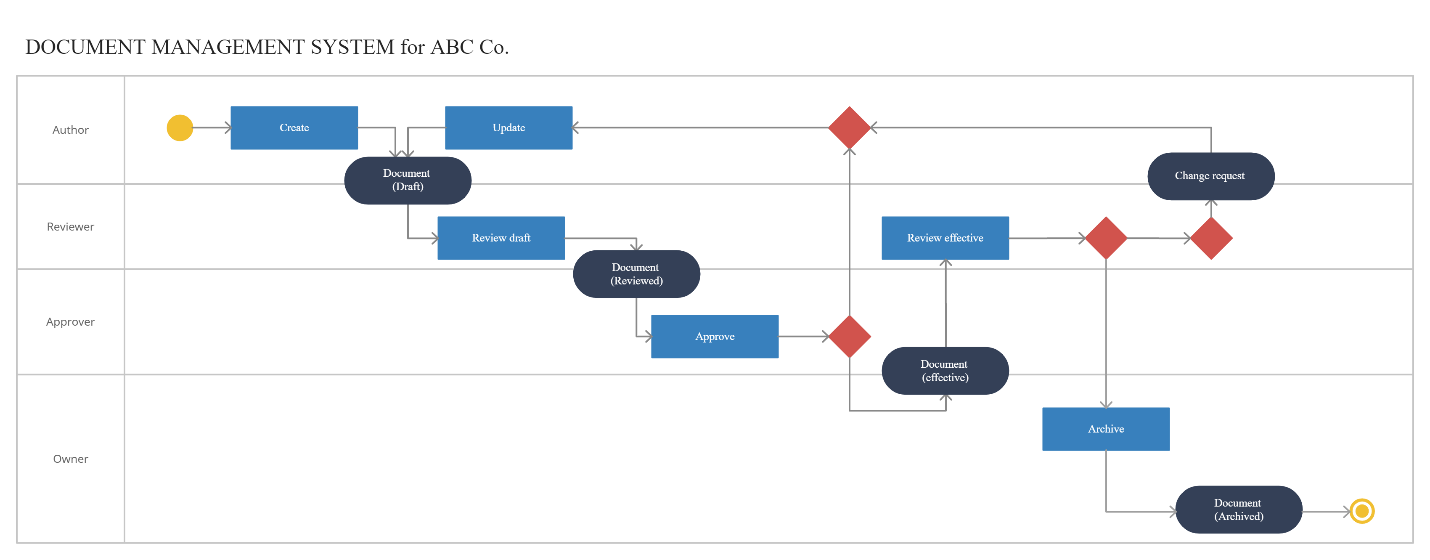
#### Activity Diagram for Login

[](https://creately.com/demo-start/?tempId=jlg33xw41)

*Click the image to edit this template*

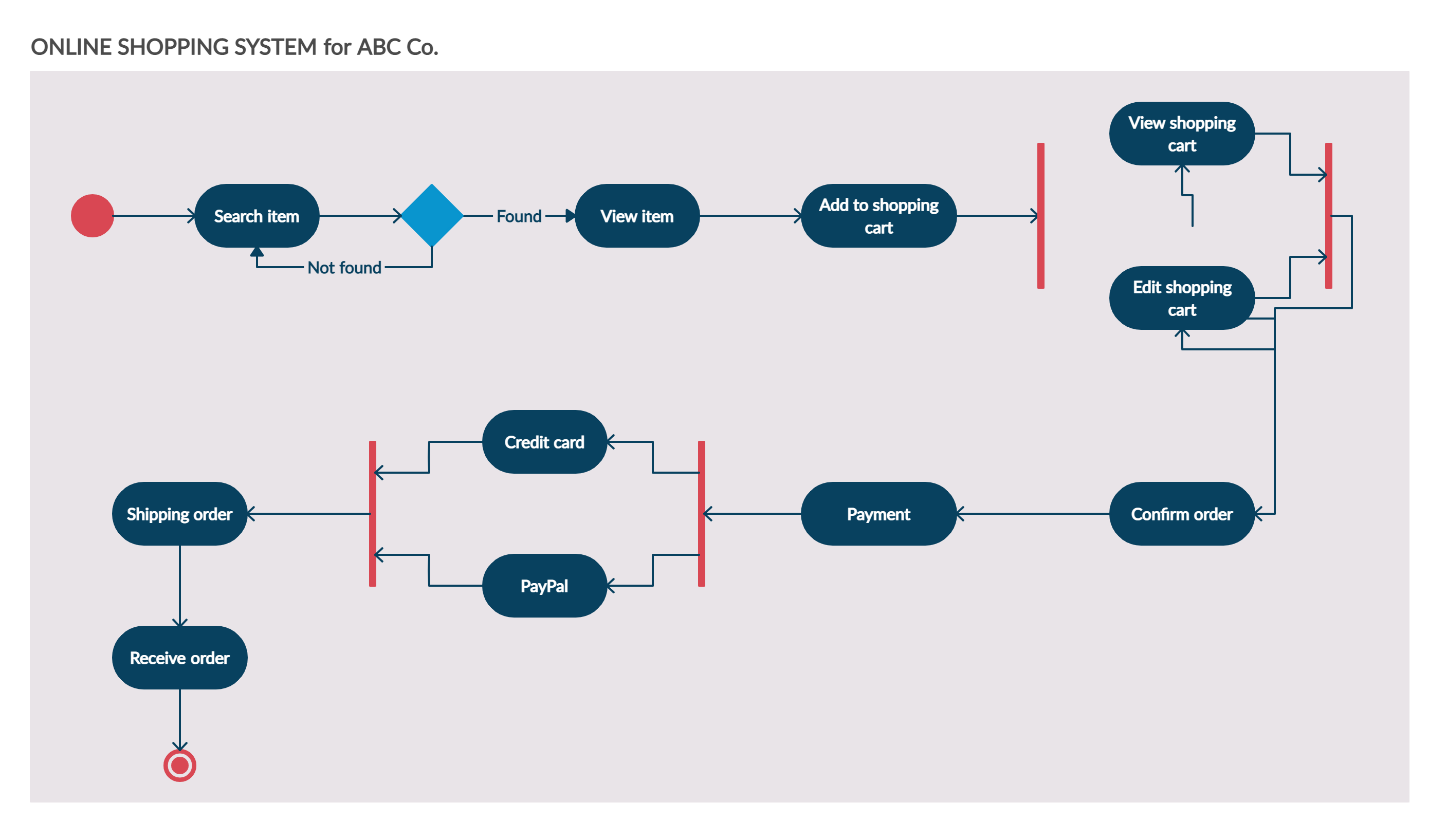
Here’s a [Login Module Use Case Diagram (UML)](https://creately.com/diagram/example/hmsymij21/Login%20module)

#### Activity Diagram for Document Management System

[](https://creately.com/demo-start/?tempId=m6gJj6c7QKA)

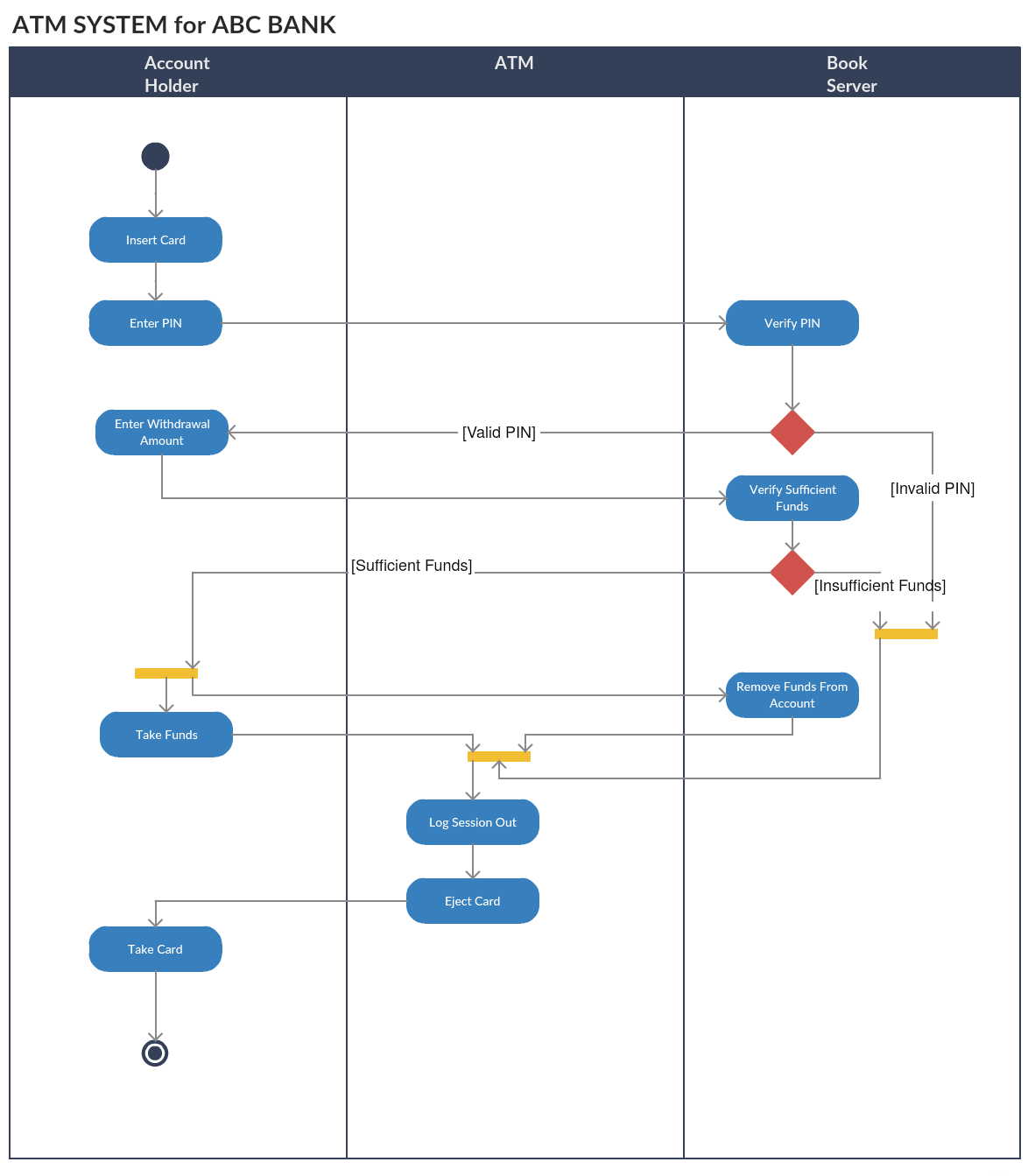
*Click the image to edit this template*

#### Activity Diagram for Online Shopping System

[](https://creately.com/demo-start/?tempId=i3b6yxgt1)

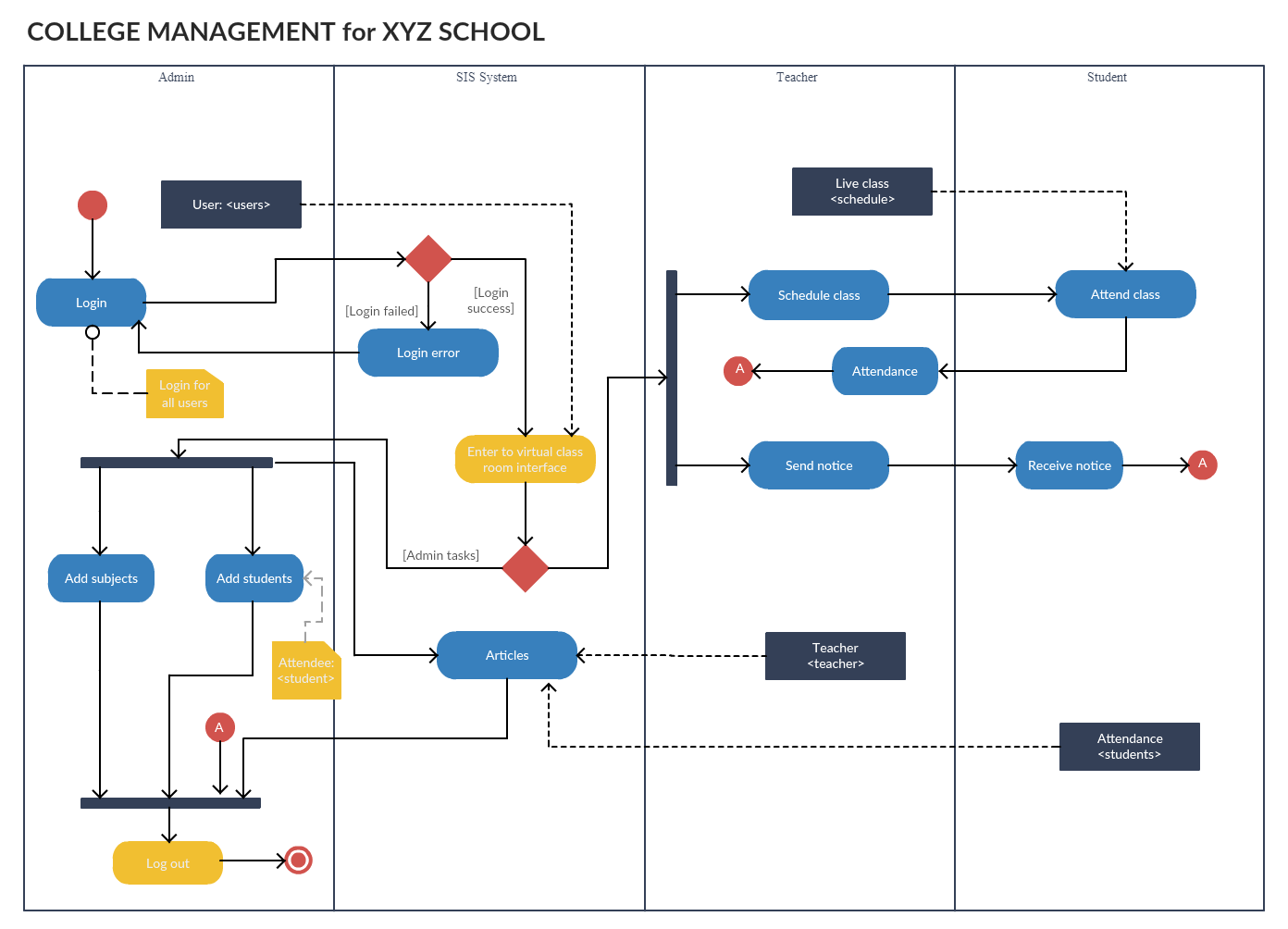
*Click the image to edit this template*

#### Activity Diagram for ATM

[](https://creately.com/demo-start/?tempId=i38i92071)

*Click on the template to edit online*

#### Activity Diagram for College Management System

[](https://creately.com/demo-start/?tempId=i39zw1o81)

Draw component diagrams assuming that you will build your system reusing existing components along with a few new ones

<https://www.lucidchart.com/pages/uml-component-diagram>

**What is a UML component diagram?**

The purpose of a component diagram is to show the relationship between different components in a system. For the purpose of UML 2.0, the term "component" refers to a module of classes that represent independent systems or subsystems with the ability to interface with the rest of the system.

There exists a whole development approach that revolves around components: component-based development (CBD). In this approach, component diagrams allow the planner to identify the different components so the whole system does what it's supposed to do.

More commonly, in an OO programming approach, the component diagram allows a senior developer to group classes together based on common purpose so that the developer and others can look at a software development project at a high level.

**Benefits of component diagrams**

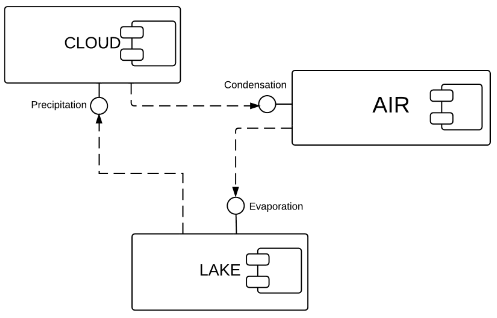
Though component diagrams may seem complex at first glance, they are invaluable when it comes to building your system. Component diagrams can help your team:

* Imagine the system’s physical structure.
* Pay attention to the system’s components and how they relate.
* Emphasize the service behavior as it relates to the interface.

**How to use component diagrams**

A component diagram in UML gives a bird’s-eye view of your software system. Understanding the exact service behavior that each piece of your software provides will make you a better developer. Component diagrams can describe software systems that are implemented in any programming language or style.

UML is a set of conventions for object-oriented diagrams that has a wide variety of applications. In component diagrams, the Unified Modeling Language dictates that components and packages are wired together with lines representing assembly connectors and delegation connectors. To learn more about UML and its uses, check out our guide, "[What Is UML](https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language)?"



Diagramming is quick and easy with Lucidchart. Start a free trial today to start creating and collaborating.

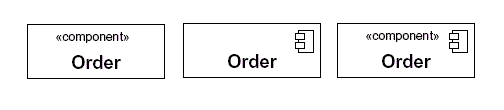
[Create a UML Diagram](https://lucid.app/pricing/lucidchart?anonId=0.a1fbc28b183e0a24a12&sessionDate=2022-10-16T12%3A05%3A10.717Z&sessionId=0.32f500d2183e0a24a15&type=discovery)

**Component diagram shapes and symbols**

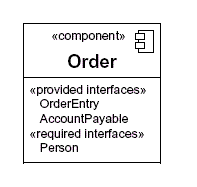
Component diagrams range from simple and high level to detailed and complex. Either way, you'll want to familiarize yourself with the appropriate UML symbols. The following are shape types that you will commonly encounter when reading and building component diagrams:

| Symbol | Name | Description |
| --- | --- | --- |
|  | Component symbol | An entity required to execute a stereotype function. A component provides and consumes behavior through interfaces, as well as through other components. Think of components as a [type of class](https://www.lucidchart.com/pages/uml-class-diagram). In UML 1.0, a component is modeled as a rectangular block with two smaller rectangles protruding from the side. In UML 2.0, a component is modeled as a rectangular block with a small image of the old component diagram shape. |
|  | Node symbol | Represents hardware or software objects, which are of a higher level than components. |
|  | Interface symbol | Shows input or materials that a component either receives or provides. Interfaces can be represented with textual notes or symbols, such as the lollipop, socket, and ball-and-socket shapes. |
|  | Port symbol | Specifies a separate interaction point between the component and the environment. Ports are symbolized with a small square. |
|  | Package symbol | Groups together multiple elements of the system and is represented by file folders in Lucidchart. Just as file folders group together multiple sheets, packages can be drawn around several components. |
|  | Note symbol | Allows developers to affix a meta-analysis to the component diagram. |
|  | Dependency symbol | Shows that one part of your system depends on another. Dependencies are represented by dashed lines linking one component (or element) to another. |

**How to use component shapes and symbols**



There are three popular ways to create a component's name compartment. You always need to include the component text inside the double angle brackets and/or the component logo. The distinction is important because a rectangle with just a name inside of it is reserved for classifiers (class elements).



As with the class notation, components also have an optional space to list interfaces, similar to the way you add attributes and methods to class notation. Interfaces represent the places where the groups of classes in the component communicate with other system components. An alternative way to represent interfaces is by extending symbols from the component box. Here is a quick rundown of the most commonly used symbols.

|  |  |
| --- | --- |
| Component Diagram Provided Interfaces | Provided interfaces: A straight line from the component box with an attached circle. These symbols represent the interfaces where a component produces information used by the required interface of another component. |
| Component Diagram Required Interfaces | Required interfaces: A straight line from the component box with an attached half circle (also represented as a dashed arrow with an open arrow). These symbols represent the interfaces where a component requires information in order to perform its proper function. |

In UML, a component diagram visually represents how the components of a software system relate to one another. To build one, try using Lucidchart’s custom component diagram shape library. Component diagrams should communicate:

* The scope of your system
* The overall structure of your software system
* Goals that the system helps human or non-human entities (known as actors) achieve

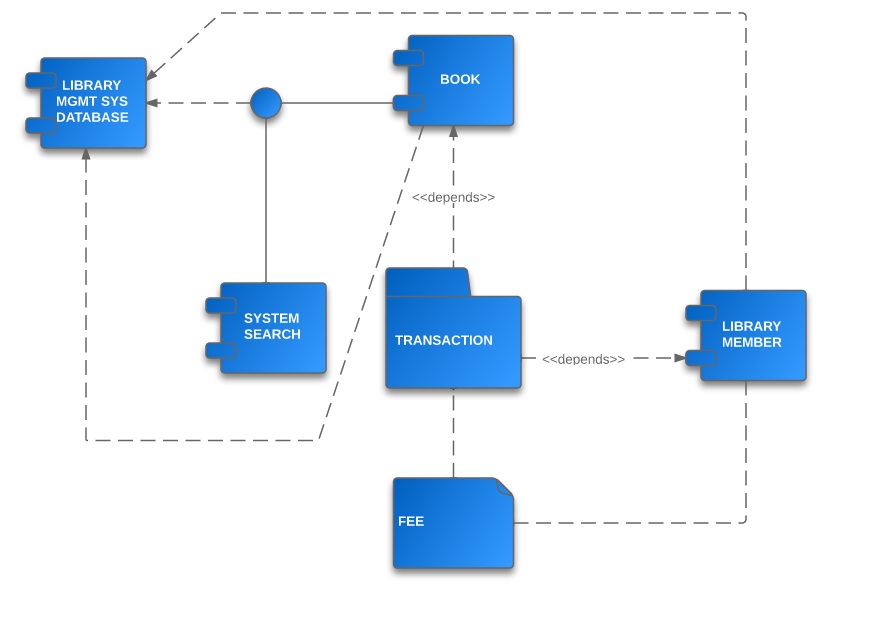
**Component diagram examples**

UML component diagrams bring simplicity to even the most complex processes. Take a look at the examples below to see how you can map the behaviors of specific processes with component diagrams in UML.

**Component diagram for a library management system**

Library systems were some of the first systems in the world to become widely run by computers. Today, many of these systems are managed in the cloud by third-party services, rather than internally. Though the term “library system” typically calls to mind a way to monitor printed books, library systems today organize all kinds of data checked in and checked out by users.

These transactions create a network of relationships between the components of the library system. To understand how these relationships work and how the system functions overall, examine the UML diagram below. You or your team can also use this diagram as a template.

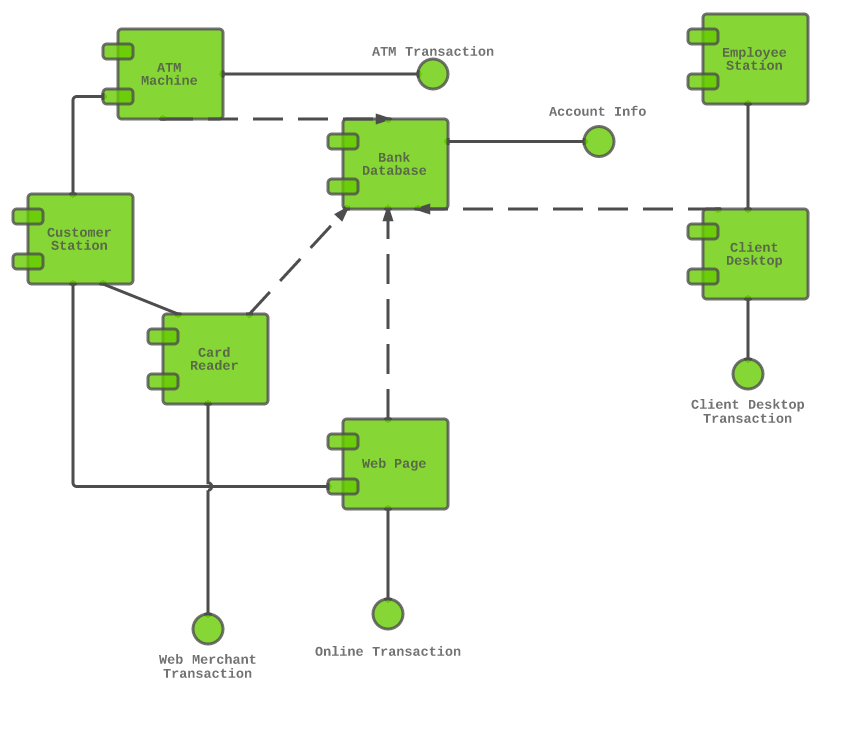


[*Click here to use this template*](https://www.lucidchart.com/documents/editNewOrRegister/230b4301-9d14-49fa-9b76-b8500970870d/0)

**Component diagram for an ATM system**

A component diagram is similar to a class diagram in that it illustrates how items in a given system relate to each other, but component diagrams show more complex and varied connections that most class diagrams can.

In the diagram below, each component is enclosed in a small box. The dotted lines with arrows show how some components are dependent on others. For example, the card reader, web page, client desktop, and ATM system are all dependent on the bank database. The dotted lines with circles at the end, known as “lollipop” symbols, indicate a realization relationship.



[*Click here to use this template*](https://www.lucidchart.com/documents/editNewOrRegister/e5a47717-bd14-4d1c-a378-bf0e69b0626e/0)

**How to make a component diagram**

In Lucidchart, you can easily craft an intricate component diagram in UML from scratch. Just follow the steps below:

1. Open a blank document or start with a template.
2. Enable the UML shape library. Click "Shapes" on the left side of the editor, check "UML" in the Shape Library Manager, and click "Save."
3. Select the shape you want from the library you added, and drag the shape from the toolbox to the canvas.
4. Model the process flow by drawing lines between shapes.

If you're looking for a step-by-step breakdown, use our guide on [how to draw a component diagram in UML](https://www.lucidchart.com/pages/how-to-draw-component-diagram-in-UML) to better assist you.

Draw deployment diagrams to model the runtime architecture of your system.

<https://creately.com/blog/diagrams/deployment-diagram-tutorial/>

Deployment diagrams are used to visualize the hardware processors/ nodes/ devices of a system, the links of communication between them and the placement of software files on that hardware.

In this [UML deployment diagram](https://creately.com/diagram-type/uml-diagram) tutorial, we will cover what is a deployment diagram, deployment diagram notations and how to draw one. You can use one of the editable [deployment diagram examples](https://creately.com/diagram-community/popular/t/deployment-diagram) to start right away.

If you are deploying to the cloud, you may skip [UML](https://creately.com/lp/uml-diagram-tool) altogether and use something like our [AWS architecture templates](https://creately.com/blog/diagrams/aws-templates-for-architecture-diagrams/) to achieve the same purpose.

## What is Deployment Diagram

A deployment diagram is a [UML diagram type](https://creately.com/lp/uml-diagram-tool) that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.

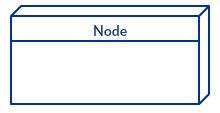
Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware.

Deployment diagrams help model the hardware topology of a system compared to other [UML diagram](https://creately.com/lp/uml-diagram-tool) types which mostly outline the logical components of a system.

### Deployment Diagram Notations

In order to [draw a deployment diagram](https://creately.com/diagram-community/popular/t/deployment-diagram), you need to first become familiar with the following deployment diagram notations and deployment diagram elements.

#### Nodes



A node, represented as a cube, is a physical entity that executes one or more components, subsystems or executables. A node could be a hardware or software element.

#### Artifacts

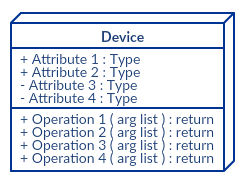


Artifacts are concrete elements that are caused by a development process. Examples of artifacts are libraries, archives, configuration files, executable files etc.

#### Communication AssociationCommunication path - deployment diagram notations

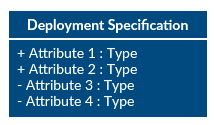
This is represented by a solid line between two nodes. It shows the path of communication between nodes.

#### Devices



A device is a node that is used to represent a physical computational resource in a system. An example of a device is an application server.

#### Deployment Specifications



Deployment specifications is a configuration file, such as a text file or an XML document. It describes how an artifact is deployed on a node.

## How to Draw a Deployment Diagram

Follow the simple steps below to [draw a deployment diagram](https://creately.com/diagram-community/popular/t/deployment-diagram). You can either use the deployment diagram examples below to get a head start or use our [UML diagram tool](https://creately.com/lp/uml-diagram-tool) to start from the beginning.

**Step 1:** Identify the purpose of your deployment diagram. And to do so, you need to identify the nodes and devices within the system you’ll be visualizing with the diagram.

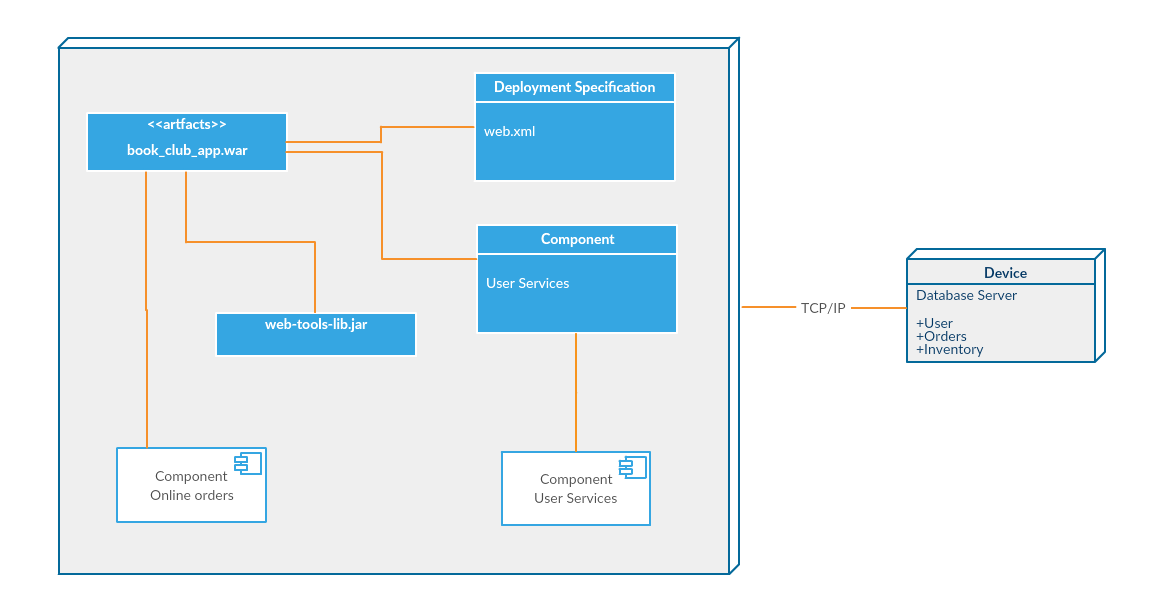
**Step 2:** Figure out the [relationships](https://creately.com/lp/interrelationship-diagram) between the nodes and devices. Once you know how they are connected, proceed to add the communication associations to the diagram.

**Step 3:** Identify what other elements like components, active objects you need to add to complete the diagram.

**Step 4:** Add dependencies between components and objects as required.

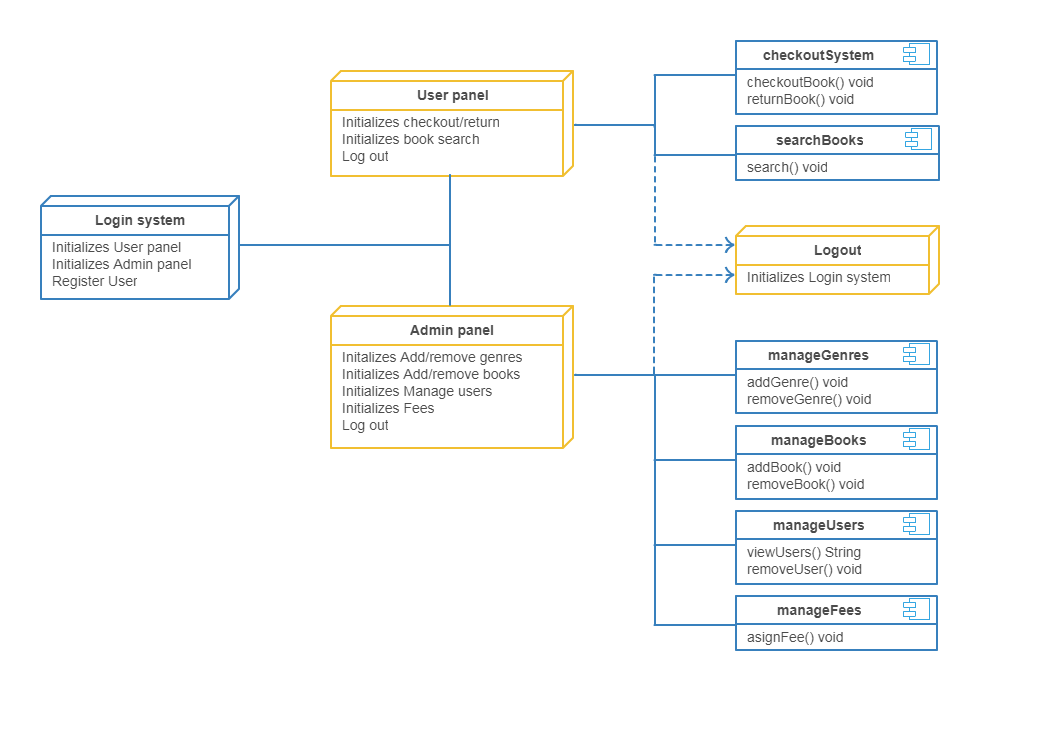
## Deployment Diagram Examples

#### Deployment Diagram for Online Shopping System

[](https://creately.com/demo-start/?tempId=i3qwfizr1)

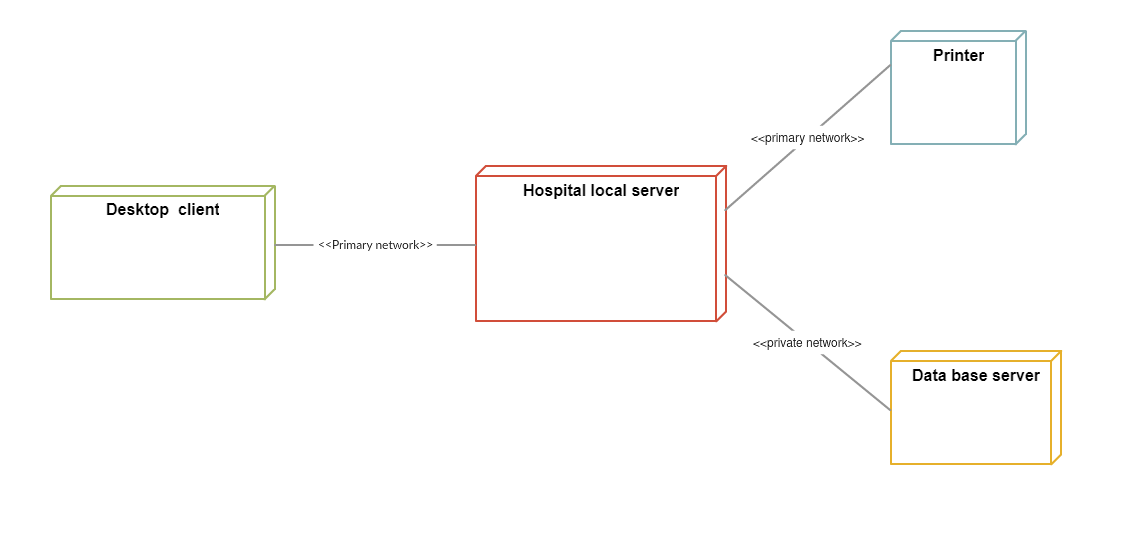
*Click on the template to edit online*

#### Deployment Diagram for Library Management System

[](https://creately.com/demo-start/?tempId=jmk6au3v1)

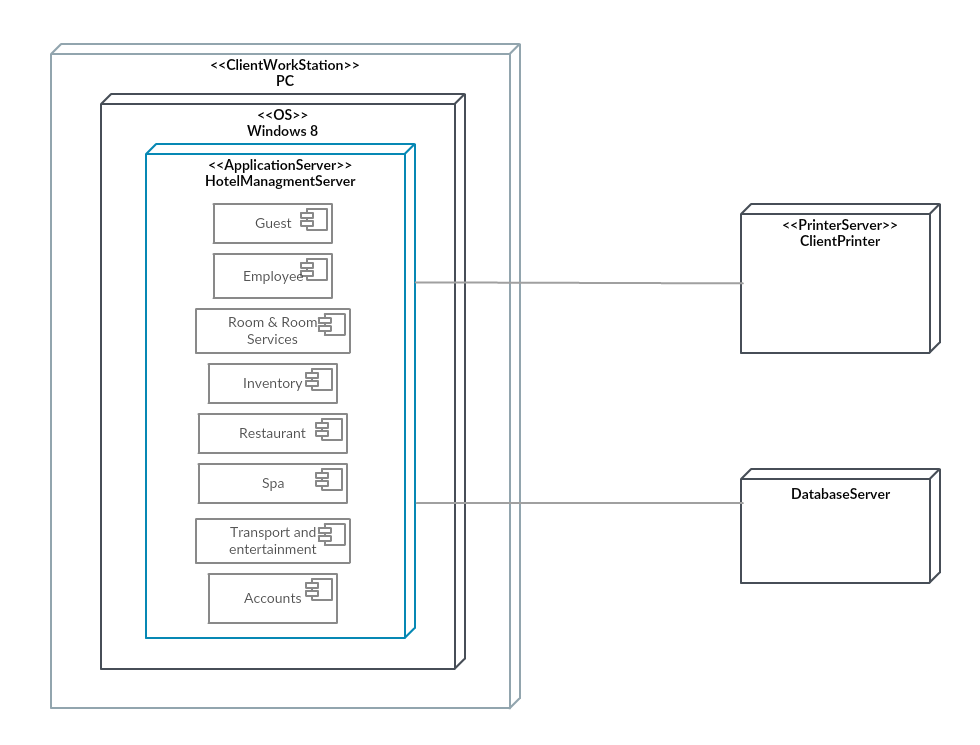
*Click on the template to edit online*

#### Deployment Diagram for Hospital Management System

[](https://creately.com/demo-start/?tempId=jmk7cp671)

*Click on the template to edit online*

#### Deployment Diagram for Hotel Management System

[](https://creately.com/demo-start/?tempId=jmk848iy1)

Mini Project

Draw following UML Diagrams for Bank Management application

a. Class Diagram

b. Object Diagram

c. ER Diagram

d. Component Diagram

<https://www.freeprojectz.com/uml-diagram/banking-management-system-uml-diagram>