Use Case Diagram

Imagine you’ve been asked to design a new software system—say, an online food delivery app. Before you dive into writing classes, functions, or database schemas, the first question is:

"What exactly should this system do?"

You need to understand the problem from the user’s point of view.

That’s where **Use Case Diagrams** come in.

In this article, we will explore:

- What a Use Case Diagram is?
- Building Blocks of a Use Case Diagram
- How to Draw a Use Case Diagram (Step-by-Step)

1. What is a Use Case Diagram?

A **Use Case Diagram** is a visual representation of how different users (also called **actors**) interact with a system.

It’s like a **bird’s-eye view** of the system’s functionality without getting into any code.

At its core, a use case diagram answers one key question:

"What can users do with this system?"
Use Case Diagrams **do not** explain how a feature is implemented. They only show:

- Who is using the system
- What they want to do
- How the system responds to those actions

Use Case Diagrams are typically used in the **early phases** of software development—right after the requirements are gathered and before detailed design begins.

Let’s say you’re building a ride-booking system.

Your Use Case Diagram might include:

Actors: Rider, Driver

Use Cases:

- **Rider:** Book Ride, Cancel Ride, Make Payment
- **Driver:** Accept Ride, Mark Ride Complete

2. Building Blocks of a Use Case Diagram

A use case diagram might look simple on the surface, just stick figures, ovals, and arrows, but each element has a specific purpose and meaning.

To read, interpret, or create a use case diagram, you need to understand its four key building blocks:

- **Actors**
- **Use Cases**
- **System Boundary**
- **Relationships**

Let’s explore each of them in detail.

1. Actors

An **actor** represents **anything that interacts with the system from outside**. Most often, actors are people (users), but they can also be external systems, sensors, or services.

Notation: Stick figure labeled with the actor’s role.

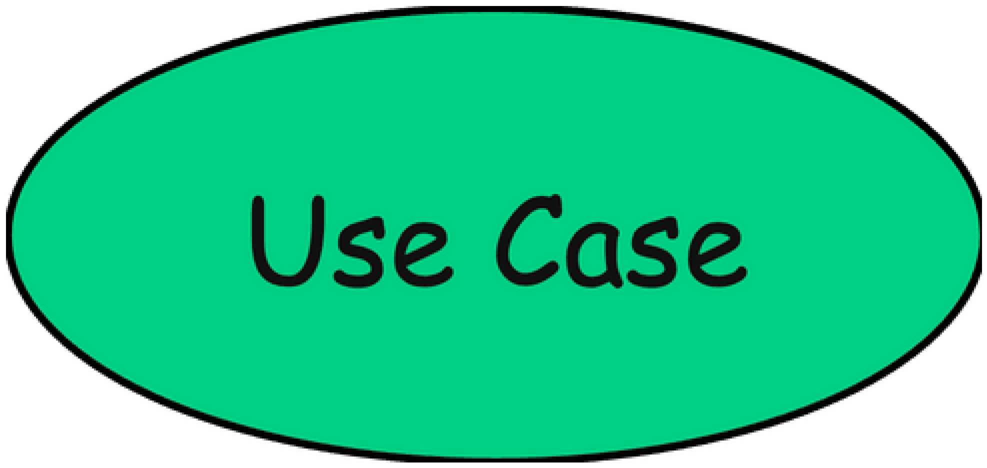
There are two types of actors:

- **Primary actors**: Initiates an interaction (e.g., a user logging in)
- **Secondary actors**: Helps fulfill a use case but don't initiate it (e.g., a payment gateway)

2. Use Cases

A **use case** is a **functionality or goal** that the system provides to the actor. Think of it as an action the user wants to perform.

Notation: Oval with the name of the use case inside.



Each use case should:

- Start with a verb (e.g., "Register", "Search", "Book Ticket", “Make Payment“)
- Represent a complete interaction from the user's point of view
- Deliver a meaningful result

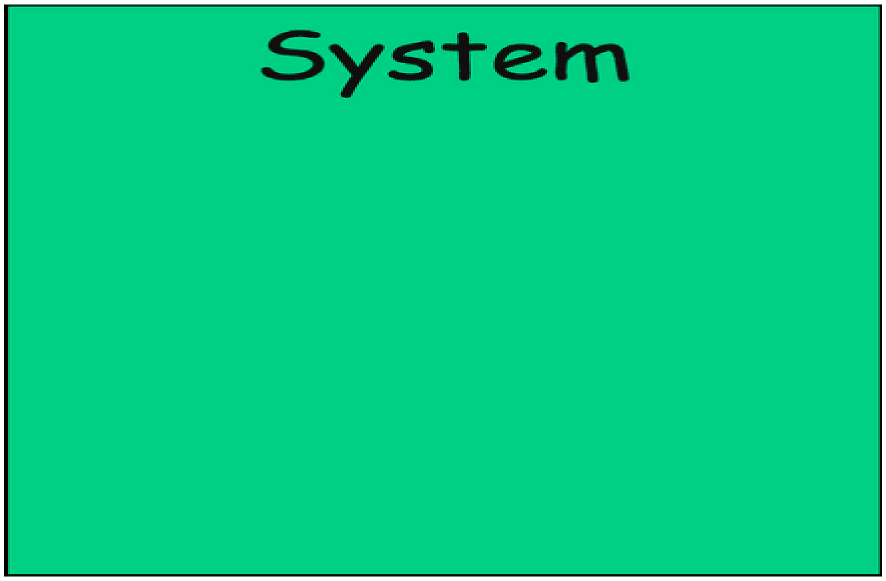
3. System Boundary

The **system boundary** defines **what’s inside the system and what’s outside**. This helps clearly define scope.

Examples: "Library Management System", "Online Banking System"

Anything inside the box is part of your system. Anything outside is not your responsibility.

Notation: A labeled box that encloses al the use cases



4. Relationships

Relationships describe how actors and use cases are connected or how different use cases relate to one another. There are four main types:

a. Association

- Connects an actor to a use case
- Example: `Customer → Place Order`

Notation: Represented by a solid line

b. Include

- Represents **common functionality** shared between use cases
- Example: `Checkout` includes `Validate Payment`
- Think of it as: "Always includes this"

Notation: Dashed arrow with label `<<include>>`

c. Extend

- Represents **optional or conditional behavior**
- Example: `Search` can extend to `Advanced Filter`
- Think of it as: "Sometimes adds this"

Notation: Dashed arrow with label `<<extend>>`

d. Generalization

- Shows **inheritance (**parent-child relationship**)** between actors or use cases

- The child actor/use case is an enhancement of the parent use case.
- Example: `Admin` is a specialized `User`

Notation: Directed arrow with a triangle arrowhead from child to parent

3. How to Draw a Use Case Diagram (Step-by-Step)

Let’s walk through the process of drawing a use case diagram step by step.

To make it more relatable, we’ll build a use case diagram for a **Movie Ticket Booking System** as we go along.

Step 1: Identify Actors

Start by identifying **who** will interact with your system. These could be:

- Human users (e.g., Customer, Admin)
- External systems (e.g., Payment Gateway, Notification Service)

For our example:

- **Customer**: A person who books and cancels tickets, browses movies.
- **Admin:** An individual manages movie listings and show schedules.
- **Payment Gateway**: An external system that processes payments.

Step 2: Identify Use Cases

Now, list out **what the actors want to do**. These are your use cases, the actions or goals the system should support.

Tips:

- Use clear, action-oriented names like “Search”, “Submit Form”, “Track Order”
- Think from the **actor's perspective**, not the system’s

For the movie booking system:

- **Browse Movies:** Customers can browse the available movies.
- **Book Ticket:** Customers can book tickets.

- **Cancel Booking:** Customers can cancel tickets.
- **Make Payment:** Customers can make payments for their tickets.
- **Add/Edit Movie Listings:** Admins manage the movie listings.

Step 3: Define the System Boundary

Draw a box around all the use cases. This box represents your system and visually defines the scope of your system.

Anything inside the box is your responsibility to implement. Anything outside is external.

Draw a rectangle and label it with the name of your system.

For our example: **Movie Ticket Booking System**

Step 4: Connect Actors to Use Cases

Now, link each actor to the relevant use cases using **solid lines** (associations).

- The **Customer** is connected to most of the features
- The **Admin** is only connected to the movie management use case
- The **Payment Gateway** interacts only with the payment flow

Step 5: Model Relationships Between Use Cases

Use arrows and labels to show the relationships among use cases.

a. Include

This means a use case **always includes** another use case.

It helps you:

- Reuse common functionality across multiple use cases
- Keep your diagram DRY (Don't Repeat Yourself)

Example: Whenever someone books a ticket, they must make a payment.

b. Extend

Used when a use case has **optional** or **conditional** behavior.

Example: While browsing movies, the user might choose to **filter by genre**—but it's not mandatory.

c. Generalization
Use when actors or use cases **share common behavior** but differ slightly.

Example: A `Registered User` and `Guest User` both act like a `Customer`, but with slight differences. Use generalization to reflect that.

Bringing It All Together