Sequence Diagram

Imagine you're ordering a pizza.

You call the pizza place. The cashier takes your order, sends it to the kitchen, and the chef starts preparing it. Once it’s ready, a delivery person picks it up and brings it to your door.

Now, picture all those interactions—**you talking to the cashier**, **the cashier passing the order to the chef**, **the chef preparing the food**, and finally **the delivery**. Each step follows a clear order, and each participant plays a specific role.

That’s exactly what **Sequence Diagrams** do in software design.

They **map out interactions between objects over time**, just like a storyboard for how things happen in a system.

In this article, we will explore:

- What a Sequence Diagram is?
- Building Blocks of a Sequence Diagram
- Types of Messages in Sequence Diagrams

1. What is a Sequence Diagram?

A **sequence diagram** is a type of UML (Unified Modeling Language) diagram that shows **how objects in a system interact with each other**, step by step.

It focuses on the **order of messages exchanged** between different components or actors to achieve a particular task or use case.

These diagrams model:

- **Who** the participants (objects or actors) are
- **What** messages are exchanged
- **In what order** the messages occur
- **How long** each participant is active

In other words, sequence diagrams help answer: “**Who is doing what, and when?”**

2. Building Blocks of a Sequence Diagram

Actors

Actors are **external entities** (usually users or external systems) that **interact with your system**. They initiate communication and trigger system behavior.

Example: A user trying to log into a website, or a payment gateway calling your API.

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

They’re shown on the **far left** and are usually the first to send a message.

Objects / Participants

Objects (also called participants or lifeline owners) are **instances of classes or components** in your system. They represent the internal entities that send or receive messages.

Example: `LoginController`, `AuthService`

Notation: Each object will have a **vertical dashed line (lifeline)** extending downward to represent its activity over time.

Lifelines

A **lifeline** is a **dashed vertical line** drawn below each participant. It shows that the object exists during the interaction and represents the **flow of time**—top is the beginning, bottom is later.

Use case: Helps track the timing and duration of each object's activity during the scenario.

Activation Bars

Activation bars (rectangles over lifelines) show when an object is **actively processing a message** or performing some task. It visually indicates **when an object is "alive" or performing logic.**

Notation: A thin rectangle on a lifeline

3. Types of Messages in Sequence Diagrams

In a sequence diagram, messages are the **arrows** that bring the diagram to life. They show **who talks to whom**, **what they say (method or message)**, and **when** they say it.

Understanding message types is crucial because they reflect **how different parts of your system communicate**—whether they wait for a response, fire-and-forget, or return data.

Let's walk through the most commonly used message types in sequence diagrams, with real-world analogies and visual syntax.

1. Synchronous Message (→)

A synchronous message is like a **phone call**. You ask a question and **wait** for the other person to answer before you move on.

- **Arrow Style:** Solid line with filled arrowhead →
- **Sender waits** for the receiver to finish
- **Used for:** method calls, API requests where a response is needed

Example: User calls `login()` on `LoginController`

2. Asynchronous Message (→>)

An asynchronous message is like **sending a text—**you don't wait for a response. You just fire off the message and continue your own work.

- **Arrow Style:** Solid line with open arrowhead →>
- **Sender doesn't wait** for a response
- **Used for:** background tasks, event notifications, message queues

Example: Send a welcome email after successful registration.

3. Return Message (←--)

Return messages indicate that the receiver is **sending a response back** to the sender. Think of it like the **reply you get after asking a question**.

- **Arrow Style: Dashed** line with open arrowhead ←--
- Usually follows a synchronous message

Example:

4. Self-Message (Recursive Call)

Sometimes, an object needs to talk to itself. This is shown using a **looped arrow** that points back to the same lifeline.

- **Used for:** recursive functions, internal helper method calls

Example:

5. Create Message

When a message **creates a new object**, it's called a **create message**.

- Typically ends with the **new object's lifeline starting**
- Arrow points to the object's head

Example: Creating a new `Session` object.

6. Destroy Message (Optional)

Sometimes you want to indicate that an object is destroyed after a certain point—like closing a file or deleting a session.

- Marked with an **'X'** at the end of the lifeline
- Used rarely, but useful for **resource cleanup** and **lifecycle clarity**