# 17. Multiple stages of processing are needed

Imagine that you need to implement a logic that requires multiple stages of processing, possibly involving conditional logic.

One example of this would be a validation of an HTTP request. First, you may want to see whether the incoming request matches any allowed paths in your application. Then, you may want to check whether the user is authenticated. After that, you may want to check whether the user is allowed to access the particular resource that the request was made for. And so on.

In this situation, you may want to short-circuit this flow as soon as you encounter a condition that wouldn't allow the request to proceed any further. For example, if the request was made to a path that doesn't exist, there is no point in checking the user's credentials. Returning the `404` response code would be sufficient.

## Suitable design patterns

### Chain of Responsibility

This design pattern was specifically developed to deal with this problem.

Basically, your logic is arranged into a chain of individual components. Each component has a condition that decides whether to move on to the next component or short-circuit the logic then and there.

A good example of this would be an ASP.NET Core middleware pipeline which was designed specifically for the request validation that was described in the example above.

### Why would you use a Chain of Responsibility

- You can control the order of multi-stage processing.
- You can stop the process at any time if moving on doesn't make sense any longer.
- Each component in the chain is responsible for a specific stage of processing, so the single responsibility principle is easy to enforce.
- Suitable for scenarios where a one-off logical flow needs to be executed.

## Builder

Builder design pattern was already mentioned in **chapter 6**. Although it's not designed for the specific request validation example outlined in the problem description, Builder is suitable for other types of situations where you need to execute multiple processing stages.

The whole purpose behind Builder is to allow you to build some sort of object step-by-step. So, if you put a Builder implementation via a multi-stage conditional logic, you would end up with a different end product for each unique combination of conditions.

Unlike Chain or Responsibility, however, Builder doesn't have to work with specific steps done in a specific order. For example, if we are to use `StringBuilder` from C# to build a string, we can apply any arbitrary logic to do it. We can perform as many manipulations with it as we want.

### Why would you use Builder instead of Chain of Responsibility

- Easier to implement when the order of the processing steps doesn’t need to be exact.
- Suitable for scenarios where a reusable object needs to be built.