Event Driven Architecture - Part 1

In this lesson, which is the part one of the event driven architecture, we will understand concepts like Blocking & Non-blocking.

We'll cover the following

When writing modern Web 2.0 applications chances are you have across terms like *Reactive programming*, *Event-driven architecture*, concepts like *Blocking* & *Non-blocking*.

What are they? Should I be aware of them?

You might have also noticed that tech like *NodeJS*, *Play*, *Tornado*, *Akka.io* are gaining more popularity in the developer circles for modern application development in comparison to the traditional tech.

What is the reason for that? Is it just that we are bored of working on the traditional tech like *Java*, *PHP* etc. & are attracted towards the shiny new stuff or are there any technical reasons behind this?

In this lesson, we will go through each and every concept step by step & realize the demands of modern software application development.

So, without any further ado, let's get on with it.

Alright, at this point in the course, we know what *persistent connections* are? What *asynchronous behaviour* is & why do we need it? We can't really write real-time apps without implementing them.

Starting with *Blocking*. What is it?

What Is Blocking?

In web applications *blocking* means the flow of execution is blocked waiting for a process to complete. Until the process completes it cannot move on. Let's say we

have a block of code of 10 lines within a function and every line triggers another external function executing a specific task.

Naturally, when the flow of execution enters the main function it will start executing the code from the top, from the first line. It will run the first line of code and will call the external function.

At this point in time until the external function returns the response. The flow is blocked. The flow won't move further, it just waits for the response. Unless we add *asynchronous behaviour* to it by annotating it and moving the task to a separate thread. But that's not what happens in the regular scenario, like in the regular *CRUD-based* apps. Right?

So, this behaviour is known as *Blocking*. The flow of execution is blocked.

What Is Non-Blocking?

Now coming to the *Non-blocking* approach. In this approach the flow doesn't wait for the first function, that is called, to return the response. It just moves on to execute the next lines of code. This approach is a little not so consistent as opposed to the blocking approach since a function might not return anything or throw an error, still the code in the sequence up next is executed.

The *non-blocking* approach facilitates *IO Input-Output* intensive operations. Besides the disk & other the hardware-based operations, communication and network operations also come under the *IO* operations.

We will continue this discussion in part two of event-driven architecture lesson. We will have an insight into what are events? what is event-driven architecture? And the technologies used to implement it.