Lesson 7: Event Emitter Module

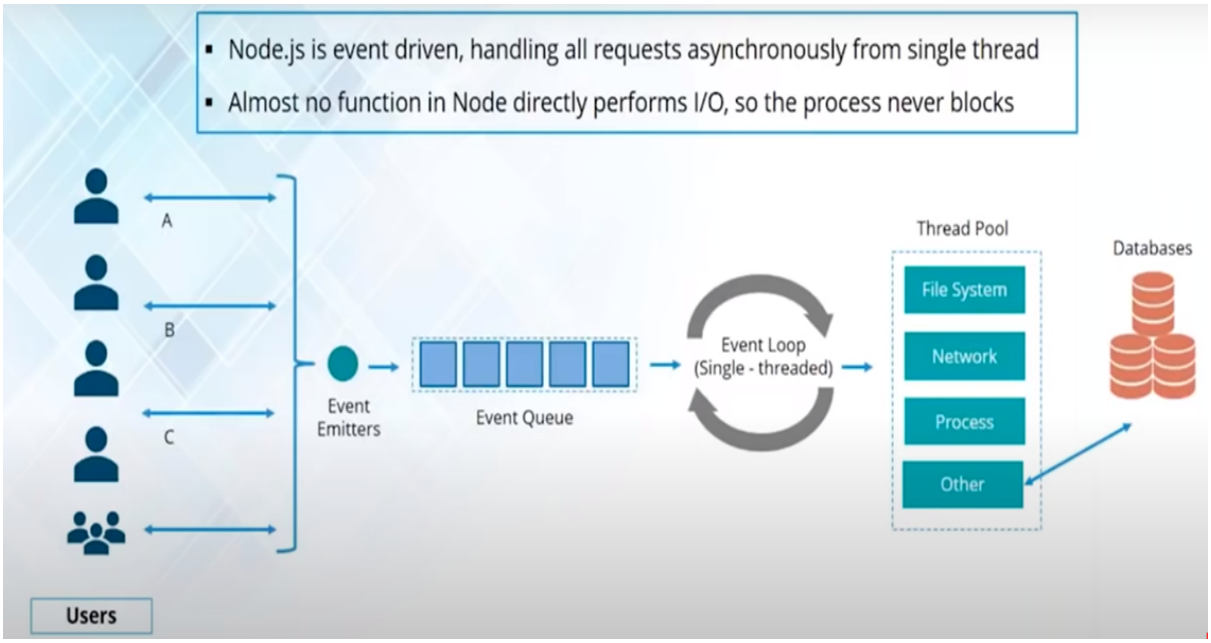
Event Loop

Node.js is a single-threaded application, but it can support concurrency via the concept of **event** and **callbacks**. Every API of Node.js is asynchronous and being single-threaded, they use **async function calls** to maintain concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever a task gets completed, it fires the corresponding event which signals the event-listener function to execute.

Event-Driven Programming

Node.js uses events heavily and it is also one of the reasons why Node.js is pretty fast compared to other similar technologies. As soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for the event to occur.

In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.



Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. The functions that listen to events act as **Observers**. Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through events module and EventEmitter class which are used to bind events and event-listeners as follows −

```
// Import events module
var events = require('events');

// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
Following is the syntax to bind an event handler with an event −
// Bind event and event  handler as follows
eventEmitter.on('eventName', eventHandler);
We can fire an event programmatically as follows −
// Fire an event
eventEmitter.emit('eventName');
```

Event Emitter Class:

Many objects in a Node emit events, for example, a net.Server emits an event each time a peer connects to it, an fs.readStream emits an event when the file is opened. All objects which emit events are the instances of events.EventEmitter.

EventEmitter class lies in the events module. It is accessible via the following code −

```
// Import events module
var events = require('events');

// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

When an EventEmitter instance faces any error, it emits an 'error' event. When a new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener' event is fired.

EventEmitter provides multiple properties like **on** and **emit**. **on** property is used to bind a function with the event and **emit** is used to fire an event.

Methods

Sr.No.	Method & Description
1	**addListener(event, listener)** Adds a listener at the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. Multiple calls passing the same combination of event and listener will result in the listener being added multiple times. Returns emitter, so calls can be chained.
2	**on(event, listener)** Adds a listener at the end of the listeners array for the specified event. No checks are made to see if the listener has already been added. Multiple calls passing the same combination of event and listener will result in the listener being added multiple times. Returns emitter, so calls can be chained.
3	**once(event, listener)** Adds a one time listener to the event. This listener is invoked only the next time the event is fired, after which it is removed. Returns emitter, so calls can be chained.
4	**removeListener(event, listener)** Removes a listener from the listener array for the specified event. **Caution** − It changes the array indices in the listener array behind the listener. removeListener will remove, at most, one instance of a listener from the listener array. If any single listener has been added multiple times to the listener array for the specified event, then removeListener must be called multiple times to remove each instance. Returns emitter, so calls can be chained.
5	**removeAllListeners([event])**

	Removes all listeners, or those of the specified event. It's not a good idea to remove listeners that were added elsewhere in the code, especially when it's on an emitter that you didn't create (e.g. sockets or file streams). Returns emitter, so calls can be chained.
6	**setMaxListeners(n)** By default, EventEmitters will print a warning if more than 10 listeners are added for a particular event. This is a useful default which helps finding memory leaks. Obviously not all Emitters should be limited to 10. This function allows that to be increased. Set to zero for unlimited.
7	**listeners(event)** Returns an array of listeners for the specified event.
8	**emit(event, [arg1], [arg2], [...])** Execute each of the listeners in order with the supplied arguments. Returns true if the event had listeners, false otherwise.

Class Methods

Sr.No.	Method & Description
1	**listenerCount(emitter, event)** Returns the number of listeners for a given event.

Events

Sr.No.	Events & Description
1	**newListener** • **event** − String: the event name • **listener** − Function: the event handler function This event is emitted any time a listener is added. When this event is triggered, the listener may not yet have been added to the array of listeners for the event.
2	**removeListener** • **event** − String The event name • **listener** − Function The event handler function This event is emitted any time someone removes a listener. When this event is triggered, the listener may not yet have been removed from the array of listeners for the event.

Common Patterns for EventEmitters:

There are two common patterns that can be used to raise and bind an event using EventEmitter class in Node.js.

1. Return EventEmitter from a function
2. Extend the EventEmitter class

Return EventEmitter from a function

In this pattern, a constructor function returns an EventEmitter object, which was used to emit events inside a function. This EventEmitter object can be used to subscribe for the events.

Extend EventEmitter Class

In this pattern, we can extend the constructor function from EventEmitter class to emit the events so that you can use EventEmitter class to raise and handle custom events in Node.js.