

Shri Vile Parle Kelavani Mandal's DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai) NAAC Accredited with "A" Grade (CGPA: 3.18)



Department of Artificial Intelligence and Machine Learning B.Tech. Sem: V Subject: Full Stack Development Laboratory (DJS22AML504)

Experiment 6

Name: Shu	bham Mourya SAP ID: 60017230110
THE STATE OF THE S	Nome - Shultony Hourya Roll No + A065 60017230110
	Enfuriment No:-6
SSECTION AND ADDRESS OF THE ADDRESS	Am: Implementing callbacks, event loops in Node Js.
	Though Is, callbacks are essential to achieving a ynchronous programming. A callback function is invoked when specific task is completed. Since Node JS operates mon-blockingly, often uses callback to
	To synchronous env, fragram will execute line-by-line of when encounters I/Oy it will halt until that task is completed. When :1/o operation is brushed, Node Is call back function that
•	randles result of that operation. The var is = require ("if") for readifie ("mout txt", function (em data)? If (exr) return console error (err); Console log (data to String ());
	console · log ("fræg rum Ended");
	Event Loops in Node JS: The event loop 9s another core betwee in Node JS trad allows non-blocking layrah sonous operation.
<u>Eundaran</u>	FOR EDUCATIONAL USE



Shri Vile Parle Kelavani Mandal's DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai) NAAC Accredited with "A" Grade (CGPA: 3.18)



J-	The state of the s
1.3	
4	
	cir Event Queue: When asychronous operation, like file reading or
	nothers k requests, are started, they are placed in event quare.
	queue for tasks that need to be executed.
	queue for tasks that need to be executed.
	Citi Non- thokana - While wall - a fex on No and I and t
Constant Cons	citi Non-tocking: While waiting for an 1/0 operation to complete, event loop cellians other code to run without blocking the
	application.
	our Collhack is When 1/2 and leaveling the war of
	callback: When 1/0 sperchan forwher, the associated callback is executed by the event loop.
	460
5 	fs. readfile ('file.txt') (er, dela) => f
The second secon	of (err) hrow err;
	console log (data);
	4).
	Console log ('Reading File');
	le al Field and le la selection le selection
	Is sad File 1) method instruction asynchronous file read. Once file seadors is completed, callback hunchon is called,
	and file content is printed.
	of aynchronau programming in Node: IS wing callbacks
	of asynchronau programming in Node: Is using contracte
	devent loop.
Sundaram	FOR EDUCATIONAL USE





Date:	
Aim	Implementing Callbacks, Event loops in Node.js
Software	
Pre-	Active internet connection
requisite	
Theory	What is Callback? Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks. For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter. So, there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results Example of Non-Blocking Code: 1. Create a text file named input.txt with the following content. Tutorials Point is giving self-learning content to teach the world in simple and easy way!!!!! 2. Update main.js to have the following code – var fs = require("fs"); fs.readFile('input.txt', function (err, data) { if (err) return console. Error(err); console.log(data.toString()); });
	console.log("Program Ended"); 3. Now run the main.js to see the result – \$ node main.js 4. Verify the Output. Program Ended Tutorials Point is giving self-learning content to teach the world in simple and easy way!!!! The program does not wait for file reading and proceeds to print "Program Ended" and at the same time, the program without blocking continues reading the file.
	Event Loops The Event Loop and Emitters are fundamental concepts in Node.js, which make it an efficient and event driven environment for server-side programming. Let's explore these concepts in detail: Event Loop: The event loop is a critical component of Node.js that enables non-blocking, asynchronous I/O operations. It's responsible for managing the execution of code in response to events, allowing Node.js to efficiently handle multiple concurrent connections without blocking the main thread. Here is how the event loop works:

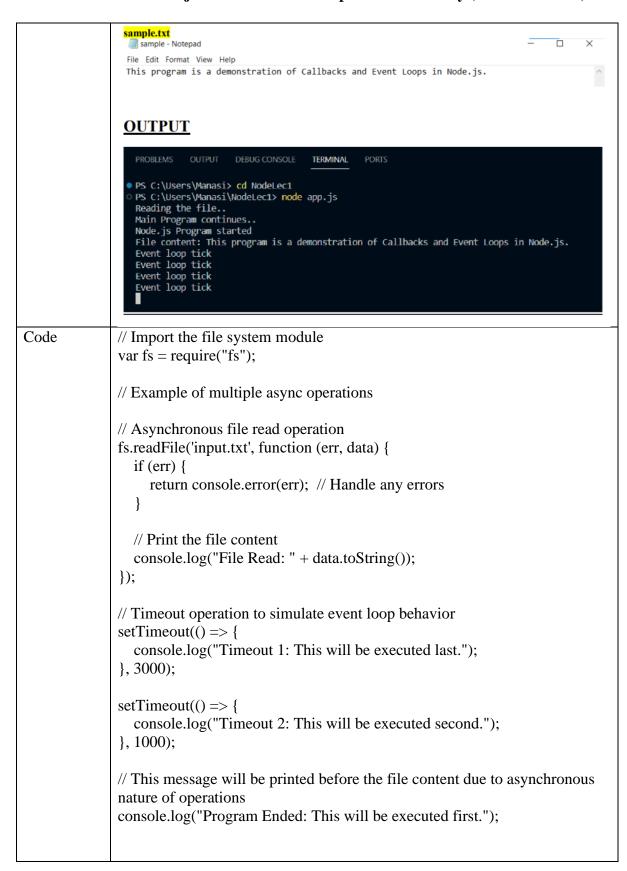
Event Queue: When asynchronous operations (such as reading files, making network requests, or handling user input) are initiated, they are placed in an event queue. □ Event Loop: The event loop continually checks the event queue to see if there are any events (such as callbacks or promises) that need to be executed. Non-Blocking: While waiting for I/O operations to complete, the event loop allows other code to run, ensuring that the application remains responsive and doesn't block. □ Callbacks: Callback functions are often used in Node.js to handle events when operations are completed. When an event is ready to be processed, its associated callback function is executed. Example of an event loop in Node.js: const fs = require('fs'); // Asynchronous file read operation fs.readFile('file.txt', 'utf8', (err, data) => { if (err) throw err; console.log(data); **})**; console.log('Reading file...'); In this example, the file read operation is asynchronous, and the callback function is executed once the operation is complete. While waiting for the I/O operation to finish, other code (e.g., the console.log) can run without blocking. **Program**

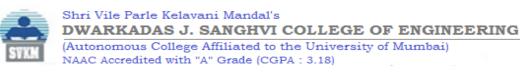
```
app.js
const fs = require('fs');

function readFileCallback(err, data){
    if(err){
        console.error('Error reading the file:', err);
    }
    else{
        console.log('File content:', data);
    }
}

fs.readFile('sample.txt', 'utf8', readFileCallback);
console.log('Reading the file...');
console.log('Main Program continues...');
console.log('Node.js Program started')
```









Result	PS E:\Shubham\DJS\SEM 5\fullstack\pr6_node> node main.js Program Ended: This will be executed first. File Read: Rohit Sharma (born 30 April 1987) is an Indian international cricketer who currently plays for and captains the India national cricket team in Test and One Day International (ODI) matches. Previously, he also captained the team in Twenty20 International (T20I) matches and led India's win in 2024 ICC Men's T20 World Cup, subsequent to which he retired from T20s in June 2024.[3][4] He is considered to be one of the best batsmen of his generation and one of the greatest opening batters of all time, [5] He is also known as Hitman for his timing, elegance, six-hitting abilities and leadership skills. Timeout 2: This will be executed second. Timeout 1: This will be executed last.
Conclusion	In this experiment, we explored the concepts of callbacks and event loops in Node.js, highlighting their significance in enabling non-blocking I/O operations. By utilizing callbacks, Node.js can efficiently handle multiple tasks concurrently without waiting for each operation to complete, which enhances its scalability and performance. The example demonstrated how asynchronous file reading works, allowing other code to execute while waiting for the file I/O to finish. Understanding these concepts is crucial for developing efficient server-side applications that remain responsive under heavy loads. Overall, callbacks and event loops are foundational to the event-driven architecture of Node.js, making it a powerful tool for modern web development.