SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

Course Code and Name : 19TS601 FULL STACK DEVELOPMENT

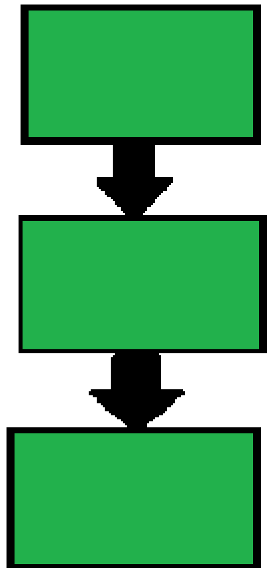
Unit 1 : JAVASCRIPT AND BASICS OF MERN STACK

Topic : Event Loop

Event loop

- Its ability to handle asynchronous tasks efficiently, even though it is single-threaded.
- JavaScript is a **single-threaded language**.
- It means that JavaScript executes code one line at a time, in a sequence.
- The main thread, where all JavaScript code runs, can only do one task at a time, and there is no way to run multiple pieces of code in parallel on this thread.
- This might sound limiting, but JavaScript's design is well-suited for handling many tasks efficiently.

JavaScript is single-threaded



Event loops work

- **Call Stack:**
  - To keep track of the currently executing function (where the program is in its execution).
- **Callback Queue:**
  - Asynchronous operations, such as I/O operations or timers, are handled by the browser or Node.js runtime.
  - When these operations are complete, corresponding functions (callbacks) are placed in the callback queue.
- **Event Loop:**
  - The event loop continuously checks the call stack and the callback queue. If the call stack is empty, it takes the first function from the callback queue and pushes it onto the call stack for execution.

Execution:

- The function on top of the call stack is executed.
- If this function contains asynchronous code, it might initiate further asynchronous operations.

Callback Execution:

- When an asynchronous operation is complete, its callback is placed in the callback queue.

Repeat:

- The event loop continues this process, ensuring that the call stack is always empty before taking the next function from the callback queue.

Microtasks and macrotasks

- In JavaScript, **microtasks** and **macrotasks** refer to different types of tasks in the event loop, which is the mechanism responsible for handling asynchronous operations.
- These tasks are executed in different phases and have different priorities within the event loop.

Macrotasks

- Macrotasks are the larger, higher-level tasks that the event loop processes.
- These include things like I/O operations (network requests, file reading), timers (e.g., setTimeout, setInterval), and user input events (clicks, keypresses).
- **Examples**:setTimeout()
- setInterval()
- I/O operations (e.g., reading files)
- UI rendering events (e.g., painting, reflow)
- Event listeners (e.g., click, keydown)

Microtasks

- Microtasks are smaller tasks that are scheduled to execute after the currently executing script and before the next event loop cycle.
- They have a higher priority than macrotasks and will be executed before any macrotasks.
- **Examples**:Promises (then, catch, finally handlers)
- MutationObserver callbacks
- queueMicrotask() API

Event Loop Execution Order

1. **Start executing synchronous code** (code that doesn't require asynchronous handling).
2. **Execute all microtasks** in the microtask queue (like promise callbacks, MutationObserver, etc.).
3. **Execute one macrotask** from the macrotask queue (like setTimeout, setInterval, event handlers).
4. Repeat the cycle.

- **Microtasks** are executed **before** the browser repaints or handles other macrotasks. Therefore, microtasks are **given higher priority**.
- **Macrotasks** are handled **after** the microtasks are executed, but they are **one per event loop cycle**.
- **Microtasks** are typically used for tasks that need to run after the current task completes but before anything else, like handling promises

```
console.log('Start');
// Schedule a macro task (setTimeout)
setTimeout(() => { console.log('Macrotask 1'); }, 0);
// Schedule a microtask (promise)
Promise.resolve().then(() => { console.log('Microtask 1'); });
// Another macrotask
 setTimeout(() => { console.log('Macrotask 2'); }, 0);
console.log('End');
```

- **Microtasks**: Higher priority, executed after the current script but before macrotasks.
- **Macrotasks**: Lower priority, executed after all microtasks have been processed.

Output

```
Start
End
Microtask 1
Macrotask 1
Macrotask 2
```

To write a program that uses the event loop to simulate a countdown timer

```
let count = 10;
function countdown()
{
if (count > 0)
 {
          console.log(count);
          count--;
          setTimeout(countdown, 1000); // Call countdown again after 
second  }
else
{
console.log("Blast off!");
}
}
console.log("Countdown started...");
countdown();
```

Note:
- Milliseconds (ms): 1/1000th of a second
- Seconds (s): 1000 ms
- Minutes (m): 60 s = 60,000 ms
- Hours (h): 60 m = 3600 s = 3,600,000 ms

Output

```
Countdown started...
10
9
8
7
6
5
4
3
2
1
Blast off!
```

Explanation

1. The countdown function is called initially.
2. The countdown function checks if the count variable is greater than 0.
3. If it is, the function logs the current count to the console, decrements the count, and schedules itself to be called again after 1 second using setTimeout.
4. The event loop takes over, executing other tasks (in this case, logging the count to the console) while waiting for the 1-second timeout to expire.
5. Once the timeout expires, the event loop calls the countdown function again, repeating the process until the count reaches 0.

1.What is event loop?

Text Book:

1.Pro MERN Stack, Full Stack Web App Development with Mongo, Express, React, and Node, Vasan Subramanian, A Press Publisher, 2019.

Reference:

David Flanagan, “Java Script: The Definitive Guide”, O’Reilly Media, Inc, 7 th Edition, 2020

2. Matt Frisbie, “Professional JavaScript for Web Developers” Wiley Publishing, Inc, 4th Edition, ISBN: 978-1-119-36656-0, 2019

Thank You!
dreamstime