**Aim: Introduction to node js and implementation of basic program using javascript.**

**Theory:**

**Introduction to Node.JS:**

Node.js is an open source runtime framework that gives developers the capacity to run JavaScript on the server-end. While JavaScript was historically used for creating dynamic web interfaces on the browser, Node.js has expanded its potential by allowing it to run outside this traditional environment.

**Node.js Highlights:**

**1. Asynchronous Nature:** Node.js is renowned for its asynchronous, event-centric I/O approach, ensuring it remains nimble and efficient. This characteristic allows it to manage numerous simultaneous connections, making it apt for applications that need scalability.

**2. Empowered by V8:** Leveraging Google's V8 JavaScript engine, Node.js translates JavaScript to direct machine instructions, resulting in impressive operational speeds.

**3. NPM Ecosystem:** Accompanied by npm, a comprehensive package manager, Node.js offers developers access to a myriad of pre-written code packages, streamlining application development.

**4. Module-Based:** The emphasis on modular coding ensures that Node.js applications are neat and maintainable. These modules can be seamlessly integrated into various parts of an application.

**5. Diverse Application Use:** Node.js is not limited in its capabilities. It's suitable for crafting everything from robust web servers to real-time communication tools.

**6. Consistent Language Stack:** Since Node.js encompasses JavaScript, it's possible for developers to code both the front and back ends of web platforms in the same language, making the process smoother and more integrated.

**Advantages of Node.js :**

1. Unified Language: Developers can use JavaScript for both client-side and server-side scripting, leading to more efficient development due to a uniform language stack.

2. Performance: Built on Google's V8 JavaScript engine, Node.js compiles JavaScript directly to machine code, enhancing execution speeds.

3. Scalability: Its event-driven, non-blocking architecture allows for high scalability, especially in I/O-bound applications.

4. Active Community: With a vibrant community, there's a constant influx of libraries, modules, and tools via npm, facilitating rapid development.

5. Real-Time Capabilities: Node.js excels in building real-time applications, such as chat applications or online gaming platforms, because of its non-blocking architecture.

6. Flexible and Lightweight: Without a strict convention, developers can build applications in varied ways. This offers flexibility but also demands a clear application structure to be maintained.

**Disadvantages of Node.js:**

1. Not Ideal for CPU-Intensive Tasks: Due to its single-threaded nature, Node.js may not be suitable for applications that require significant CPU processing, as it can become a performance bottleneck.

2. Learning Curve:For those unfamiliar with asynchronous programming, there can be a steep learning curve when diving into Node.js development.

3. Maturity of Tools:While the Node ecosystem is expansive, some modules in npm might not be as mature or well-maintained as desired.

4. Callback Hell: One of the frequent criticisms is the callback pyramid that can emerge due to multiple nested callbacks, though this can be managed with promises, async/await, or other design patterns.

5. Lack of Conventions:The flexibility of Node.js, while a strength, can also be a downfall for larger teams as the absence of clear conventions can lead to inconsistency in codebase.

**REPL:**

REPL stands for **Read Eval Print Loop,** **which** represents a computer environment **such** **as** a Windows console or Unix/Linux shell where **commands** **are** entered and the system responds with output in an interactive **manner.** Node.js or Node comes with a REPL environment. It performs the following **functions:** **-**

**Reading** **-** **reads** **user** input, parses the input into **a** JavaScript **data** **structure** and stores **it** in **memory.**

**Eval** **-** **Retrieve** and **evaluate** **sample** **data.**

**Print** **-** **print** the **results.**

**Loops** **-** Loops the above command until the user presses ctrl-c **twice.**

**Node.JS Process Models:**

At its core, Node.js operates using a unique process model that distinguishes it from traditional server-side languages. This model is based on an event-driven, non-blocking I/O framework that is designed for optimal scalability and performance.

**1. Single-Threaded Event Loop:**

Unlike multi-threaded systems where each request spawns a new thread, Node.js handles multiple connections using a single thread, the main event loop.

This approach reduces the overhead associated with thread context switching and memory allocation.

**2. Non-blocking I/O:**

When a Node.js application needs to perform an I/O task, such as reading from the file system, querying a database, or making an API request, it doesn't wait around for the task to complete.

Instead, it delegates the task and immediately moves on to handle other tasks.

**3. Callback Mechanism:**

Once the I/O operation is finished, a callback function is invoked to process the result, allowing the application to handle the response or errors accordingly.

This is how Node.js can handle many connections simultaneously, without the need for multi-threading.

**4. Event Queue and Event Loop:**

Tasks such as I/O operations, timers, or even manual event emissions are queued in an event queue.

The event loop constantly checks this queue. When an event's corresponding task is complete and the stack is clear, its callback function is executed.

**5. Challenges and Solutions:**

While the single-threaded, non-blocking model offers great performance benefits, it also poses challenges. For instance, CPU-intensive tasks can block the event loop, leading to performance degradation.

To tackle such challenges, Node.js can be integrated with worker threads or external tools to offload heavy computations.

**Traditional Web Server Model**

Before the rise of event-driven architectures, like that of Node.js, web servers primarily operated on a multi-threaded model, which is often termed as the traditional web server model. Here’s a breakdown of how this model functions:

**1. One Thread per Request:**

In traditional models, for each incoming client request, the server typically spawns a new thread or forks a new process to handle that request.

This ensures that each client's request is processed in isolation, avoiding potential interference with other requests.

**2. Blocking I/O:**

Unlike non-blocking architectures, the traditional model often processes I/O operations in a synchronous manner. This means that the server waits for one task to complete before moving on to another, making it "blocking."

If a task, like fetching data from a database, takes a considerable amount of time, the server remains occupied and isn't available to handle other incoming requests using that same thread.

**3. Resource Intensive:**

Due to its multi-threaded nature, this model can become resource-intensive as the number of concurrent users or requests increases.

Each thread consumes memory, and the overhead associated with thread creation and destruction can add significant performance costs, especially under heavy load.

**4. Scalability Concerns:**

While multi-threaded servers can handle a reasonable number of simultaneous connections, they may face scalability issues as the volume grows. That's because the system might reach its thread or process limit.

Horizontal scaling, which involves adding more server machines, is often a go-to solution to handle the increased load.

**5. Mature and Reliable:**

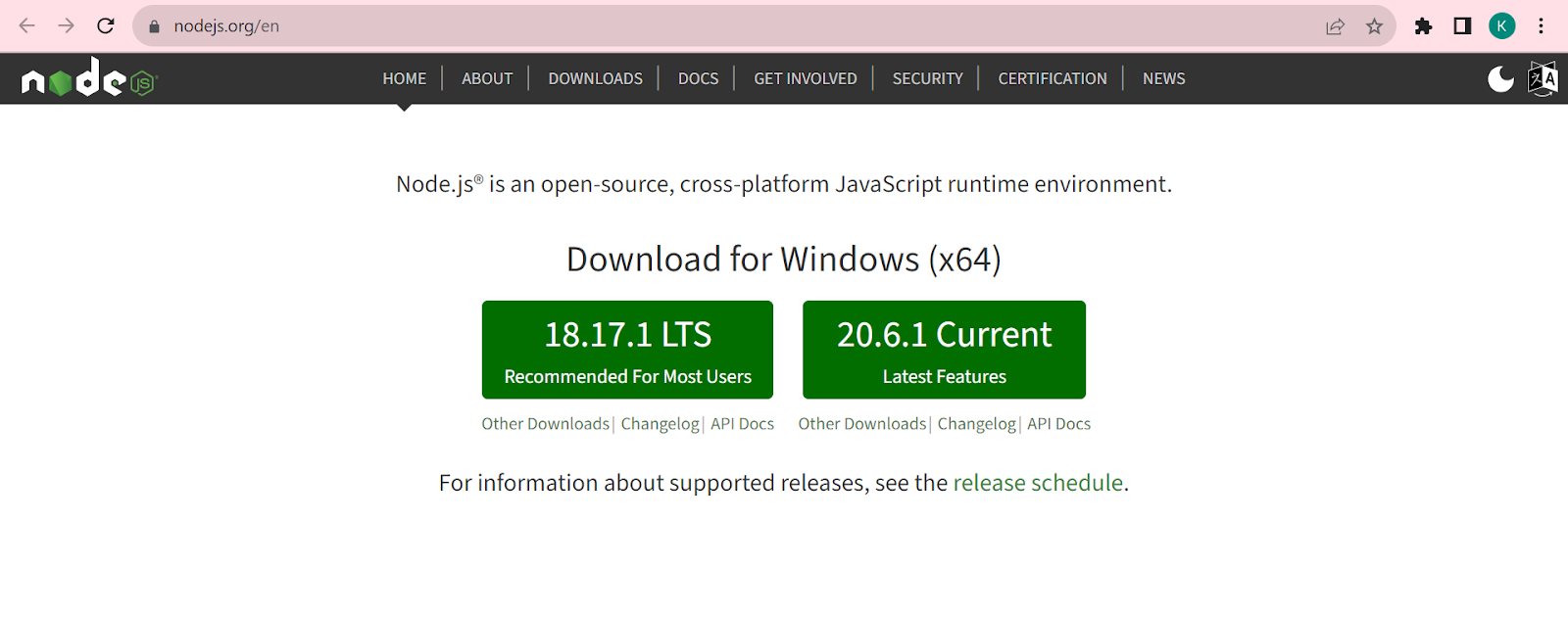
Despite its challenges, the traditional web server model has been around for a long time and is well-understood. Servers like Apache have been using this model for decades, offering robustness and reliability.

Over the years, many optimizations and enhancements have been developed to overcome the model's inherent limitations.

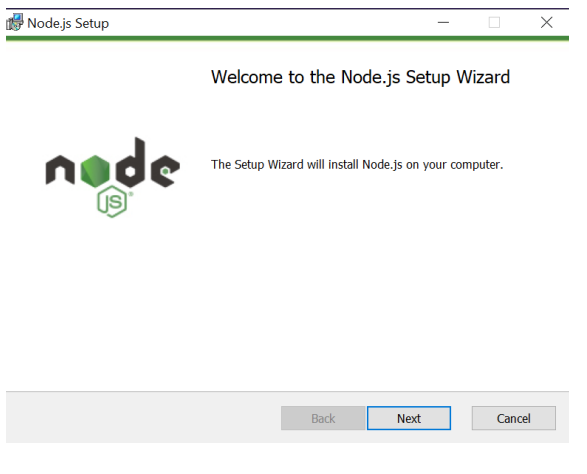
**Installation of Node Js**

Go to the official Node.js website at<https://nodejs.org/>.

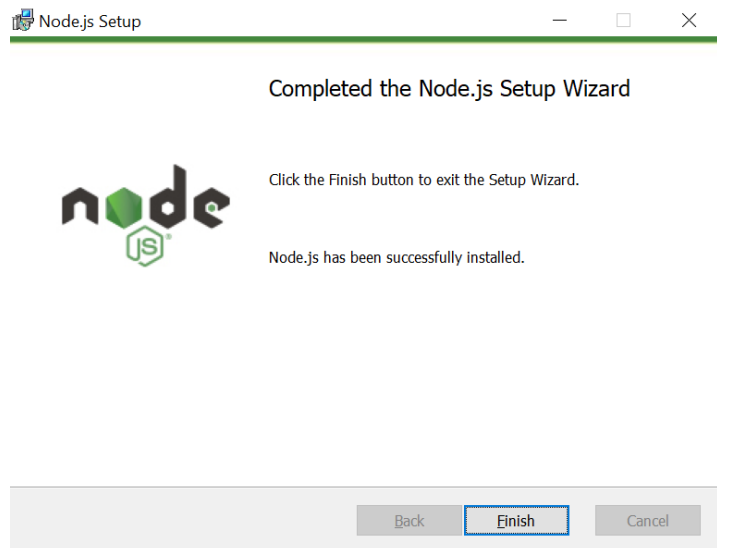
From where you may download according to your needs from the options i.e either to download the latest version or the recommended for most user version :-



After downloading open the installer and run it by locating the file and double click on it :-



Continue with the setup wizard according to the steps until finish :-



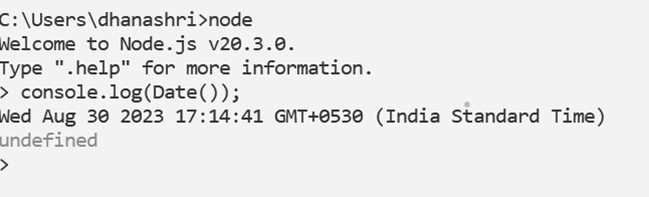
Click on the “Finish” button to complete the installation.

Now in order to verify that node is installed correctly open the command prompt on your pc and run the command :-

node -v

This shall give you the current version of Node.js installed on your pc and confirm correct installation.

1. Print Today’s date and time using REPL:



**2. Write a program to Print the given pattern**

**1 2 3 4 5**

**1 2 3 4**

**1 2 3**

**1 2**

**1**

**Code:**

**//Pattern 01(Question 02)**

**console.log("Pattern 01");**

**string = " ";**

**let n =6;**

**for(let i = 1; i <=n; i++){**

**for(let j=1; j <=n-i; j++){**

**string += j;**

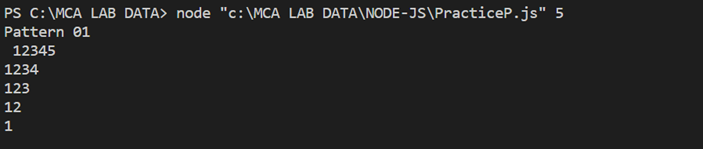
**}**

**string +="\n";**

**}**

**console.log(string);**

**Output:**

****

**3.**

**Code:**

**//Fibonacci series (Question 4 Logic 01)**

**console.log("Fibonacci Series(Logic 01(using while loop)):");**

**let no = process.argv[2];**

**let no1 = 0, no2 = 1,nt;**

**console.log(no1);**

**console.log(no2);**

**nt = no1 + no2;**

**while(nt <= no){**

**console.log(nt);**

**no1 = no2;**

**no2 = nt;**

**nt= no1 + no2;**

**}**

**//Fibonacci series (Question 4 Logic 02)**

**console.log("Fibonacci Series(Logic 02(using for loop)):");**

**let nu = process.argv[2];**

**let nu1 = 0, nu2 = 1,nt1;**

**//console.log(no1);**

**//console.log(no2);**

**nt1 = nu1 + nu2;**

**for(let i=0; i<=nu; i++)**

**{**

**console.log(nu1);**

**nt1= nu1 + nu2;**

**nu1 = nu2;**

**nu2 = nt1;**

**}**

**Output:**

**4.**

**Code:**

**//Print prime numbers from 1 to 100**

**console.log("Print prime numbers 1 to 100(Question 05):");**

**for (let i = 1; i <= 100; i++) {**

**let flag = 0;**

**for (let j = 2; j < i; j++) {**

**if (i % j == 0) {**

**flag = 1;**

**break;**

**}**

**}**

**if (i > 1 && flag == 0) {**

**console.log(i);**

**}**

**}**