NODE JS

What is Node.js?

Node.js is a web application framework built on Google Chrome's JavaScript EngineV8Engine.

Node.js = Runtime Environment + JavaScript Library

Features of Node.js

Asynchronous & Event-Driven: Non-blocking APIs ensure the server doesn't wait for responses, improving efficiency.

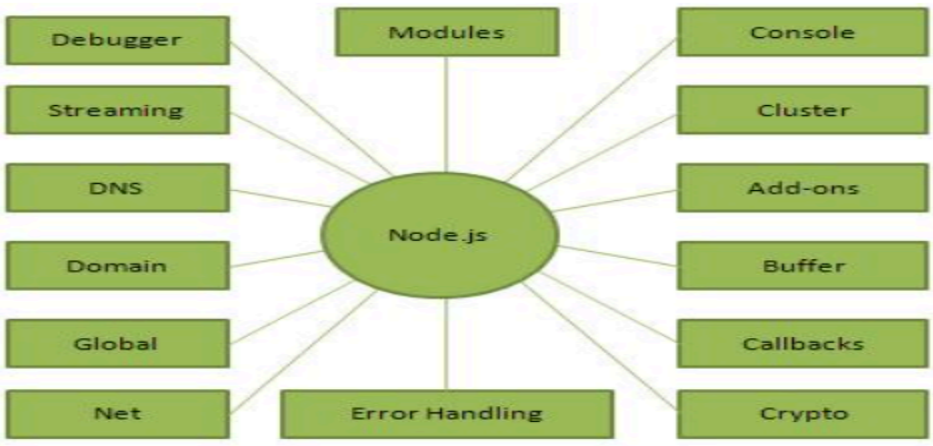
Very Fast: Built on Google Chrome's V8 JavaScript Engine, ensuring quick execution.

Single-Threaded & Scalable: Uses event looping instead of multiple threads, allowing it to handle many requests efficiently.

No Buffering: Outputs data in chunks, reducing memory usage.

Concepts

The following diagram depicts some important parts of Node.js which we will discuss in detail .



Where to Use Node.js?

Following are the areas where Node.js is proving itself as a perfect technology partner.:

I/O bound Applications

Data Streaming Applications

Data Intensive Real-time Applications (DIRT)

JSON APIs based Applications

Single Page Applications

Node.js - REPL Terminal :

REPL: Stands for Read, Eval, Print, Loop – an interactive command-line environment.

Read: Reads and parses user input into a JavaScript data structure.

Eval: Evaluates the input.

Print: Displays the result.

Loop: Repeats the process until the user exits (Ctrl + C twice).

Uses: Helpful for testing, debugging, and experimenting with Node.js code.

Simple Expression

Let's try simple mathematics at REPL command prompt:

```
C:\Nodejs_WorkSpace>node
> 1 + 3
4
> 1 + ( 2 * 3 ) - 4
3
>
```

Multiline Expression

Node REPL supports multiline expression similar to JavaScript. See the following do-while loop in action:

```
C:\Nodejs_WorkSpace> node
> var x = 0
undefined
> do {
... x++;
... console.log("x: " + x);
... } while ( x < 5 );
x: 1
x: 2
x: 3
x: 4
x: 5
undefined
>
```

Underscore variable

Use _ to get the last result.

```
C:\Nodejs_WorkSpace>node
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

REPL Commands:

ctrl + c - terminate the current command.

ctrl + c twice - terminate the Node REPL.

ctrl + d - terminate the Node REPL.

Up/Down Keys - see command history and modify previous commands.

tab Keys - list of current commands.

.help - list of all commands.

.break - exit from multiline expression.

.clear - exit from multiline expression

.save - save current Node REPL session to a file.

.load - load file content in current Node REPL session.

Node.js - Callbacks Concept:

Callback: An asynchronous function executed after a task completes.

Non-Blocking Execution: Node.js APIs use callbacks to avoid waiting for tasks like file I/O.

Example: A file read function starts reading and immediately returns control to execute the next instruction. Once done, it calls the callback with the file content.

Scalability: Enables handling multiple requests efficiently without blocking operations.

Example

Create a js file named test.js in **C:\>Nodejs_WorkSpace**.

File: test.js

```
//import events module
var events = require('events');
//create an eventEmitter object
var eventEmitter = new events.EventEmitter();

//create a function connected which is to be executed
//when 'connection' event occurs
var connected = function connected() {
    console.log('connection succesful.');

    // fire the data_received event
    eventEmitter.emit('data_received.');
}

// bind the connection event with the connected function
eventEmitter.on('connection', connected);

// bind the data_received event with the anonymous function
eventEmitter.on('data_received', function(){
    console.log('data received succesfully.');
});

// fire the connection event
eventEmitter.emit('connection');

console.log("Program Ended.");
```

Node.js - Buffer Module

The Buffer module can be used to create Buffer and SlowBuffer classes. The Buffer module can be imported using the following syntax:

```
var buffer = require("buffer");
```

What are Streams?

Streams are objects that let you read data from a source or write data to a destination in continuous fashion. In Node.js, there are four types of streams :

- **Readable** – Stream which is used for read operation.
- **Writable** – Stream which is used for write operation.
- **Duplex** – Stream which can be used for both read and write operation.
- **Transform** – A type of duplex stream where the output is computed based on input.

Create a js file named main.js with the following code –

```
var fs = require("fs");
var data = '';

// Create a readable stream
var readerStream = fs.createReadStream('input.txt');

// Set the encoding to be utf8.
readerStream.setEncoding('UTF8');

// Handle stream events --> data, end, and error
readerStream.on('data', function(chunk) {
   data += chunk;
});

readerStream.on('end',function() {
   console.log(data);
});

readerStream.on('error', function(err) {
   console.log(err.stack);
});

console.log("Program Ended");
```

Node.js - File System :

The fs module is used for File I/O and can be imported using:

var fs = require("fs");

It provides both synchronous and asynchronous methods. Asynchronous methods take a callback function with an error as the first parameter. Asynchronous methods are preferred as they do not block program execution, whereas synchronous methods do.

Let us create a js file named **main.js** with the following code –

```
var fs = require("fs");

// Asynchronous read
fs.readFile('input.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});

// Synchronous read
var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());

console.log("Program Ended");
```

Methods

Sr. No.	method	Description
1	**fs.rename***oldPath, newPath, callback*	Asynchronous rename. No arguments other than a possible exception are given to the completion callback.
2	**fs.ftruncate***fd, len, callback*	Asynchronous ftruncate. No arguments other than a possible exception are given to the completion
3	**fs.ftruncateSync***fd, len*	Synchronous ftruncate
4	**fs.truncate***path, len, callback*	Asynchronous truncate. No arguments other than a possible exception are given to the completion callback.
5	**fs.truncateSync***path, len*	Synchronous truncate

Flags

Flags for read/write operations are –

Sr.No.	Flag & Description
1	**r** Open file for reading. An exception occurs if the file does not exist.
2	**r+** Open file for reading and writing. An exception occurs if the file does not exist.
3	**rs** Open file for reading in synchronous mode.
4	**rs+** Open file for reading and writing, asking the OS to open it synchronously. See notes for 'rs' about using this with caution.
5	**w** Open file for writing. The file is created (if it does not exist) or truncated (if it exists).
6	**wx** Like 'w' but fails if the path exists.
7	**w+** Open file for reading and writing. The file is created (if it does not exist) or truncated (if it exists).

8	**wx+** Like 'w+' but fails if path exists.
9	**a** Open file for appending. The file is created if it does not exist.
10	**ax** Like 'a' but fails if the path exists.
11	**a+** Open file for reading and appending. The file is created if it does not exist.
12	**ax+** Like 'a+' but fails if the the path exists.

```
var fs = require("fs");
var buffer = new Buffer(1024);

//Example: Opening File
function openFile(){
   console.log("\nOpen file");
   fs.open('test.txt', 'r+', function(err,fd) {
      if (err) console.log(err.stack);
   console.log("File opened");
   });
}

//Example: Getting File Info
function getStats(){
   console.log("\nGetting File Info");
   fs.stat('test.txt', function (err, stats) {
      if (err) console.log(err.stack);
      console.log(stats);
      console.log("isFile ? "+stats.isFile());
      console.log("isDirectory ? "+stats.isDirectory());
   });
}

//Example: Writing File
function writeFile(){
   console.log("\nWrite file");
   fs.open('test1.txt', 'w+', function(err,fd) {
      var data = "TutorialsPoint.com - Simply Easy Learning!";
      buffer.write(data);

      fs.write(fd, buffer,0,data.length,0,function(err, bytes){
         if (err) console.log(err.stack);
      console.log(bytes + " written!");
      });
   });
}
```

Node.js - Console

The console is a global object used to print to std out and stderr. It works synchronously when the destination is a file or terminal and asynchronously when the destination is a pipe.

Sr. No.	method	Description
1	**console.log** [*data*][, . . .]	Prints to stdout with newline. This function can take multiple arguments in a printf-like way.
2	**console.info** [*data*][, . . .]	Prints to stdout with newline. This function can take multiple arguments in a printf-like way.
3	**console.error** [*data*][, . . .]	Prints to stderr with newline. This function can take multiple arguments in a printf-like way.
4	**console.warn** [*data*][, . . .]	Prints to stderr with newline. This function can take multiple arguments in a printf-like way

```
var counter = 10;

console.log("Counter: %d", counter);

console.time("Getting data");
//make a database call to retrive the data
//getDataFromDataBase();
console.timeEnd('Getting data');

console.info("Program Ended!")
```

Verify the Output.

```
Counter: 10
Getting data: 0ms
Program Ended!
```

Events

Process is an event Emitter and it emits the following events.

Sr.No.	Event	Description
1	**exit**	Emitted when the process is about to exit. There is no way to prevent the exiting of the event loop at this point, and once all exit listeners have finished running the process will exit.
2	**beforeExit**	This event is emitted when node empties it's event loop and has nothing else to schedule. Normally, node exits when there is no work scheduled, but a listener for 'beforeExit' can make asynchronous calls, and cause node to continue.
3	**uncaughtException**	Emitted when an exception bubbles all the way back to the event loop. If a listener is added for this exception, the default action *whichistoprintastacktraceandexit* will not occur.

Node.js - DNS Module

The DNS module is used for DNS lookups and utilizes the underlying operating system's name resolution functionalities. It provides an asynchronous network wrapper. The module can be imported using the following syntax

```
var dns = require("dns")
```

RR types:

1. A - IPV4 addresses, default
2. AAAA - IPV6 addresses
3. MX - mail exchange records
4. TXT - text records
5. SRV - SRV records
6. PTR - used for reverse IP lookups
7. NS - name server records
8. CNAME - canonical name records
9. SOA - start of authority record

```
var dns = require('dns');

dns.lookup('www.google.com', function onLookup(err, address, family) {
   console.log('address:', address);
   dns.reverse(address, function (err, hostnames) {
   if (err) {
      console.log(err.stack);
   }

   console.log('reverse for ' + address + ': ' + JSON.stringify(hostnames));
});
});
```

Node.js - Domain Module :

The domain module is used to handle unhandled errors in Node.js. If errors are not managed, they can crash the application. Errors can be intercepted using internal binding (executing within the run method) or external binding (adding an error emitter explicitly using the add method). The module can be imported using:

```javascript
var domain = require("domain");
```

The Domain class, a child of EventEmitter, helps route errors and uncaught exceptions to an active domain object. To handle errors, listen to its error event. A domain is created using:

```javascript
var domain1 = domain.create();
```

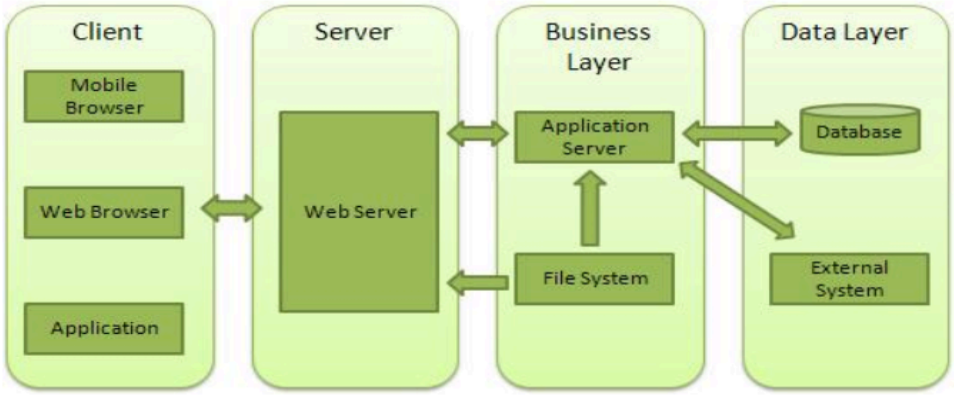
Node.js - Web Module :

A Web Server processes requests using the HTTP protocol and returns web pages to clients. It serves HTML documents, images, stylesheets, and scripts. Many web servers also support server-side scripting or connect to application servers for tasks like database queries and complex logic. The web server then returns the processed output to the client.

Apache Web Server is one of the most commonly used open-source web servers.

Web Application Architecture

A Web application is usually divided into four layers –



- **Client** – This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.
- **Server** – This layer has the Web server which can intercept the requests made by the clients and pass them the response.
- **Business** – This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.
- **Data** – This layer contains the databases or any other source of data.

```
//http module is required to create a web server
var http = require('http');
//fs module is required to read file from file system
var fs = require('fs');
//url module is required to parse the URL passed to server
var url = require('url');

//create the server
http.createServer(function (request, response) {
   //parse the pathname containing file name
   var pathname = url.parse(request.url).pathname;
   //print the name of the file for which request is made.
   //if url is http://localhost:8081/test.htm then
   //pathname will be /test.htm
   console.log("Request for " + pathname + " received.");
   //read the requested file content from file system
   fs.readFile(pathname.substr(1), function (err, data) {
      //if error occured during file read
      //send a error response to client
      //that web page is not found.
      if (err) {
         console.log(err.stack);
         // HTTP Status: 404 : NOT FOUND
         // Content Type: text/plain
         response.writeHead(404, {'Content-Type': 'text/html'});
      }else{
         //Page found
         // HTTP Status: 200 : OK
         // Content Type: text/plain
         response.writeHead(200, {'Content-Type': 'text/html'});
         // write the content of the file to response body
         response.write(data.toString());
      }
      // send the response body
      response.end();
   });
}).listen(8081);
// console will print the message
```

File: test.htm

```
<html>
<head>
<title>Sample Page</title>
</head>
<body>
Hello World!
</body>
</html>
```

Now run the server.js to see the result:

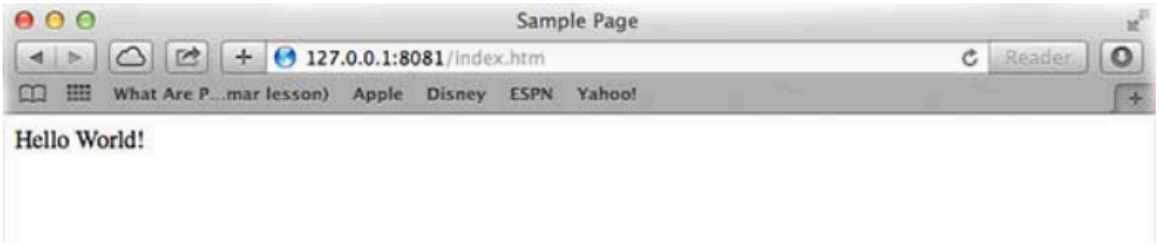
```
C:\Nodejs_WorkSpace>node server.js
```

Verify the Output. Server has started

```
Server running at http://127.0.0.1:8081/
```

Make a request to Node.js server

Open http://127.0.0.1:8081/test.htm in any browser and see the below result.



Creating Web client using Node

A web client can be created using http module. See the below example:

```
//http module is required to create a web client
var http = require('http');

//options are to be used by request
var options = {
   host: 'localhost',
   port: '8081',
   path: '/test.htm'
};


//callback function is used to deal with response
var callback = function(response){
    // Continuously update stream with data
    var body = '';
    response.on('data', function(data) {
        body += data;
    });
    response.on('end', function() {
        // Data received completely.
        console.log(body);
    });
}
//make a request to the server
var req = http.request(options, callback);
req.end();
```

Verify the Output.

```
<html>
<head>
<title>Sample Page</title>
</head>
<body>
Hello World!
</body>
</html>
```

Verify the Output at server end.

```
Server running at http://127.0.0.1:8081/
Request for /test.htm received.
Request for /test.htm received.
```