NodeJS Interview Questions

NodeJS is one of the important framework in MEAN stack. NodeJS is used to program server side part of applications. In this article, we present a set of NodeJS interview questions that could be helpful during interview process. The interview questions are divided into three categories based on the difficulty.

- Entry Level – Node.js Interview Questions
- Intermediate Level – Node.js Interview Questions
- Adavanced Level – Node.js Interview Questions

NodeJS Interview Questions – Entry Level

What is Node.js ?

Node.js is :

- Open Source (Source code of Node.js is available openly and can be modified and built)
- Cross Platform (Works on different operating systems)
- Asynchronous
- Event Driven (Web Requests are considered as events)
- JavaScript Runtime Environment (JavaScript is run outside the browser)

Which is the scripting/programming language used for Node.js application programming ?

The answer is one word. JavaScript.

How is Node.js different from previous server side programming frameworks ?

Previous frameworks were not event driven. Node.js is a single thread driven.

What are the features of Node.js ?

Some of the important features of Node.js are :

- Asynchronous and Event Driven
- Code Execution is very fast since Node.js is built on Chrome's V8 JavaScript Engine.
- Highly Scalable because of Event Looping.

How is event-driven programming ?

Requests to the Node.js are considered as events. When a request is made, Node.js server accepts the request, attaches it to a callback, and starts preparing the response. When the response is being prepared, the server does not wait; it takes in other requests from clients. Whenever a response is ready, it is served via callback function. Thus everything is considered an event,

What is MEAN stack ? What is the place of Node.js in the MEAN stack ?

MEAN is a JavaScript software stack. It is used for building dynamic websites and web applications. MEAN Stack components are :

1. **M**ongoDB
2. **E**xpress.js
3. **A**ngular.js or Angular
4. **N**ode.js

How do you install a module in node ?

A module is installed in node using NPM command line interface, npm. The command to install a module is : `npm install <package>`

What is npm ?

npm stands for Node Package Manager. npm is the package manager for JavaScript.

How is local installation of a dependency done ?

The default behavior of npm is to install the package/module local to the project. To install a dependency locally, navigate to the project folder and run the following command. `npm install <package>`

How is global installation of a dependency done ?

To install a dependency globally, i.e., the module can be used in any node project on the machine, use global option in the npm install command as shown below : `npm install <package> --global` or `npm install <package> --g`

How do you uninstall a dependency using npm ?

To uninstall a package or dependency, run npm unistall command with the package name to be removed. `npm uninstall <package>`

What is the command for importing a package.

`require` command is used to include a package/module in a script file.

Example to include `fs` module in the script file is :

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


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


How to add new functionalities to a module ?

New functions could be added to the existing modules. The step by step process is :

1. Include the module in the script file using `require` statement.
2. Add a function to the module using dot (.) operator.
3. Export the module for the changes to take effect.

Following is an example where a new function named `printMessage` is added to `fs` module.

```
// include the module
that you like extend
```

```
// include the module that you like extend
var fs = require('fs');
// add a new function, printMessage(), to the module
fs.printMessage = function(str){
    console.log("Message from newly added function to the module");
    console.log(str);
}
// re-export the module for changes to take effect
module.exports = fs
// you may use the newly added function
fs.printMessage("Success");
```

Reference : Node.js – Add new functions to module

How to override functions of module ?

As required for the project, existing functions could be overridden.

The step by step process to override functions of a module is :

1. Include the module in the script file.
2. Delete the function (whose functionality has to be altered) using module variable.
3. Add function with the same to the module variable.
4. Export the module for the changes to take effect.

Following is an example, to demonstrate the same :

```
// include the module
whose functions are to
```

```
// include the module whose functions are to be overridden
var fs = require('fs');
// delete the function you would like to override
delete fs['readFile'];
// add new functional with the same name as deleted function
fs.readFile = function(str){
    console.log("The functionality has been overridden.");
    console.log(str);
}
// re-export the module for changes to take effect
module.exports = fs
// you may use the newly overriden function
fs.readFile("sample.txt");
```

Reference : Node.js – Override functions of a module

What is a callback ?

Callback function is used in Node.js to deal with event handling.

When an asynchronous function is called upon a resource for some task, the control is let immediately to continue with the subsequent statements after the function. The task on the resource would start in parallel. This helps Node.js continue with other tasks while the function is working with the resource. Once the task with the resource is completed, Node.js resumes with the callback function. Callback function is called with arguments : data object, result object and (or) error object containing information regarding the task.

An example for Node.js Callback function is :

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

```
var fs = require('fs');
// read file sample.txt
fs.readFile('sample.txt',
    // callback function that is called when reading file is done
    function(err, data) {
        if (err) throw err;
        // data is a buffer containing file content
        console.log("Reading file completed : " + new Date().toISOString());
});
console.log("After readFile asynchronously : " + new Date().toISOString());
```

Reference : Node.js – Callback Function

What is callback hell ?

Callback hell is a situation when callback functions are nested. Nesting a callback function in another callback function leads to unmanageable code.

How can you avoid callback hell ?

There are many procedures to avoid callback hell. Some of them are ;

- Async
  - async.waterfall()
  - async.series()
- Promises

What is a Promise in Node.js ?

A `Promise` is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers to an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of the final value, the asynchronous method returns a *promise* for the value at some point in the future.

What module do you use for accessing/modifying files in Node.js ?

Node FS (File System) can be used for file operations. Node FS is a builtin module. It can be included in the script using the following statement :

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


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


Some of the functions fs module provides are :

- fs.readFile()
- fs.writeFile()
- fs.appendFile()
- fs.open()
- fs.rename()
- fs.unlink()

The above mentioned functions are asynchronous. Synchronous variants for these are also available.

What is chaining in Node.js ?

Each function of a module returns the modified variable. So, another function call can be concatenated to the previous function call.

Following is a non-working example for chaining in Node.js :

```
var.someFunction().an
otherFunction().someA
```

```
var.someFunction().anotherFunction().someAnotherFunc();
```

How to exit from a Node.js Script ?

The execution of a Node.js script happens sequentially. For some reason, if you want to exit, the built-in `process` module has `exit()` method.

```
process.exit(-1);
```

```
process.exit(-1);
```

How to ensure that the dependencies in a Node.js project are secure ?

How do you redirect a url to other in Node.js ?

`http` module can be used to setup an url redirect.

In the following example, we created an HTTP server. When the server gets a request for a resource, we can send a redirect response back to the browser using `response.writeHead()` function. The response code should be `301` for a redirect and the redirect url is provided as well in writeHead() function.

redirectUrlExample.js

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

```
var http = require('http');
var fs = require('fs');
// create a http server
http.createServer(function (req, res) {

    if (req.url == '/page-c.html') {
        // redirect to page-b.html with 301 (Moved Permanently) HTTP code in the response
        res.writeHead(301, { "Location": "http://" + req.headers['host'] + '/page-b.html' });
        return res.end();
    } else {
        // for other URLs, try responding with the page
        console.log(req.url)
        // read requested file
        fs.readFile(req.url.substring(1),
            function(err, data) {
                if (err) throw err;
                res.writeHead(200);
                res.write(data.toString('utf8'));
                return res.end();
        });
    }
}).listen(8085);
```

What is Buffer in Node.js ?

Node.js Buffer is a class that helps to handle and work with octet streams. Octet streams would generally come into picture when dealing with TCP data streams and file system operations.

Give us an example to create HTTP web server ?

We can use `http` module to create a HTTP web server in Node.js.

Brief steps to create HTTP web server :

1. Include http module.
2. Create http server to listen on a port.
3. Write program statements to prepare response.
4. Run the script to start the web server.
5. From the same machine, you can request the server using localhost:port

An example script to create a web server is :

```
// include http module
in the file
```

```
// include http module in the file
var http = require('http');
// create a server
http.createServer(function (req, res) {
    // http header
    // 200 - is the OK message
    // to respond with html content, 'Content-Type' should be 'text/html'
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write('Node.js says hello!'); //write a response to the client
    res.end(); //end the response
}).listen(9000); //the server object listens on port 9000
```

Reference : Node.js – Create HTTP Web Server

How do you connect your Node.js application to MySQL ?

From Node.js, we can connect to MySQL server. We can use `mysql` module.

1. Include `mysql` module in the script file.
2. Create a connection variable with the information : MySQL Server IP, Port, Username, Password.
3. Call connect method using connection variable. If the setup is fine, the callback function should receive a null error object.

An example Node.js script to connect to MySQL server :

```
// include mysql module
var mysql =
```

```
// include mysql module
var mysql = require('mysql');
// create a connection variable with the details required
var con = mysql.createConnection({
  host: "localhost", // ip address of server running mysql
  user: "arjun", // user name to your mysql database
  password: "password" // corresponding password
});
// connect to the database.
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

Reference : Node.js – Connect to MySQL Database

What is a result object in Node.js MySQL ?

When a MySQL Query is executed in Node.js, an object called Result Object is returned to the callback function. The Result Object contains result set or properties that provide information regarding the execution of a query in MySQL Server.

The data in result object depends on the type of query that is made to MySQL Server.

How do you connect to MongoDB from Node.js application ?

What is try-catch in Node.js ?

Write a program to upload files to Node.js server ?

- Node.js - Convert JSON to Buffer
- Node.js - Array to Buffer

Node.js HTTP

- Node.js - Create HTTP Web Server
- Node.js - Redirect URL

Node.js MySQL

- Node.js MySQL
- Node.js MySQL - Connect to MySQL Database
- Node.js MySQL - SELECT FROM
- Node.js MySQL - SELECT WHERE
- Node.js MySQL - ORDER BY
- Node.js MySQL - INSERT INTO
- Node.js MySQL - UPDATE
- Node.js MySQL - DELETE
- Node.js MySQL - Result Object

Node.js MongoDB

- Node.js MongoDB
- Node.js - Connect to MongoDB
- Node.js - Create Database in MongoDB
- Node.js - Drop Database in MongoDB
- Node.js - Create Collection in MongoDB
- Node.js - Delete Collection in MongoDB
- Node.js - Insert Documents to MongoDB Collection
- MongoError: failed to connect to server

Node.js Mongoose

- Node.js Mongoose Tutorial
- Node.js Mongoose - Installation
- Node.js Mongoose - Connect to MongoDB
- Node.js Mongoose - Define a Model
- Node.js Mongoose - Insert Single Document to MongoDB
- Node.js Mongoose - Insert Multiple Documents to MongoDB

Node.js URL

- Node.js - Parse URL parameters

Node.js FS (File System)

- Node FS
- Node FS - Read a File
- Node FS - Create a File
- Node FS - Write to a File
- Node FS - Append to a File
- Node FS - Rename a File
- Node FS - Delete a File
- Node FS Extra - Copy a Folder

Node.js JSON

- Node.js Parse JSON
- Node.js Write JSON Object to File

Node.js Error Handling

- Node.js Try Catch

Node.js Examples

- Node.js Examples
- Node.js - Handle Get Requests
- Node.js Example - Upload files to Node.js server

Useful Resources

- Node.js Interview Questions
- How to Learn Programming