Express.js Tutorial

Express.js tutorial provides basic and advanced concepts of Express.js. Our Express.js tutorial is designed for beginners and professionals both.

Express.js is a web framework for Node.js. It is a fast, robust and asynchronous in nature.

Our Express.js tutorial includes all topics of Express.js such as Express.js installation on windows and linux, request object, response object, get method, post method, cookie management, scaffolding, file upload, template etc.

What is Express.js

Express is a fast, assertive, essential and moderate web framework of Node.js. You can assume express as a layer built on the top of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.

Let's see some of the core features of Express framework:

* It can be used to design single-page, multi-page and hybrid web applications.
* It allows to setup middlewares to respond to HTTP Requests.
* It defines a routing table which is used to perform different actions based on HTTP method and URL.
* It allows to dynamically render HTML Pages based on passing arguments to templates.

Why use Express

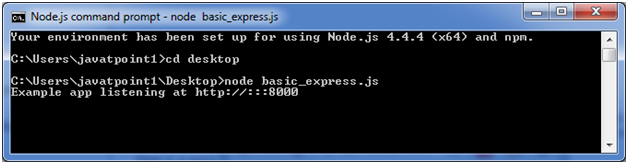
* Ultra fast I/O
* Asynchronous and single threaded
* MVC like structure
* Robust API makes routing easy

How does Express look like

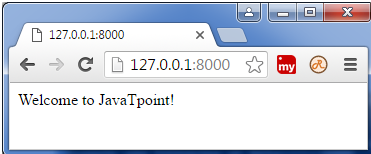
Let's see a basic Express.js app.

**File: basic\_express.js**

1. var express = require('express');
2. var app = express();
3. app.get('/', function (req, res) {
4. res.send('Welcome to JavaTpoint!');
5. });
6. var server = app.listen(8000, function () {
7. var host = server.address().address;
8. var port = server.address().port;
9. console.log('Example app listening at http://%s:%s', host, port);
10. });



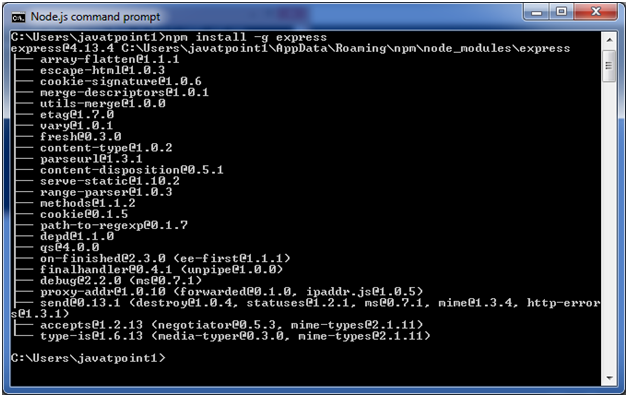
**Output:**



Install Express.js

Firstly, you have to install the express framework globally to create web application using Node terminal. Use the following command to install express framework globally.

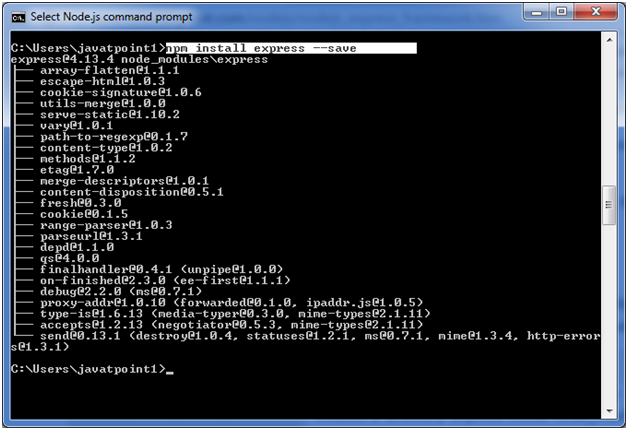
1. npm install -g express



Installing Express

Use the following command to install express:

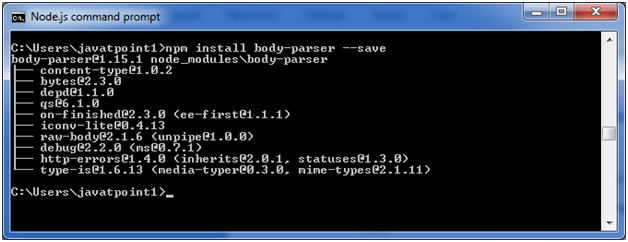
1. npm install express --save



The above command install express in node\_module directory and create a directory named express inside the node\_module. You should install some other important modules along with express. Following is the list:

* **body-parser:** This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
* **cookie-parser:** It is used to parse Cookie header and populate req.cookies with an object keyed by the cookie names.
* **multer:** This is a node.js middleware for handling multipart/form-data.

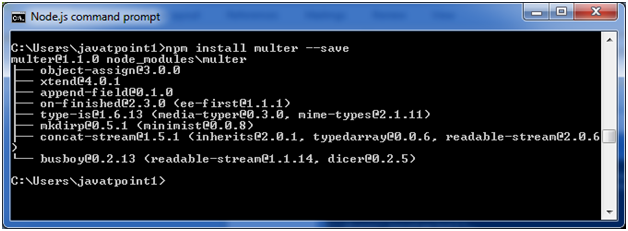
1. npm install body-parser --save



1. npm install cookie-parser --save



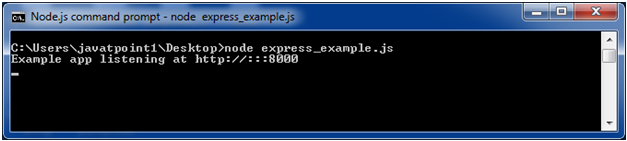
1. npm install multer --save



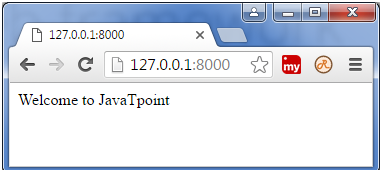
Express.js App Example

Let's take a simple Express app example which starts a server and listen on a local port. It only responds to homepage. For every other path, it will respond with a 404 Not Found error.

1. File: express\_example.js
2. var express = require('express');
3. var app = express();
4. app.get('/', function (req, res) {
5. res.send('Welcome to JavaTpoint');
6. })
7. var server = app.listen(8000, function () {
8. var host = server.address().address
9. var port = server.address().port
10. console.log("Example app listening at http://%s:%s", host, port)
11. })



Open http://127.0.0.1:8000/ in your browser to see the result.



Express.js Request Object

Express.js Request and Response objects are the parameters of the callback function which is used in Express applications.

The express.js request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

**Syntax:**

1. app.get('/', function (req, res) {
2. // --
3. })

Express.js Request Object Properties

The following table specifies some of the properties associated with request object.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

|  |  |  |
| --- | --- | --- |
| **Index** | **Properties** | **Description** |
| 1. | req.app | This is used to hold a reference to the instance of the express application that is using the middleware. |
| 2. | req.baseurl | It specifies the URL path on which a router instance was mounted. |
| 3. | req.body | It contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as body-parser. |
| 4. | req.cookies | When we use cookie-parser middleware, this property is an object that contains cookies sent by the request. |
| 5. | req.fresh | It specifies that the request is "fresh." it is the opposite of req.stale. |
| 6. | req.hostname | It contains the hostname from the "host" http header. |
| 7. | req.ip | It specifies the remote IP address of the request. |
| 8. | req.ips | When the trust proxy setting is true, this property contains an array of IP addresses specified in the ?x-forwarded-for? request header. |
| 9. | req.originalurl | This property is much like req.url; however, it retains the original request URL, allowing you to rewrite req.url freely for internal routing purposes. |
| 10. | req.params | An object containing properties mapped to the named route ?parameters?. For example, if you have the route /user/:name, then the "name" property is available as req.params.name. This object defaults to {}. |
| 11. | req.path | It contains the path part of the request URL. |
| 12. | req.protocol | The request protocol string, "http" or "https" when requested with TLS. |
| 13. | req.query | An object containing a property for each query string parameter in the route. |
| 14. | req.route | The currently-matched route, a string. |
| 15. | req.secure | A Boolean that is true if a TLS connection is established. |
| 16. | req.signedcookies | When using cookie-parser middleware, this property contains signed cookies sent by the request, unsigned and ready for use. |
| 17. | req.stale | It indicates whether the request is "stale," and is the opposite of req.fresh. |
| 18. | req.subdomains | It represents an array of subdomains in the domain name of the request. |
| 19. | req.xhr | A Boolean value that is true if the request's "x-requested-with" header field is "xmlhttprequest", indicating that the request was issued by a client library such as jQuery |

Request Object Methods

Following is a list of some generally used request object methods:

req.accepts (types)

This method is used to check whether the specified content types are acceptable, based on the request's Accept HTTP header field.

**Examples:**

1. req.accepts('html');
2. //=>?html?
3. req.accepts('text/html');
4. // => ?text/html?

req.get(field)

This method returns the specified HTTP request header field.

**Examples:**

1. req.get('Content-Type');
2. // => "text/plain"
3. req.get('content-type');
4. // => "text/plain"
5. req.get('Something');
6. // => undefined

req.is(type)

This method returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter.

**Examples:**

1. // With Content-Type: text/html; charset=utf-8
2. req.is('html');
3. req.is('text/html');
4. req.is('text/\*');
5. // => true

req.param(name [, defaultValue])

This method is used to fetch the value of param name when present.

**Examples:**

1. // ?name=sasha
2. req.param('name')
3. // =**>** "sasha"
4. // POST name=sasha
5. req.param('name')
6. // =**>** "sasha"
7. // /user/sasha for /user/:name
8. req.param('name')
9. // =**>** "sasha"

Express.js Response Object

The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.

What it does

* It sends response back to the client browser.
* It facilitates you to put new cookies value and that will write to the client browser (under cross domain rule).
* Once you res.send() or res.redirect() or res.render(), you cannot do it again, otherwise, there will be uncaught error.

Response Object Properties

Let's see some properties of response object.

|  |  |  |
| --- | --- | --- |
| **Index** | **Properties** | **Description** |
| 1. | res.app | It holds a reference to the instance of the express application that is using the middleware. |
| 2. | res.headersSent | It is a Boolean property that indicates if the app sent HTTP headers for the response. |
| 3. | res.locals | It specifies an object that contains response local variables scoped to the request |

Response Object Methods

Following are some methods:

Response Append method

**Syntax:**

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. res.append(field [, value])

This method appends the specified value to the HTTP response header field. That means if the specified value is not appropriate then this method redress that.

**Examples:**

1. res.append('Link', ['<http://localhost/>', '<http://localhost:3000/>']);
2. res.append('Warning', '199 Miscellaneous warning');

Response Attachment method

**Syntax:**

1. res.attachment([filename])

This method facilitates you to send a file as an attachment in the HTTP response.

**Examples:**

1. res.attachment('path/to/js\_pic.png');

Response Cookie method

**Syntax:**

1. res.cookie(name, value [, options])

This method is used to set a cookie name to value. The value can be a string or object converted to JSON.

**Examples:**

1. res.cookie('name', 'Aryan', { domain: '.xyz.com', path: '/admin', secure: **true** });
2. res.cookie('Section', { Names: [Aryan,Sushil,Priyanka] });
3. res.cookie('Cart', { items: [1,2,3] }, { maxAge: 900000 });

Response ClearCookie method

**Syntax:**

1. res.clearCookie(name [, options])

As the name specifies, the clearCookie method is used to clear the cookie specified by name.

**Examples:**

**To set a cookie**

1. res.cookie('name', 'Aryan', { path: '/admin' });

**To clear a cookie:**

1. res.clearCookie('name', { path: '/admin' });

Response Download method

**Syntax:**

1. res.download(path [, filename] [, fn])

This method transfers the file at path as an "attachment" and enforces the browser to prompt user for download.

**Example:**

1. res.download('/report-12345.pdf');

Response End method

**Syntax:**

1. res.end([data] [, encoding])

This method is used to end the response process.

**Example:**

1. res.end();
2. res.status(404).end();

Response Format method

**Syntax:**

1. res.format(object)

This method performs content negotiation on the Accept HTTP header on the request object, when present.

**Example:**

1. res.format({
2. 'text/plain': function(){
3. res.send('hey');
4. },
5. 'text/html': function(){
6. res.send('
7. hey');
8. },
9. 'application/json': function(){
10. res.send({ message: 'hey' });
11. },
12. 'default': function() {
13. // log the request and respond with 406
14. res.status(406).send('Not Acceptable');
15. }
16. });

Response Get method

**Syntax:**

1. res.get(field)

This method provides HTTP response header specified by field.

**Example:**

1. res.get('Content-Type');

Response JSON method:

**Syntax:**

1. res.json([body])

This method returns the response in JSON format.

**Example:**

1. res.json(**null**)
2. res.json({ name: 'ajeet' })

Response JSONP method

**Syntax:**

1. res.jsonp([body])

This method returns response in JSON format with JSONP support.

**Examples:**

1. res.jsonp(null)
2. res.jsonp({ name: 'ajeet' })

Response Links method

**Syntax:**

1. res.links(links)

This method populates the response?s Link HTTP header field by joining the links provided as properties of the parameter.

**Examples:**

1. res.links({
2. next: 'http://api.rnd.com/users?page=5',
3. last: 'http://api.rnd.com/users?page=10'
4. });

Response Location method

**Syntax:**

1. res.location(path)

This method is used to set the response location HTTP header field based on the specified path parameter.

**Examples:**

1. res.location('http://xyz.com');

Response Redirect method

**Syntax:**

1. res.redirect([status,] path)

This method redirects to the URL derived from the specified path, with specified HTTP status

**Examples:**

1. res.redirect('http://example.com');

Response Render method

**Syntax:**

1. res.render(view [, locals] [, callback])

This method renders a view and sends the rendered HTML string to the client.

**Examples:**

1. // send the rendered view to the client
2. res.render('index');
3. // pass a local variable to the view
4. res.render('user', { name: 'aryan' }, function(err, html) {
5. // ...
6. });

Response Send method

**Syntax:**

1. res.send([body])

This method is used to send HTTP response.

**Examples:**

1. res.send(**new** Buffer('whoop'));
2. res.send({ some: 'json' });
3. res.send('
4. .....some html
5. ');

Response sendFile method

**Syntax:**

1. res.sendFile(path [, options] [, fn])

This method is used to transfer the file at the given path. It sets the Content-Type response HTTP header field based on the filename's extension.

**Examples:**

1. res.sendFile(fileName, options, function (err) {
2. // ...
3. });

Response Set method

**Syntax:**

1. res.set(field [, value])

This method is used to set the response of HTTP header field to value.

**Examples:**

1. res.set('Content-Type', 'text/plain');
3. res.set({
4. 'Content-Type': 'text/plain',
5. 'Content-Length': '123',
6. })

Response Status method

**Syntax:**

1. res.status(code)

This method sets an HTTP status for the response.

**Examples:**

1. res.status(403).end();
2. res.status(400).send('Bad Request');

Response Type method

**Syntax:**

1. res.type(type)

This method sets the content-type HTTP header to the MIME type.

**Examples:**

1. res.type('.html');              // =**>** 'text/html'
2. res.type('html');               // =**>** 'text/html'
3. res.type('json');               // =**>** 'application/json'
4. res.type('application/json');   // =**>** 'application/json'
5. res.type('png');                // =**>** image/png:

Express.js GET Request

GET and POST both are two common HTTP requests used for building REST API's. GET requests are used to send only limited amount of data because data is sent into header while POST requests are used to send large amount of data because data is sent in the body.

Express.js facilitates you to handle GET and POST requests using the instance of express.

Express.js GET Method Example 1

**Fetch data in JSON format:**

Get method facilitates you to send only limited amount of data because data is sent in the header. It is not secure because data is visible in URL bar.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

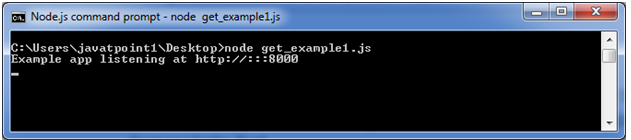
Let's take an example to demonstrate GET method.

**File: index.html**

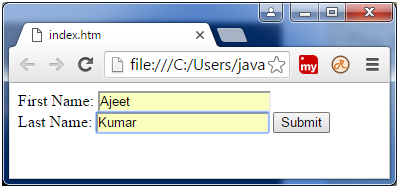
1. **<html>**
2. **<body>**
3. **<form** action="http://127.0.0.1:8081/process\_get" method="GET"**>**
4. First Name: **<input** type="text" name="first\_name"**>**  **<br>**
5. Last Name: **<input** type="text" name="last\_name"**>**
6. **<input** type="submit" value="Submit"**>**
7. **</form>**
8. **</body>**
9. **</html>**

**File: get\_example1.js**

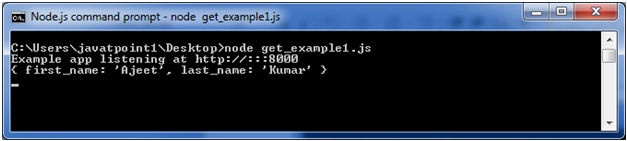
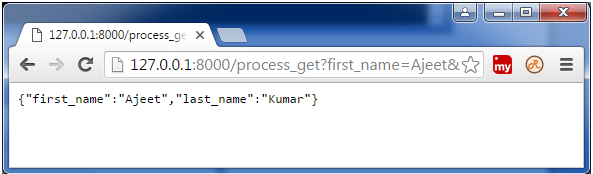
1. var express = require('express');
2. var app = express();
3. app.use(express.**static**('public'));
5. app.get('/index.html', function (req, res) {
6. res.sendFile( \_\_dirname + "/" + "index.html" );
7. })
8. app.get('/process\_get', function (req, res) {
9. response = {
10. first\_name:req.query.first\_name,
11. last\_name:req.query.last\_name
12. };
13. console.log(response);
14. res.end(JSON.stringify(response));
15. })
16. var server = app.listen(8000, function () {
18. var host = server.address().address
19. var port = server.address().port
20. console.log("Example app listening at http://%s:%s", host, port)
22. })



Open the page index.html and fill the entries:



Now, you get the data in JSON format.

Express.js GET Method Example 2

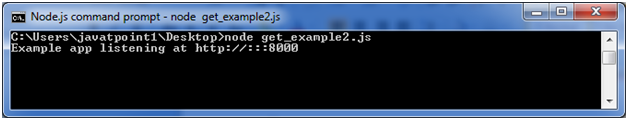
**Fetch data in paragraph format**

**File: index.html**

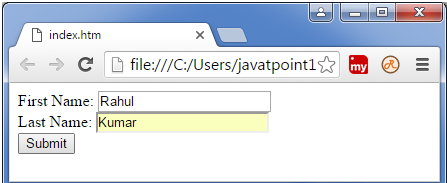
1. **<html>**
2. **<body>**
3. **<form** action="http://127.0.0.1:8000/get\_example2" method="GET"**>**
4. First Name: **<input** type="text" name="first\_name"**/>**  **<br/>**
5. Last Name: **<input** type="text" name="last\_name"**/><br/>**
6. **<input** type="submit" value="Submit"**/>**
7. **</form>**
8. **</body>**
9. **</html>**

**File: get\_example2.js**

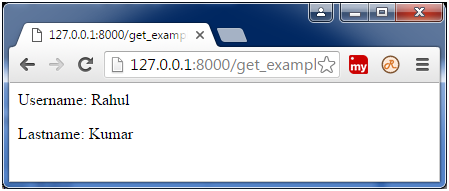
1. var express = require('express');
2. var app=express();
3. app.get('/get\_example2', function (req, res) {
4. res.send('**<p>**Username: ' + req.query['first\_name']+'**</p><p>**Lastname: '+req.query['last\_name']+'**</p>**');
5. })
6. var server = app.listen(8000, function () {
7. var host = server.address().address
8. var port = server.address().port
9. console.log("Example app listening at http://%s:%s", host, port)
10. })



Open the page index.html and fill the entries:



**Output:**



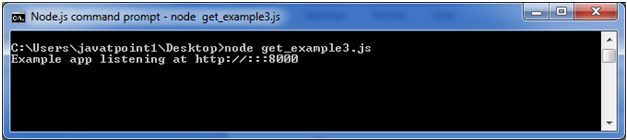
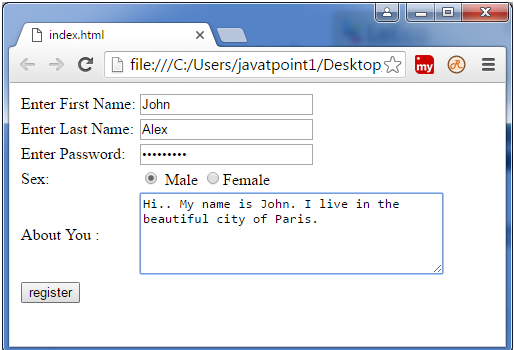
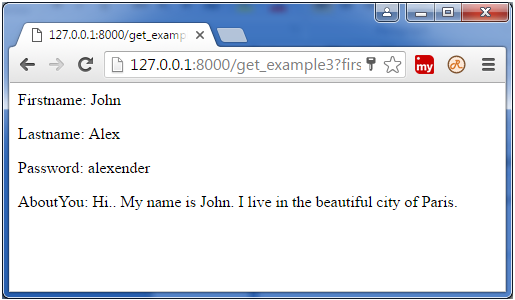
Express.js GET Method Example 3

**File:index.html**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<body>**
4. **<form** action="http://127.0.0.1:8000/get\_example3"**>**
5. **<table>**
6. **<tr><td>**Enter First Name:**</td><td><input** type="text" name="firstname"**/><td></tr>**
7. **<tr><td>**Enter Last Name:**</td><td><input** type="text" name="lastname"**/><td></tr>**
8. **<tr><td>**Enter Password:**</td><td><input** type="password" name="password"**/></td></tr>**
9. **<tr><td>**Sex:**</td><td>**
10. **<input** type="radio" name="sex" value="male"**>** Male
11. **<input** type="radio" name="sex" value="female"**>**Female
12. **</td></tr>**
13. **<tr><td>**About You :**</td><td>**
14. **<textarea** rows="5" cols="40" name="aboutyou" placeholder="Write about yourself"**>**
15. **</textarea>**
16. **</td></tr>**
17. **<tr><td** colspan="2"**><input** type="submit" value="register"**/></td></tr>**
18. **</table>**
19. **</form>**
20. **</body>**
21. **</html>**

**File: get\_example3.js**

1. var express = require('express');
2. var app=express();
4. app.get('/get\_example3', function (req, res) {
5. res.send('<p>Firstname: ' + req.query['firstname']+'</p>
6. <p>Lastname: '+req.query['lastname']+'</p><p>Password: '+req.query['password']+'</p>
7. <p>AboutYou: '+req.query['aboutyou']+'</p>');
8. })
10. var server = app.listen(8000, function () {
11. var host = server.address().address
12. var port = server.address().port
13. console.log("Example app listening at http://%s:%s", host, port)
14. })

# Express.js POST Request

GET and POST both are two common HTTP requests used for building REST API's. POST requests are used to send large amount of data.

Express.js facilitates you to handle GET and POST requests using the instance of express.

## Express.js POST Method

Post method facilitates you to send large amount of data because data is send in the body. Post method is secure because data is not visible in URL bar but it is not used as popularly as GET method. On the other hand GET method is more efficient and used more than POST.

Let's take an example to demonstrate POST method.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

**Example1:**

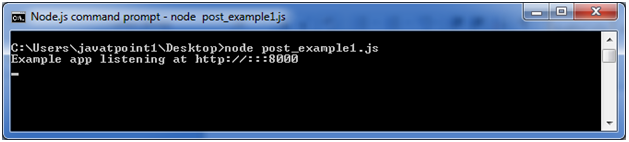
**Fetch data in JSON format**

**File: Index.html**

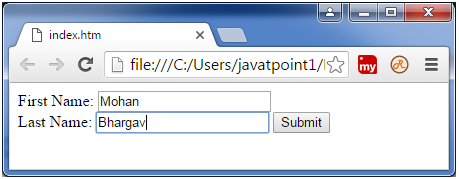
1. **<html>**
2. **<body>**
3. **<form** action="http://127.0.0.1:8000/process\_post" method="POST"**>**
4. First Name: **<input** type="text" name="first\_name"**>**  **<br>**
5. Last Name: **<input** type="text" name="last\_name"**>**
6. **<input** type="submit" value="Submit"**>**
7. **</form>**
8. **</body>**
9. **</html>**

**File: post\_example1.js**

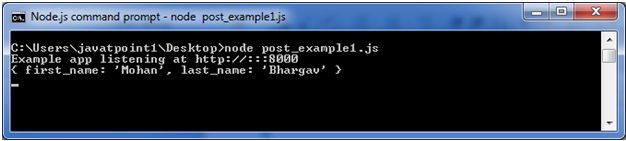
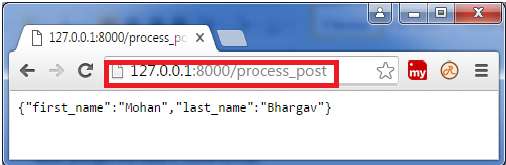
1. var express = require('express');
2. var app = express();
3. var bodyParser = require('body-parser');
4. // Create application/x-www-form-urlencoded parser
5. var urlencodedParser = bodyParser.urlencoded({ extended: false })
6. app.use(express.static('public'));
7. app.get('/index.html', function (req, res) {
8. res.sendFile( \_\_dirname + "/" + "index.html" );
9. })
10. app.post('/process\_post', urlencodedParser, function (req, res) {
11. // Prepare output in JSON format
12. response = {
13. first\_name:req.body.first\_name,
14. last\_name:req.body.last\_name
15. };
16. console.log(response);
17. res.end(JSON.stringify(response));
18. })
19. var server = app.listen(8000, function () {
20. var host = server.address().address
21. var port = server.address().port
22. console.log("Example app listening at http://%s:%s", host, port)
23. })



Open the page index.html and fill the entries:



Now, you get the data in JSON format.

#### **Note: In the above picture, you can see that entries are not visible in the URL bar unlike GET method.**

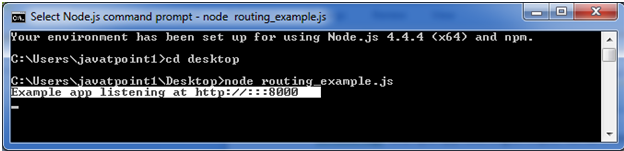
Express.js Routing

Routing is made from the word route. It is used to determine the specific behavior of an application. It specifies how an application responds to a client request to a particular route, URI or path and a specific HTTP request method (GET, POST, etc.). It can handle different types of HTTP requests.

Let's take an example to see basic routing.

**File: routing\_example.js**

1. var express = require('express');
2. var app = express();
3. app.get('/', function (req, res) {
4. console.log("Got a GET request for the homepage");
5. res.send('Welcome to JavaTpoint!');
6. })
7. app.post('/', function (req, res) {
8. console.log("Got a POST request for the homepage");
9. res.send('I am Impossible! ');
10. })
11. app.delete('/del\_student', function (req, res) {
12. console.log("Got a DELETE request for /del\_student");
13. res.send('I am Deleted!');
14. })
15. app.get('/enrolled\_student', function (req, res) {
16. console.log("Got a GET request for /enrolled\_student");
17. res.send('I am an enrolled student.');
18. })
19. // This responds a GET request for abcd, abxcd, ab123cd, and so on
20. app.get('/ab\*cd', function(req, res) {
21. console.log("Got a GET request for /ab\*cd");
22. res.send('Pattern Matched.');
23. })
24. var server = app.listen(8000, function () {
25. var host = server.address().address
26. var port = server.address().port
27. console.log("Example app listening at http://%s:%s", host, port)
28. })



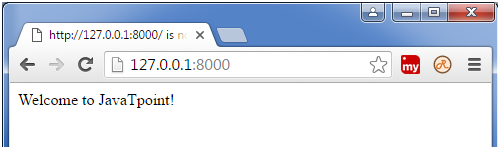
You see that server is listening.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

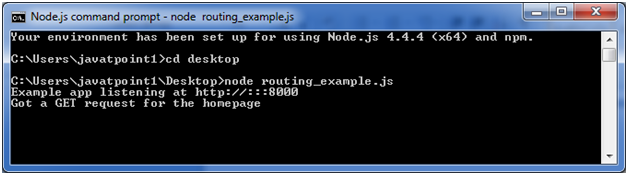
Now, you can see the result generated by server on the local host http://127.0.0.1:8000

**Output:**

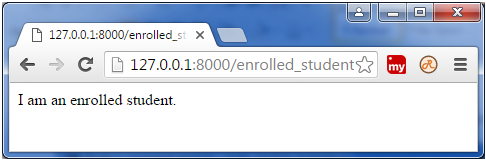
This is the homepage of the example app.



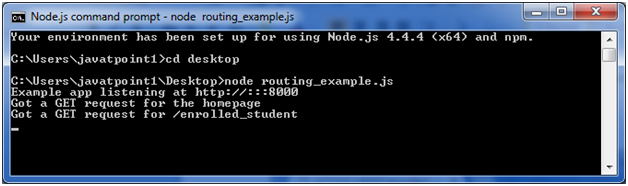
**Note:** The Command Prompt will be updated after one successful response.



You can see the different pages by changing routes. **http://127.0.0.1:8000/enrolled\_student**

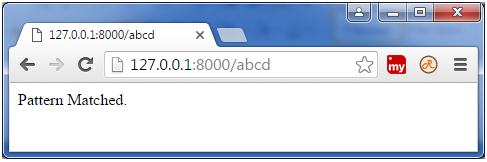


**Updated command prompt:**

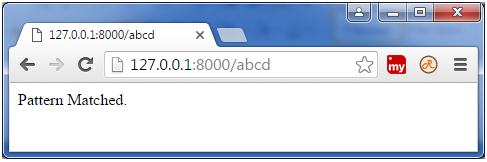


This can read the pattern like abcd, abxcd, ab123cd, and so on.

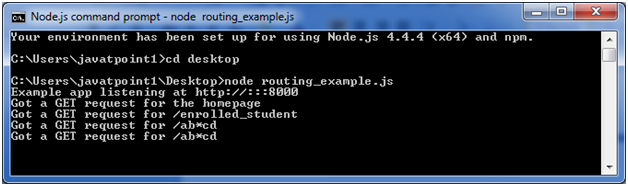
Next route **http://127.0.0.1:8000/abcd**



**Next route http://127.0.0.1:8000/ab12345cd**



**Updated command prompt:**



Express.js Cookies Management

What are cookies

Cookies are small piece of information i.e. sent from a website and stored in user's web browser when user browses that website. Every time the user loads that website back, the browser sends that stored data back to website or server, to recognize user.



Install cookie

You have to acquire cookie abilities in Express.js. So, install cookie-parser middleware through npm by using the following command:



Import cookie-parser into your app.

1. var express = require('express');
2. var cookieParser = require('cookie-parser');
3. var app = express();
4. app.use(cookieParser());

Define a route:

Cookie-parser parses Cookie header and populate req.cookies with an object keyed by the cookie names.

Let's define a new route in your express app like set a new cookie:

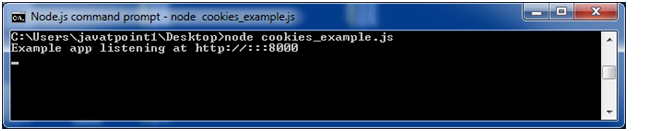
1. app.get('/cookie',function(req, res){
2. res.cookie('cookie\_name' , 'cookie\_value').send('Cookie is set');
3. });
4. app.get('/', function(req, res) {
5. console.log("Cookies :  ", req.cookies);
6. });

Browser sends back that cookie to the server, every time when it requests that website.

Express.js Cookies Example

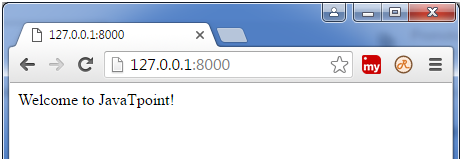
**File:** cookies\_example.js

1. var express = require('express');
2. var cookieParser = require('cookie-parser');
3. var app = express();
4. app.use(cookieParser());
5. app.get('/cookieset',function(req, res){
6. res.cookie('cookie\_name', 'cookie\_value');
7. res.cookie('company', 'javatpoint');
8. res.cookie('name', 'sonoo');
10. res.status(200).send('Cookie is set');
11. });
12. app.get('/cookieget', function(req, res) {
13. res.status(200).send(req.cookies);
14. });
15. app.get('/', function (req, res) {
16. res.status(200).send('Welcome to JavaTpoint!');
17. });
18. var server = app.listen(8000, function () {
19. var host = server.address().address;
20. var port = server.address().port;
21. console.log('Example app listening at http://%s:%s', host, port);
22. });



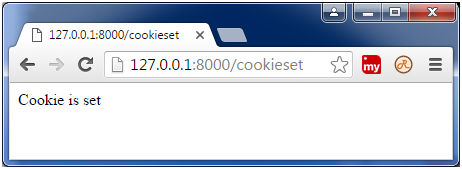
**Output:**

Open the page **http://127.0.0.1:8000/** on your browser:



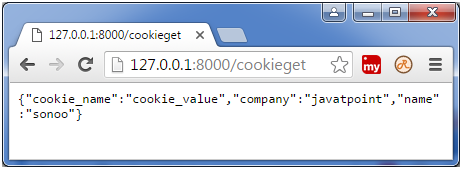
**Set cookie:**

Now open **http://127.0.0.1:8000/cookieset** to set the cookie:



**Get cookie:**

Now open**http://127.0.0.1:8000/cookieget** to get the cookie:



# Express.js File Upload

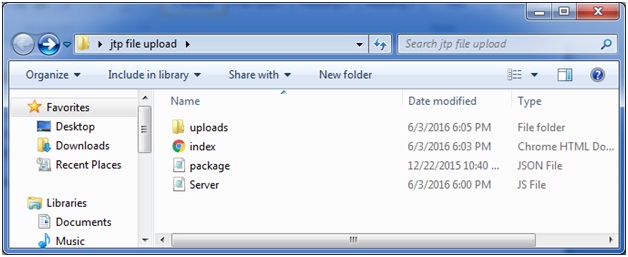
In Express.js, file upload is slightly difficult because of its asynchronous nature and networking approach.

It can be done by using middleware to handle multipart/form data. There are many middleware that can be used like multer, connect, body-parser etc.

Let's take an example to demonstrate file upload in Node.js. Here, we are using the middleware 'multer'.

Create a folder "jtp file upload" having the following files:

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)



**uploads:** It is an empty folder i.e. created to store the uploaded images.

**package:** It is JSON file, having the following data:

**File: package.json**

1. {
2. "name": "file\_upload",
3. "version": "0.0.1",
4. "dependencies": {
5. "express": "4.13.3",
6. "multer": "1.1.0"
7. },
8. "devDependencies": {
9. "should": "~7.1.0",
10. "mocha": "~2.3.3",
11. "supertest": "~1.1.0"
12. }
13. }

**File: index.html**

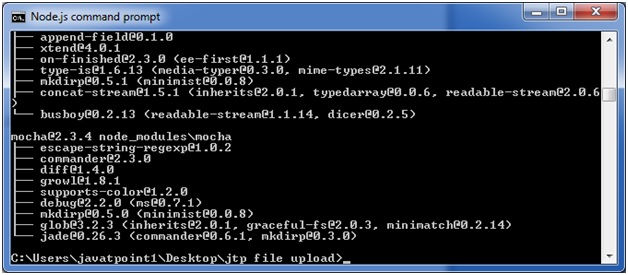
1. **<html>**
2. **<head>**
3. **<title>**File upload in Node.js by Javatpoint**</title>**
4. **<script** src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"**></script>**
5. **<script** src="http://cdnjs.cloudflare.com/ajax/libs/jquery.form/3.51/jquery.form.min.js"**></script>**
6. **<script>**
7. $(document).ready(function() {
8. $('#uploadForm').submit(function() {
9. $("#status").empty().text("File is uploading...");
11. $(this).ajaxSubmit({
13. error: function(xhr) {
14. status('Error: ' + xhr.status);
15. },
17. success: function(response) {
18. console.log(response)
19. $("#status").empty().text(response);
20. }
21. });
23. return false;
24. });
25. });
26. **</script>**
27. **</head>**
28. **<body>**
29. **<h1>**Express.js File Upload: by Javatpoint**</h1>**
30. **<form** id="uploadForm" enctype="multipart/form-data" action="/uploadjavatpoint" method="post"**>**
31. **<input** type="file" name="myfile" **/><br/><br/>**
32. **<input** type="submit" value="Upload Image" name="submit"**><br/><br/>**
33. **<span** id="status"**></span>**
34. **</form>**
35. **</body>**
36. **</html>**

**File: server.js**

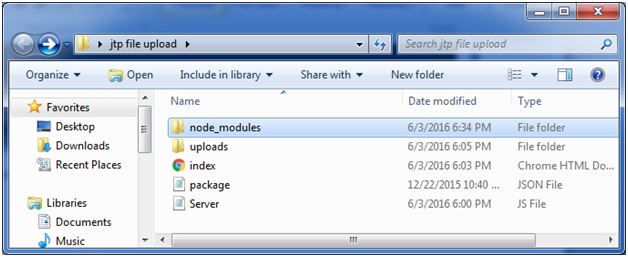
2. var express =   require("express");
3. var multer  =   require('multer');
4. var app =   express();
5. var storage =   multer.diskStorage({
6. destination: function (req, file, callback) {
7. callback(**null**, './uploads');
8. },
9. filename: function (req, file, callback) {
10. callback(**null**, file.originalname);
11. }
12. });
13. var upload = multer({ storage : storage}).single('myfile');
15. app.get('/',function(req,res){
16. res.sendFile(\_\_dirname + "/index.html");
17. });
19. app.post('/uploadjavatpoint',function(req,res){
20. upload(req,res,function(err) {
21. **if**(err) {
22. **return** res.end("Error uploading file.");
23. }
24. res.end("File is uploaded successfully!");
25. });
26. });
28. app.listen(2000,function(){
29. console.log("Server is running on port 2000");
30. });

To install the package.json, **execute the following code**:

1. npm install

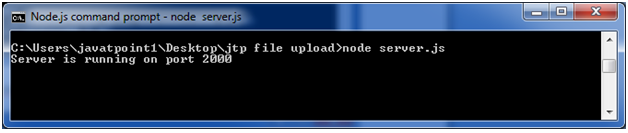


It will create a new folder "node\_modules" inside the "jtp file upload" folder.



Dependencies are installed. Now, run the server:

1. node server.js



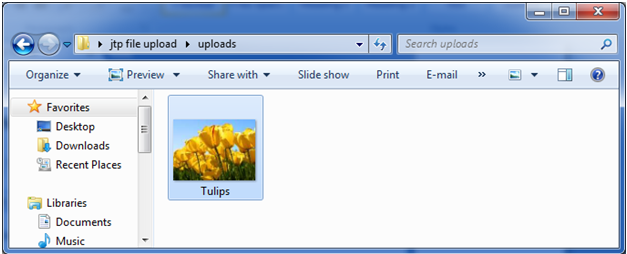
Open the local page http://127.0.0.1:2000/ to upload the images.



Select an image to upload and click on "Upload Image" button.



Here, you see that file is uploaded successfully. You can see the uploaded file in the "Uploads" folder.



### **Download Node.js Express File Upload Example**

Express.js Middleware

Express.js Middleware are different types of functions that are invoked by the Express.js routing layer before the final request handler. As the name specified, Middleware appears in the middle between an initial request and final intended route. In stack, middleware functions are always invoked in the order in which they are added.

Middleware is commonly used to perform tasks like body parsing for URL-encoded or JSON requests, cookie parsing for basic cookie handling, or even building JavaScript modules on the fly.

What is a Middleware function

Middleware functions are the functions that access to the request and response object (req, res) in request-response cycle.

A middleware function can perform the following tasks:

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

* It can execute any code.
* It can make changes to the request and the response objects.
* It can end the request-response cycle.
* It can call the next middleware function in the stack.

Express.js Middleware

Following is a list of possibly used middleware in Express.js app:

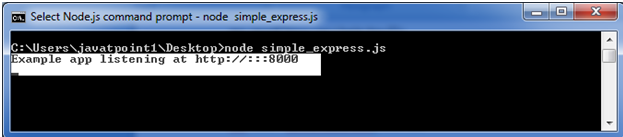
* Application-level middleware
* Router-level middleware
* Error-handling middleware
* Built-in middleware
* Third-party middleware

Let's take an example to understand what middleware is and how it works.

Let's take the most basic Express.js app:

**File: simple\_express.js**

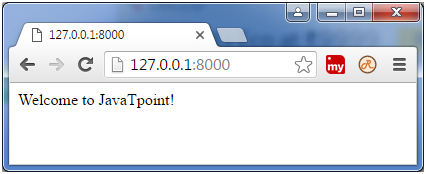
1. var express = require('express');
2. var app = express();
4. app.get('/', function(req, res) {
5. res.send('Welcome to JavaTpoint!');
6. });
7. app.get('/help', function(req, res) {
8. res.send('How can I help You?');
9. });
10. var server = app.listen(8000, function () {
11. var host = server.address().address
12. var port = server.address().port
13. console.log("Example app listening at http://%s:%s", host, port)
14. })



You see that server is listening.

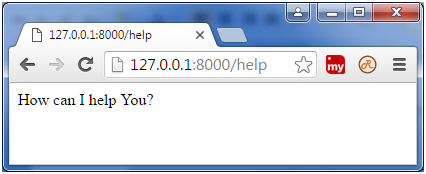
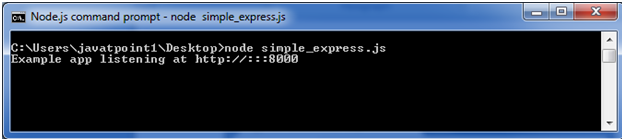
Now, you can see the result generated by server on the local host **http://127.0.0.1:8000**

**Output:**



Let's see the next page: **http://127.0.0.1:8000/help**

**Output:**

**Note:** You see that the command prompt is not changed. Means, it is not showing any record of the GET request although a GET request is processed in the **http://127.0.0.1:8000/help page**.

Use of Express.js Middleware

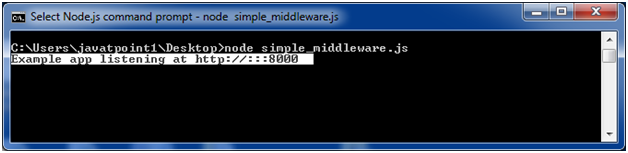
If you want to record every time you a get a request then you can use a middleware.

**See this example:**

**File: simple\_middleware.js**

Learn more

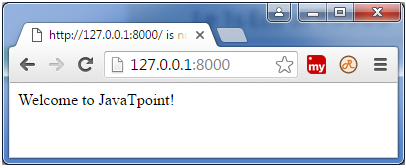
1. var express = require('express');
2. var app = express();
3. app.use(function(req, res, next) {
4. console.log('%s %s', req.method, req.url);
5. next();
6. });
7. app.get('/', function(req, res, next) {
8. res.send('Welcome to JavaTpoint!');
9. });
10. app.get('/help', function(req, res, next) {
11. res.send('How can I help you?');
12. });
13. var server = app.listen(8000, function () {
14. var host = server.address().address
15. var port = server.address().port
16. console.log("Example app listening at http://%s:%s", host, port)
17. })



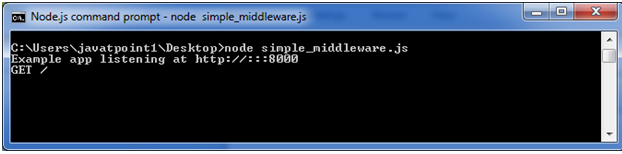
You see that server is listening.

Now, you can see the result generated by server on the local host **http://127.0.0.1:8000**

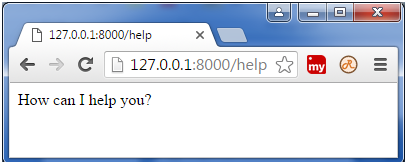
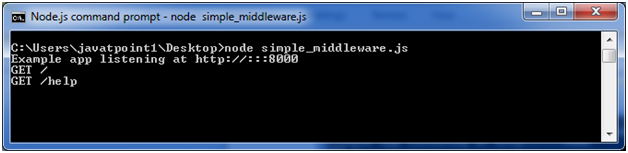
**Output:**



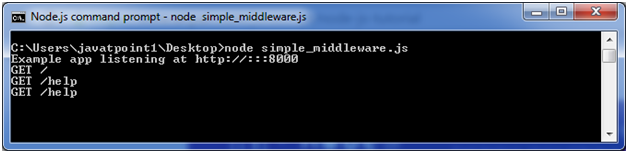
You can see that output is same but command prompt is displaying a GET result.



Go to **http://127.0.0.1:8000/help**

As many times as you reload the page, the command prompt will be updated.



**Note:** In the above example next() middleware is used.

**Middleware example explanation**

* In the above middleware example a new function is used to invoke with every request via**app.use()**.
* Middleware is a function, just like route handlers and invoked also in the similar manner.
* You can add more middlewares above or below using the same API.

Express.js Scaffolding

What is scaffolding

Scaffolding is a technique that is supported by some MVC frameworks.

It is mainly supported by the following frameworks:

Ruby on Rails,OutSystems Platform, Express Framework, Play framework, Django, MonoRail, Brail, Symfony, Laravel, CodeIgniter, Yii, CakePHP, Phalcon PHP, Model-Glue, PRADO, Grails, Catalyst, Seam Framework, Spring Roo, ASP.NET etc.

Scaffolding facilitates the programmers to specify how the application data may be used. This specification is used by the frameworks with predefined code templates, to generate the final code that the application can use for CRUD operations (create, read, update and delete database entries).

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

Express.js Scaffold

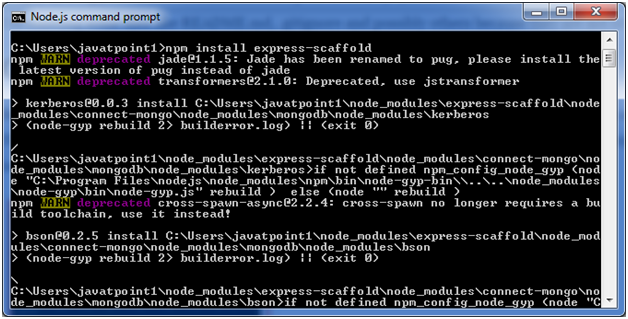
An Express.js scaffold supports candy and more web projects based on Node.js.

Install scaffold

Execute the following command to install scaffold.

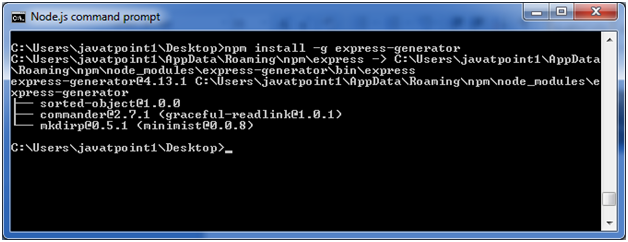
1. npm install express-scaffold

It will take a few seconds and the screen will look like this:



After this step, execute the following command to install express generator:

1. npm install -g express-generator



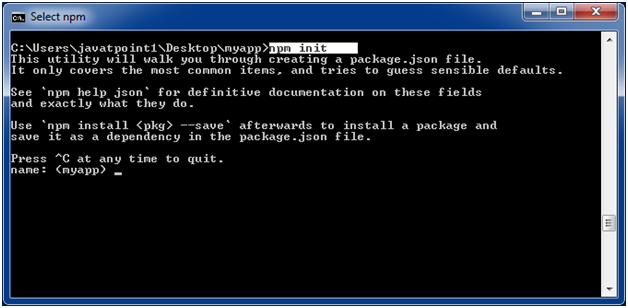
Now, you can use express to scaffold a web-app.

**Let's take an example:**

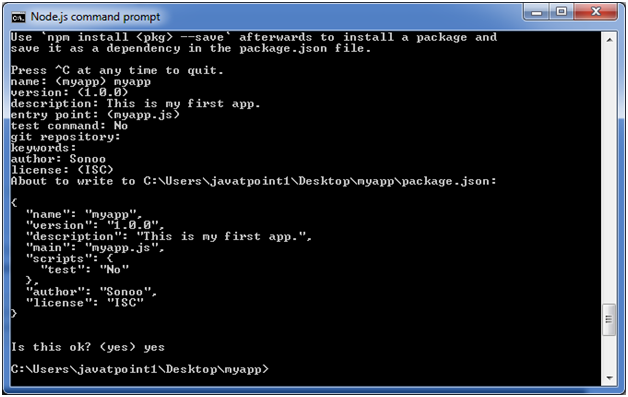
First create a directory named myapp. Create a file named app.js in the myapp directory having the following code:

1. var express = require('express');
2. var app = express();
3. app.get('/', function (req, res) {
4. res.send('Welcome to JavaTpoint!');
5. });
6. app.listen(8000, function () {
7. console.log('Example app listening on port 8000!');
8. });

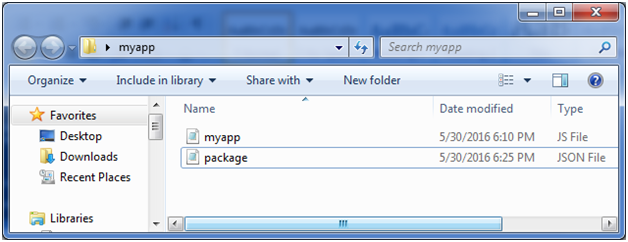
Open Node.js command prompt, go to myapp and run npm init command (In my case, I have created myapp folder on desktop)



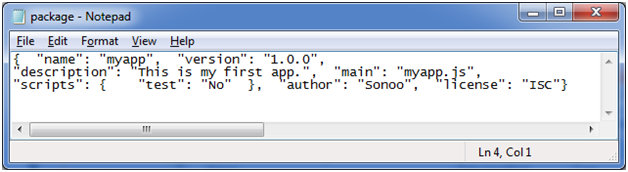
Fill the entries and press enter.



It will create a package.json file in myapp folder and the data is shown in JSON format.



**Output:**



Express.js Template Engine

What is a template engine

A template engine facilitates you to use static template files in your applications. At runtime, it replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. So this approach is preferred to design HTML pages easily.

Following is a list of some popular template engines that work with Express.js:

* Pug (formerly known as jade)
* mustache
* dust
* atpl
* eco
* ect
* ejs
* haml
* haml-coffee
* handlebars
* hogan
* jazz
* jqtpl
* JUST
* liquor
* QEJS
* swig
* templayed
* toffee
* underscore
* walrus
* whiskers

In the above template engines, pug (formerly known as jade) and mustache seems to be most popular choice. Pug is similar to Haml which uses whitespace. According to the template-benchmark, pug is 2x slower than Handlebars, EJS, Underscore. ECT seems to be the fastest. Many programmers like mustache template engine mostly because it is one of the simplest and versatile template engines.

Using template engines with Express

Template engine makes you able to use static template files in your application. To render template files you have to set the following application setting properties:

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

* **Views:** It specifies a directory where the template files are located.

**For example:** app.set('views', './views').

* **view engine:** It specifies the template engine that you use. For example, to use the Pug template engine: app.set('view engine', 'pug').

Let's take a template engine pug (formerly known as jade).

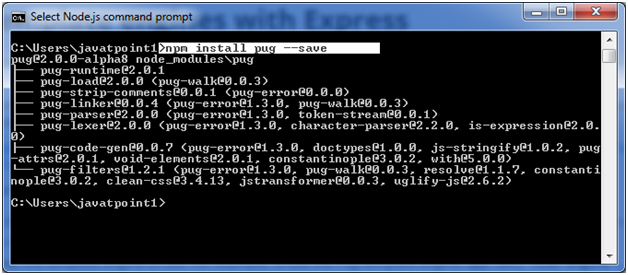
Pug Template Engine

Let's learn how to use pug template engine in Node.js application using Express.js. Pug is a template engine for Node.js. Pug uses whitespaces and indentation as the part of the syntax. Its syntax is aesy to learn.

Install pug

Execute the following command to install pug template engine:

1. npm install pug --save



Pug template must be written inside .pug file and all .pug files must be put inside views folder in the root folder of Node.js application.

**Note:** By default Express.js searches all the views in the views folder under the root folder. you can also set to another folder using views property in express. For example: app.set('views','MyViews').

The pug template engine takes the input in a simple way and produces the output in HTML. See how it renders HTML:

**Simple input:**

1. doctype html
2. html
3. head
4. title A simple pug example
5. body
6. h1 This page is produced by pug template engine
7. p some paragraph here..

**Output produced by pug template:**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<title>**A simple pug example**</title>**
5. **</head>**
6. **<body>**
7. **<h1>**This page is produced by pug template engine**</h1>**
8. **<p>**some paragraph here..**</p>**
9. **</body>**
10. **</html>**

Express.js can be used with any template engine. Let's take an example to deploy how pug template creates HTML page dynamically.

**See this example:**

Create a file named **index.pug** file inside views folder and write the following pug template in it:

1. doctype html
2. html
3. head
4. title A simple pug example
5. body
6. h1 This page is produced by pug template engine
7. p some paragraph here..

**File: server.js**

1. var express = require('express');
2. var app = express();
3. //set view engine
4. app.set("view engine","pug")
5. app.get('/', function (req, res) {
6. res.render('view.pug', index);
7. res.render('index');
8. });
9. var server = app.listen(5000, function () {
10. console.log('Node server is running..');
11. });

# Node.js Interview Questions

A list of top frequently asked **Node.js interview questions** and answers are given below.

### **1) What is Node.js?**

Node.js is Server-side scripting which is used to build scalable programs. It is a web application framework built on Google Chrome's JavaScript Engine. It runs within the Node.js runtime on Mac OS, Windows, and Linux with no changes. This runtime facilitates you to execute a JavaScript code on any machine outside a browser.

### **2) Is Node.js free to use?**

Yes. It is released under MIT license and is free to use.

### **3) Is Node a single threaded application?**

Yes. Node is a single-threaded application with event looping.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

### **4) What is the purpose of Node.js?**

These are the following purposes of Node.js:

* Real-time web applications
* Network applications
* Distributed systems
* General purpose applications

### **5) What are the advantages of Node.js?**

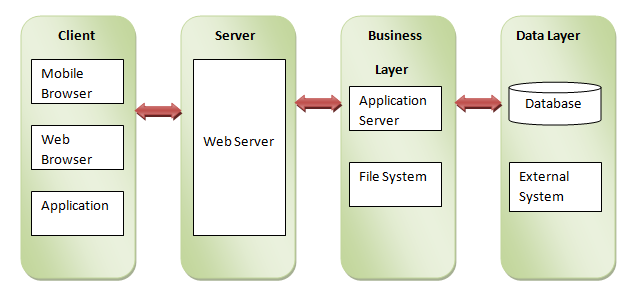
Following are the main advantages of Node.js:

* Node.js is asynchronous and event-driven. All API?s of Node.js library are non-blocking, and its server doesn't wait for an API to return data. It moves to the next API after calling it, and a notification mechanism of Events of Node.js responds to the server from the previous API call.
* Node.js is very fast because it builds on Google Chrome?s V8 JavaScript engine. Its library is very fast in code execution.
* Node.js is single threaded but highly scalable.
* Node.js provides a facility of no buffering. Its application never buffers any data. It outputs the data in chunks.

### **6) Explain Node.js web application architecture?**

A web application distinguishes into 4 layers:

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



### **7) What do you understand by the term I/O?**

The term I/O stands for input and output. It is used to access anything outside of your application. The I/O describes any program, operation, or device that transfers data to or from a medium or another medium. This medium can be a physical device, network, or files within a system.

I/O is loaded into the machine memory to run the program once the application starts.

### **8) How many types of API functions are available in Node.js?**

There are two types of API functions in Node.js:

* Asynchronous, Non-blocking functions
* Synchronous, Blocking functions

### **9) What do you understand by the first class function in JavaScript?**

When functions are treated like any other variable, then those functions are called first-class functions. Apart from JavaScript, many other programming languages, such as Scala, Haskell, etc. follow this pattern. The first class functions can be passed as a param to another function (callback), or a function can return another function (higher-order function). Some examples of higher-order functions that are popularly used are map() and filter().

### **10) What is the difference between JavaScript and Node.js?**

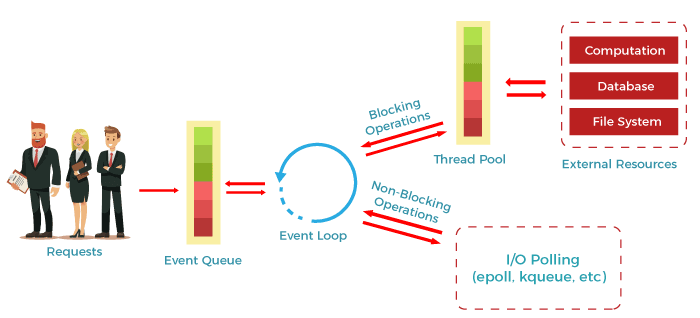
**Difference between JavaScript and Node.js**

The following table specifies the crucial differences between JavaScript and Node.js:

|  |  |  |
| --- | --- | --- |
| **Comparison features** | **JavaScript** | **Node.js** |
| Type | JavaScript is a programming language. More precisely, you can say that it is a scripting language used for writing scripts on the website. | Node.js is an interpreter and run time environment for JavaScript. |
| Utility | JavaScript is used for any client-side activity for a web application. | Node.js is used for accessing or performing any non-blocking operation of any operating system. |
| Running Engine | The running engine for JavaScript is Spider monkey (Firefox), JavaScript Core (Safari), V8 (Google Chrome), etc. | The running engine for Node.js is V8 (Google Chrome). |
| Browser compatibility | JavaScript can only be run in browsers. | The Node.js code can be run outside the browser. |
| Platform dependency | JavaScript is basically used on the client-side and is used in frontend development. | Node.js is mostly used on the server-side and is used in server-side development. |
| HTML compatibility | JavaScript is capable enough to add HTML and play with the DOM. | Node.js is not compatible enough to add HTML tags. |
| Examples | Some examples of the JavaScript frameworks are RamdaJS, TypedJS, etc. | Some examples of the Node.js modules are Lodash, express, etc. We have to import these modules from npm. |
| Written in | JavaScript is the upgraded version of ECMA script that uses Chrome's V8 engine and is written in C++. | Node.js is written in C, C++, and Javascript. |

### **11) Explain the working of Node.js?**

The workflow of a Node.js web server typically looks like the following diagram. Let us see the flow of operations in detail:



* According to the above diagram, the clients send requests to the webserver to interact with the web application. These requests can be non-blocking or blocking and used for querying the data, deleting data, or updating the data.
* js receives the incoming requests and adds those to the Event Queue.
* After this step, the requests are passed one by one through the Event Loop. It checks if the requests are simple enough not to require any external resources.
* The event loop then processes the simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients.
* A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.
* Once the task is completed, the response is sent to the Event Loop that sends that response back to the client.

### **12) How can you manage the packages in your Node.js project?**

We can manage the packages in our Node.js project by using several package installers and their configuration file accordingly. Most of them use npm or yarn. The npm and yarn both provide almost all libraries of JavaScript with extended features of controlling environment-specific configurations. We can use package.json and package-lock.json to maintain versions of libs being installed in a project. So, there is no issue in porting that app to a different environment.

### **13) Why is Node.js Single-threaded?**

Node.js is a single-threaded application with event looping for async processing. The biggest advantage of doing async processing on a single thread under typical web loads is that you can achieve more performance and scalability than the typical thread-based implementation.

### **14) What do you understand by callback hell in Node.js?**

Callback hell is a phenomenon that creates a lot of problems for a JavaScript developer when he tries to execute multiple asynchronous operations one after the other. A function is called an asynchronous function when some external activity must complete before processing a result. It is called asynchronous because there is an unpredictable amount of time before a result becomes available. These functions require a callback function to handle errors and process the result.

**Example:**

1. getData(function(a){
2. getMoreData(a, function(b){
3. getMoreData(b, function(c){
4. getMoreData(c, function(d){
5. getMoreData(d, function(e){
6. ...
7. });
8. });
9. });
10. });
11. });

### **15) How is Node.js better than other most popular frameworks?**

Based on the following criteria, we can say that Node.js is better than other most popular frameworks:

* js makes development simple because of its non-blocking I/O and even-based model. This simplicity results in short response time and concurrent processing, unlike other frameworks where developers use thread management.
* js runs on a chrome V8 engine which is written in C++. It enhances its performance highly with constant improvement.
* With Node.js, we will use JavaScript in both the frontend and backend development that will be much faster.
* js provides ample libraries so that we don't need to reinvent the wheel.

### **16) In which types of applications is Node.js most frequently used?**

Node.js is most frequently and widely used in the following applications:

* Internet of Things
* Real-time collaboration tools
* Real-time chats
* Complex SPAs (Single-Page Applications)
* Streaming applications
* Microservices architecture etc.

### **17) What are some commonly used timing features of Node.js?**

Following is a list of some commonly used timing features of Node.js:

* **setTimeout/clearTimeout:** This timing feature of Node.js is used to implement delays in the code execution.
* **setInterval/clearInterval:** The setInterval or clearInterval timing feature is used to run a code block multiple times in the application.
* **setImmediate/clearImmediate:** This timing feature of Node.js is used to set the execution of the code at the end of the event loop cycle.
* **nextTick:** This timing feature sets the execution of code at the beginning of the next event loop cycle.

### **18) What do you understand by the term fork in Node.js?**

Generally, a fork is used to spawn child processes. In Node.js, it is used to create a new instance of the V8 engine to run multiple workers to execute the code.

### **19) Which is the best tool we can use to assure consistent code style in Node.js?**

ESLint tool is one of the best tools we can use with any IDE to ensure a consistent coding style. It also helps in maintaining the codebase.

### **20) What is the main difference between front-end and back-end development?**

The following table specifies the key differences between a front-end and back-end development:

|  |  |
| --- | --- |
| **Front-end Development** | **Back-end Development** |
| The front-end development in an application refers to the client-side of an application. | The back-end development in an application refers to the server-side of an application. |
| As the name specifies, the front-end development is the part of a web application where users can see and interact. | As the name specifies, the back-end development consists of everything that happens behind the scenes and users cannot see and interact with. |
| The front-end development includes everything that attributes to the visual aspects of a web application. | The back-end development generally includes a web server that communicates with the database to serve the users' requests. |
| HTML, CSS, Bootstrap, jQuery, JavaScript, AngularJS, and React.js are essential front-end development technologies. | Java, PHP, Python, C++, Node.js, etc., are the technologies required for back-end development. |
| Examples of some front-end frameworks are AngularJS, React.js, jQuery, Sass, etc. | Examples of some back-end frameworks are Express, Django, Rails, Laravel, Spring, etc. |

### **21) Give an example to demonstrate how can we use async await in Node.js?**

See the following example of using async-await pattern:

1. function wait (timeout) {
2. **return** **new** Promise((resolve) => {
3. setTimeout(() => {
4. resolve()
5. }, timeout);
6. });
7. }
8. async function requestWithRetry (url) {
9. **const** MAX\_RETRIES = 10;
10. **for** (let i = 0; i <= MAX\_RETRIES; i++) {
11. **try** {
12. **return** await request(url);
13. } **catch** (err) {
14. **const** timeout = Math.pow(2, i);
15. console.log('Waiting', timeout, 'ms');
16. await wait(timeout);
17. console.log('Retrying', err.message, i);
18. }
19. }
20. }

### **22) What are the modules in Node.js? Which are the different modules used in Node.js?**

In Node.js applications, modules are like JavaScript libraries and include a set of functions. To include a module in a Node.js application, we must use the require() function with the parentheses containing the module's name.

Node.js has several modules which are used to provide the basic functionality needed for a web application. Following is a list of some of them:

|  |  |
| --- | --- |
| **Core Modules** | **Description** |
| HTTP: | The HTTP module includes classes, methods, and events to create a Node.js HTTP server. |
| util: | The util module includes utility functions required in the application and is very useful for developers. |
| url: | The url module is used to include the methods for URL parsing. |
| fs: | The fs module includes events, classes, and methods to handle the file I/O operations. |
| stream: | The stream module is used to include the methods to handle streaming data. |
| query string: | The query string module is used to include the methods to work with a query string. |
| zlib: | The zlib module is used to include the methods to compress or decompress the files used in an application. |

### **23) What are buffers in Node.js?**

In general, a buffer is a temporary memory mainly used by the stream to hold on to some data until it is consumed. Buffers are used to represent a fixed-size chunk of memory allocated outside of the V8 JavaScript engine. It can't be resized. It is like an array of integers, which each represents a byte of data. It is implemented by the Node. js Buffer class. Buffers also support legacy encodings like ASCII, utf-8, etc.

### **24) What is error-first callback?**

Error-first callbacks are used to pass errors and data. If something goes wrong, the programmer has to check the first argument because it is always an error argument. Additional arguments are used to pass data.

1. fs.readFile(filePath, function(err, data) {
2. if (err) {
3. //handle the error
4. }
5. // use the data object
6. });

### **25) What is an asynchronous API?**

All the API's of Node.js library are asynchronous means non-blocking. A Node.js based server never waits for an API to return data. The Node.js server moves to the next API after calling it, and a notification mechanism of Events of Node.js responds to the server for the previous API call.

### **26) How can you avoid callbacks?**

To avoid callbacks, you can use any one of the following options:

* You can use **modularization**. It breaks callbacks into independent functions.
* You can use **promises**.
* You can use **yield** with Generators and Promises.

### **27) Does Node.js provide Debugger?**

Yes, Node.js provides a simple TCP based protocol and built-in debugging client. For debugging your JavaScript file, you can use debug argument followed by the js file name you want to debug.

#### **Syntax:**

1. node debug [script.js | -e "script" | **<host>**:**<port>**]

### **28) What is a control flow function?**

Control flow function is a generic piece of code that runs in between several asynchronous function calls.

### **29) How "Control Flow" controls the functions calls?**

The control flow does the following job:

* Control the order of execution
* Collect data
* Limit concurrency
* Call the next step in a program

### **30) Is it possible to access DOM in Node?**

No, it is not possible to access DOM in Node.

### **31) What types of tasks can be done asynchronously using the event loop?**

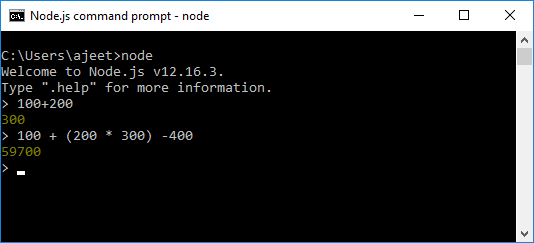
* I/O operations
* Heavy computation
* Anything requiring blocking

### **32) What is REPL in Node.js?**

REPL stands for Read Eval Print Loop. It specifies a computer environment like a window console or Unix/Linux shell where you can enter a command, and the computer responds with an output. It is very useful in writing and debugging the codes. REPL environment incorporates Node.js.

**See the Example:**

1. $ node
2. > 100 + 200
3. 300
4. > 100 + ( 200 \* 300 ) - 400
5. 59700
6. >



### **33) Explain the tasks of terms used in Node REPL.**

Following are the terms used in REPL with their defined tasks:

**Read:** It reads user's input; parse the input into JavaScript data-structure and stores in memory.

**Eval:** It takes and evaluates the data structure.

**Print:** It is used to print the result.

**Loop:** It loops the above command until user press ctrl-c twice to terminate.

### **34) Is it possible to evaluate simple expressions using Node REPL?**

Yes. You can evaluate simple expressions using Node REPL.

### **35) What is the use of the underscore variable in REPL?**

In REPL, the underscore variable is used to get the last result.

1. C:\Nodejs\_WorkSpace**>**node
2. **>** var x = 10
3. undefined
4. **>** var y = 20
5. undefined
6. **>** x + y
7. 30
8. **>** var sum = \_
9. undefined
10. **>** console.log(sum)
11. 30
12. undefined
13. **>**

### **36) Does Node.js supports cryptography?**

Yes, Node.js Crypto module supports cryptography. It provides cryptographic functionality that includes a set of wrappers for open SSL's hash HMAC, cipher, decipher, sign and verify functions. For example:

1. const crypto = require('crypto');
2. const secret = 'abcdefg';
3. const hash = crypto.createHmac('sha256', secret)
4. .update('Welcome to JavaTpoint')
5. .digest('hex');
6. console.log(hash);

### **37) What is npm? What is the main functionality of npm?**

npm stands for Node Package Manager. Following are the two main functionalities of npm:

* Online repositories for node.js packages/modules which are searchable on search.nodejs.org
* Command line utility to install packages, do version management and dependency management of Node.js packages.

### **38) What tools can be used to assure a consistent style in Node.js?**

Following is a list of tools that can be used in developing code in teams, to enforce a given style guide and to catch common errors using static analysis.

* JSLint
* JSHint
* ESLint
* JSCS

### **39) What is the difference between operational and programmer errors?**

Operational errors are not bugs, but create problems with the system like request timeout or hardware failure. On the other hand, programmer errors are actual bugs.

### **40) What is the difference between the global installation of dependencies and local installation of dependencies?**

* Global installation of dependencies is stored in /npm directory. While local installation of dependencies stores in the local mode. Here local mode refers to the package installation in node\_modules directory lying in the folder where Node application is present.
* Globally deployed packages cannot be imported using require() in Node application directly. On the other hand, locally deployed packages are accessible via require().
* To install a Node project globally -g flag is used.
  1. C:\Nodejs\_WorkSpace**>**npm install express ?g
* To install a Node project locally, the syntax is:
  1. C:\Nodejs\_WorkSpace**>**npm install express

### **41) What is the use of a buffer class in Node.js?**

The Node.js provides Buffer class to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap. It is a global class and can be accessed in an application without importing a buffer module. Buffer class is used because pure JavaScript is not compatible with binary data. So, when dealing with TCP streams or the file system, it's necessary to handle octet streams.

### **42) What is the role of assert in Node.js?**

The Node.js Assert is a way to write tests. It provides no feedback when running your test unless one fails. The assert module provides a simple set of assertion tests that can be used to test invariants. The module is intended for internal use by Node.js, but can be used in application code via require ('assert'). For example:

1. var assert = require('assert');
2. function add (a, b) {
3. return a + b;
4. }
5. var expected = add(1,2);
6. assert( expected === 3, 'one plus two is three');

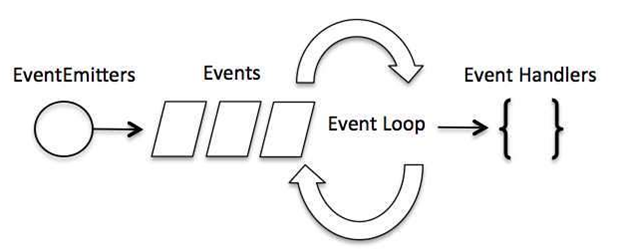
### **43) What are the streams in Node.js?**

The Streams are the objects that facilitate you to read data from a source and write data to a destination. There are four types of streams in Node.js:

* **Readable:** This stream is used for reading operations.
* **Writable:** This stream is used for write operations.
* **Duplex:** This stream can be used for both reading and write operations.
* **Transform:** It is a type of duplex stream where the output computes according to input.

### **44) What is event-driven programming in Node.js?**

In Node.js, event-driven programming means as soon as Node starts its server, it initiates its variables, declares functions and then waits for an event to occur. It is one of the reasons why Node.js is pretty fast compared to other similar technologies.



### **45) What is the difference between events and callbacks in Node.js?**

Although, Events and Callbacks look similar the differences lies in the fact that callback functions are called when an asynchronous function returns its result whereas event handling works on the observer pattern. Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through the events module and EventEmitter class which is used to bind events and event listeners.

### **46) What is the Punycode in Node.js?**

The Punycode is an encoding syntax which is used to convert Unicode (UTF-8) string of characters to ASCII string of characters. It is bundled with Node.js v0.6.2 and later versions. If you want to use it with other Node.js versions, then use npm to install Punycode module first. You have to used require ('Punycode') to access it.

#### **Syntax:**

1. punycode = require('punycode');

### **47) What does Node.js TTY module contains?**

The Node.js TTY module contains tty.ReadStream and tty.WriteStream classes. In most cases, there is no need to use this module directly. You have to used require ('tty') to access this module.

#### **Syntax:**

1. var tty = require('tty');

### **48) What are the key differences between Angular and Node.js?**

Key differences between Angular and Node.js:

|  |  |
| --- | --- |
| **Angular** | **Node.js** |
| Angular is a structural front-end development framework for developing dynamic web apps. | Node.js is a cross-platform, run-time, server-side environment for applications written in JavaScript language. |
| Angular is entirely written in TypeScript language. | Node.js is written in C, C++, and JavaScript languages. |
| Angular is used for building single-page, client-side web applications. | Node.js is used for building fast and scalable, client-side, and server-side networking applications. |
| Angular is easy to use. The developers need to add the Angular file to use it in their applications. | Node.js is slightly complicated to use. Here, the developers need to install Node.js on their computer system. |
| Angular split a web application into MVC components. Here, the models and views are much simpler than what is found in other JavaScript client-side frameworks. | Node.js generates database queries and uses the event-driven nature of JavaScript to support non-blocking operations, making the platform efficient. |
| Angular is based on the model-view-controller design pattern and follows that pattern completely. | Node.js is single-threaded. It means the web requests and processing runs on the same thread. |
| Angular is a Web Framework. | Node.js provides different Web Frameworks like Socket.io, Hapi.js, Meteor.js, Express.js, and Sails.js, etc. |
| Angular is ideal for creating highly active and interactive web apps. | Node.js is the best for developing small-size projects. |
| Angular requires a deep understanding of prototyping, scope, and various other JavaScript aspects. | Node.js facilitates developers to use JavaScript on the client as well as the server-side. So, they can focus on learning one language. |

### **49) What are the main differences between operational and programmer errors?**

The most crucial difference between operational and programmer errors is that the operational errors are not bugs but problems with the system such as to request timeout or hardware failure. On the other hand, the programmer errors are actual bugs in the application.

### **50) What do you understand by an EventEmitter in Node.js?**

In Node.js, an EventEmitter is a class that includes all the objects capable of emitting events. This can be achieved by attaching named events that are emitted by the object using an eventEmitter.on() function. Thus whenever this object throws an event, the attached functions are invoked synchronously.

**Example:**

1. const EventEmitter = require('events');
2. class MyEmitter extends EventEmitter {}
3. const myEmitter = new MyEmitter();
4. myEmitter.on('event', () =**>** {
5. console.log('an event occurred!');
6. });
7. myEmitter.emit('event');

### **51) What is the difference between readFile and createReadStream in Node.js?**

In Node.js, there are two ways to read and execute files: readFile and CreateStream.

* The readFile() process is a fully buffered process that returns the response only when the complete file is pushed into the buffer and is read. This process is called a memory-intensive process, and in the case of large files, the processing can be very slow.
* On the other hand, the createReadStream() is a partially buffered process that treats the entire process as an event series. The entire file is split into chunks and then processed and sent back as a response one by one. After completing this step, they are finally removed from the buffer. Unlike the readFile process, the createReadStream process is effective for the processing of large files.

### **52) What is the concept of Punycode in Node.js?**

In Node.js, the concept of Punycode is used for converting one type of string into another type. Punycode is an encoding syntax used for converting Unicode (UTF-8) string of characters into a basic ASCII string of characters. Now, the hostnames can only understand the ASCII characters so, after the Node.js version 0.6.2 onwards, it was bundled up with the default Node package.

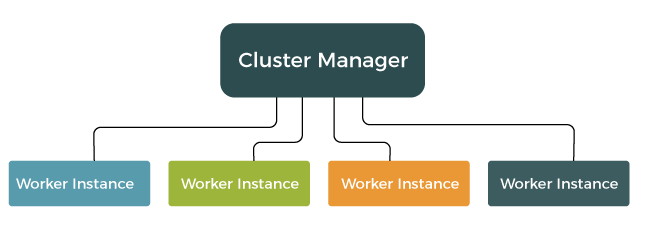
To use it with any previous versions, you have to use the following code:

**Syntax:**

1. punycode = require('punycode');

### **53) How can you enhance the Node.js performance through clustering?**

Just because the Node.js applications run on a single processor, they don't take advantage of a multiple-core system by default. Clustering is used to overcome this issue. The cluster mode is used to start up multiple node.js processes, thereby having multiple instances of the event loop. When we start using clusters in a Node.js app, it creates multiple node.js processes. But there is also a parent process called the cluster manager, which is responsible for monitoring the health of the individual instances of the application.



### **54) What is a thread pool in Node.js? Which library handles it?**

In Node.js, the libuv library is used to handle the Thread pool. The libuv library is a multi-platform C library that supports asynchronous I/O-based operations such as file systems, networking, and concurrency.

