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.

Prerequisite

Before learning Express.js, you must have the basic knowledge of JavaScript and Node.js.

# 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

Installing Express

Use the following command to install express:

**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.

**npm install body-parser --save**

**npm install cookie-parser –save**

**npm install multer --save**

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.

|  |  |  |
| --- | --- | --- |
| **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:**

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:

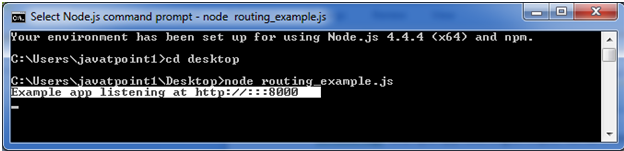
Express.js Routing

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. })

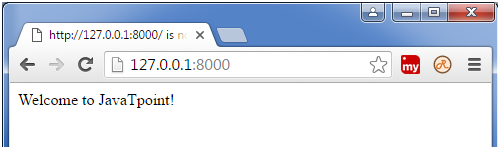


You see that server is listening.

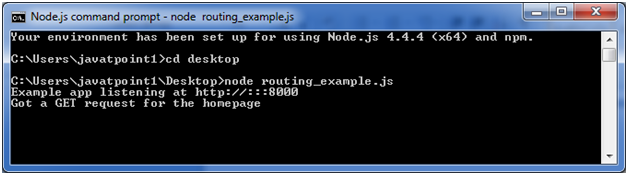
Now, you can see the result generated by server on the local host http://127.0.0.1:8000

**Output:**

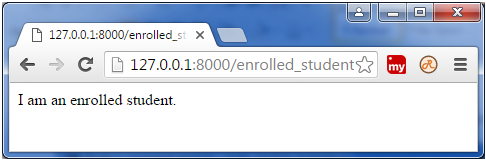
This is the homepage of the example app.



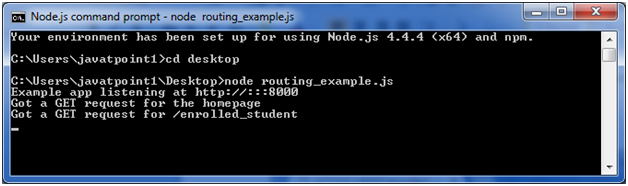
**Note:** The Command Prompt will be updated after one successful response.



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

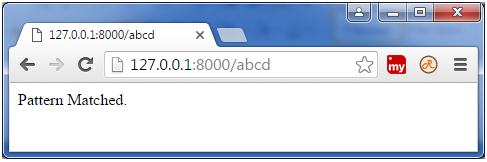


**Updated command prompt:**

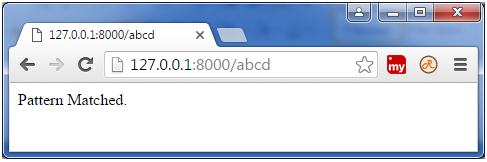


This can read the pattern like abcd, abxcd, ab123cd, and so on.

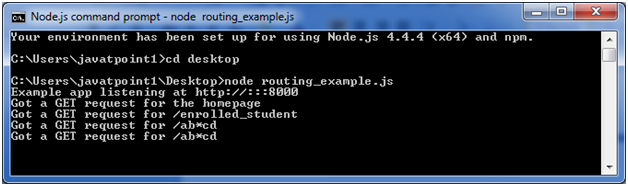
Next route **http://127.0.0.1:8000/abcd**



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



**Updated command prompt:**



Express.js 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:

* **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).

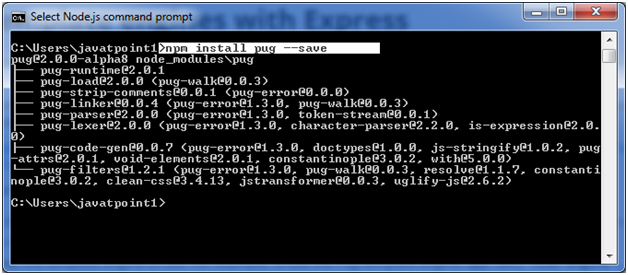
Pug Template Engine

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. });