**https://www.twilio.com/blog/2016/07/how-to-receive-a-post-request-in-node-js.html**

**https://expressjs.com/en/4x/api.html**

**Node JS Modules**

A node.js application consists of following three important parts.

1 import required module : We use require directive to load a Node.js module.

2. create server : A server which will listen to client's request similar to Apache HTTP Server.

3. Read request and return response : Server created in earlier step will read HTTP request made by client which can be a browser to console and return the response.

**Import required modules :** We use require directive to load a Node.js module

**Create Server :** A server which will listen to client's request similar to Apache Http Server.

**Read request and return response :** Server created in earlier step will read HTTP request made by client which can be a browser or console and return the response.

**main.js**

var http = require("http");

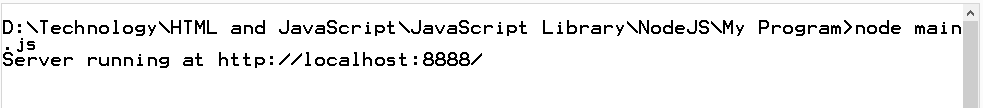
http.createServer(function(req,res){

res.writeHead(200,{"Content-Type":"text/plain"});

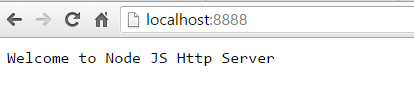
res.end('Welcome to Node JS Http Server');

}).listen(8888);

console.log('Server running at http://localhost:8888/')



After running the program in browser



**Node JS with Express**

Let's install a famous Node.js web framework called express.

Open the Node.js command prompt and execute the following command.

**Web Application :**Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

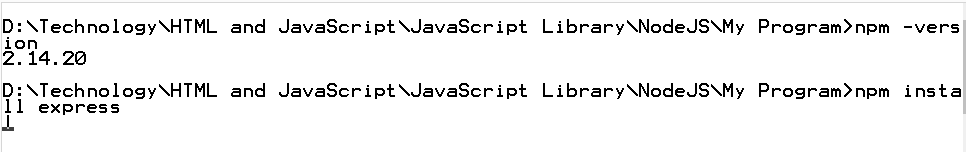
APIs With a myriad of HTTP utility methods and middleware are your disposal, creating a robust API is quick and easy.

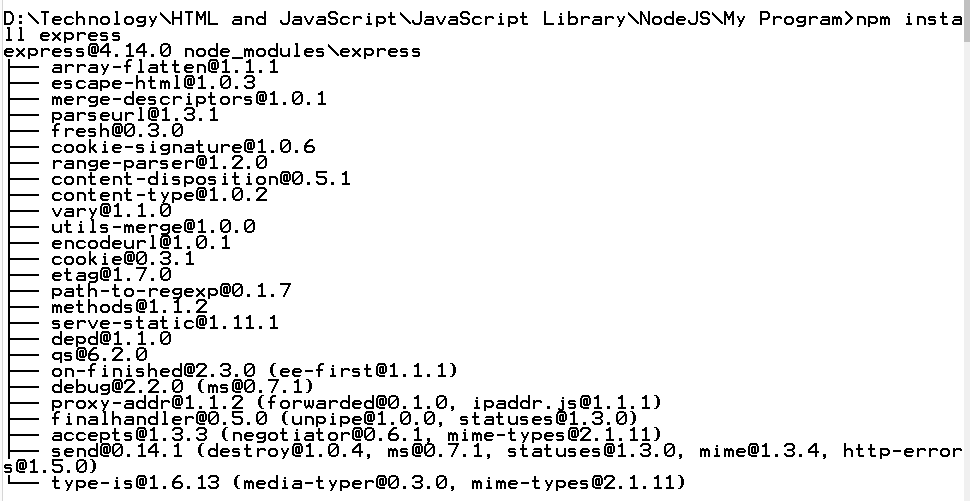
**Performance :** Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

**Installing**

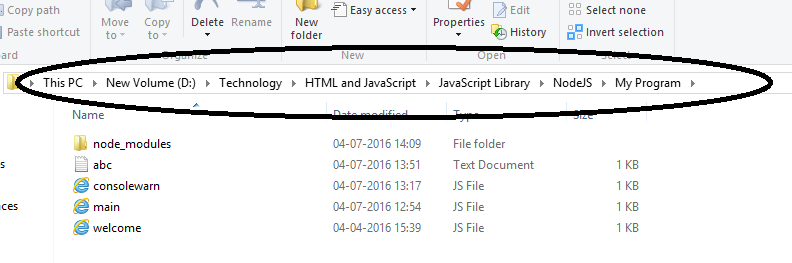
Assuming you have already installed Node.js, create a directory to hold your application, and make that your working directory.

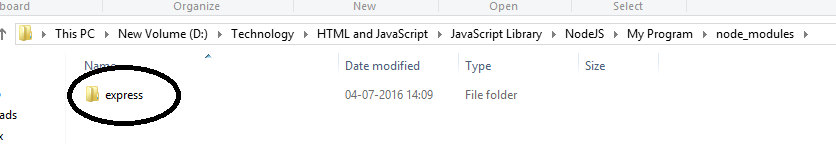
Now install the express using the npm



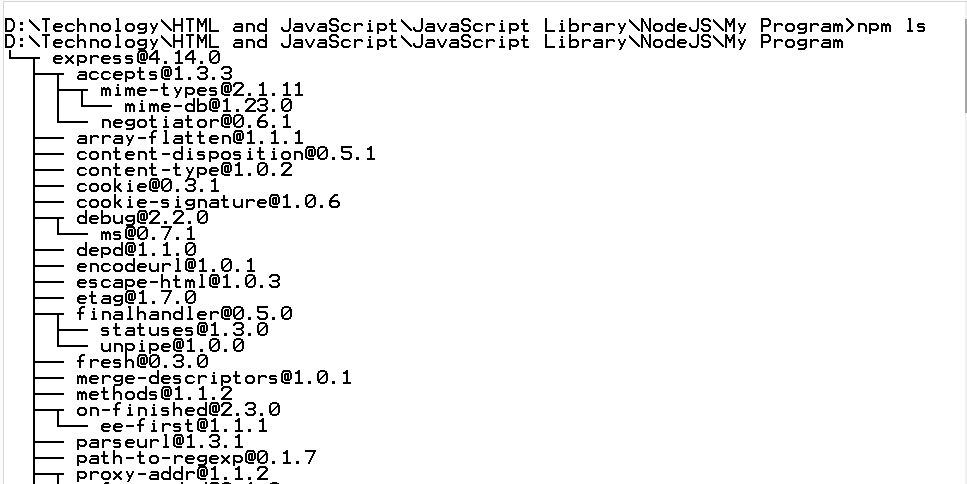


Here the module are installed



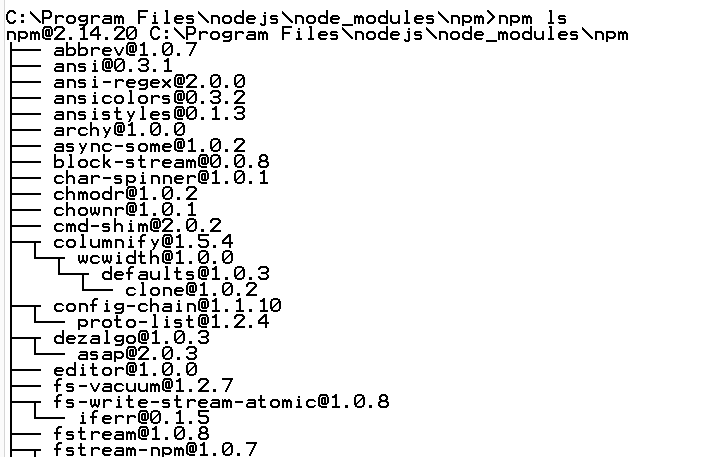


To see the list of module installed



This command are use to see the list of module installed locally

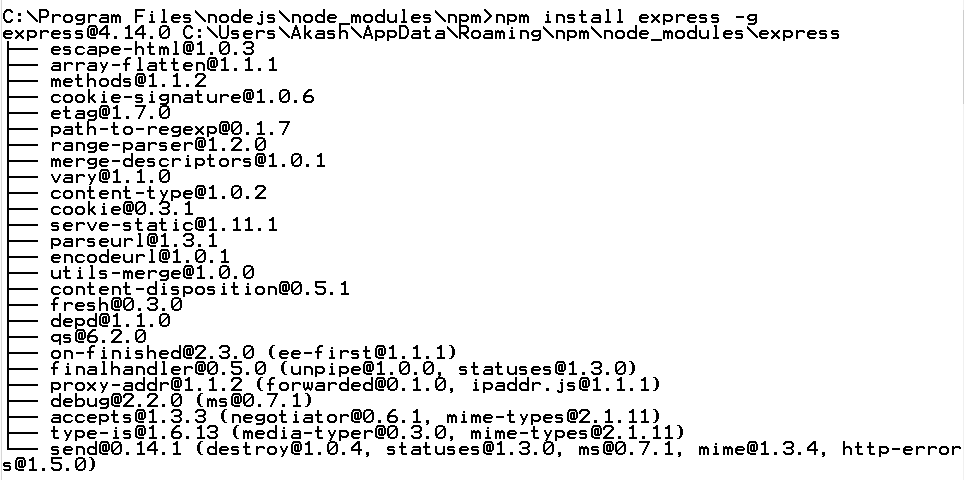
To see the list of module in Node.js software installed



Global installed packages / dependencies are stored in system directory. Let's install express module using global installation. Although it will also produce the same result but modules will be installed globally.

Open Node.js command prompt and execute the following code

**npm install express -g**



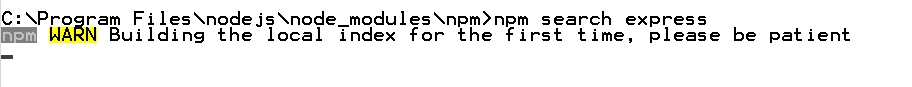
Here first line tells about the module version and its locations where it is getting installed.

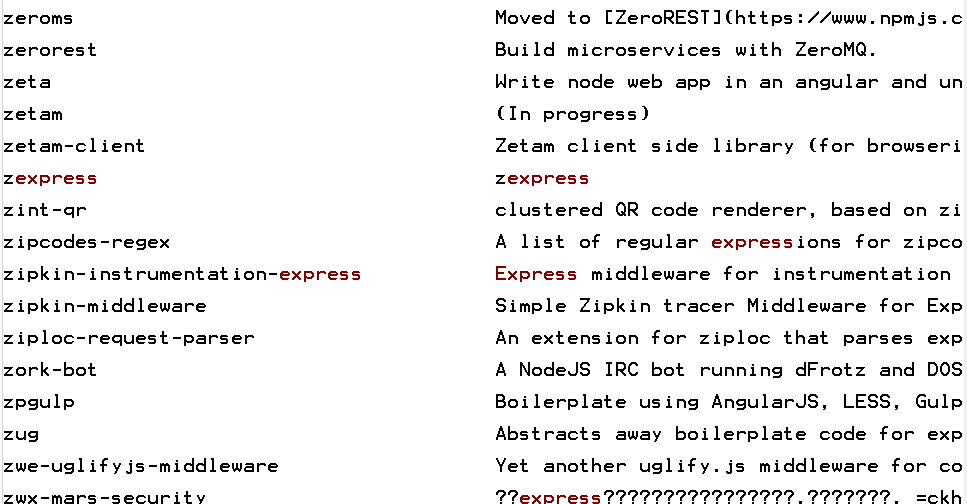
**Uninstalling a module**

npm uninstall express

**Searching a module**

"npm search express"





**npm install -g express**

**Simple example to run the web server on port number 3000.**

**webServer.js**

**const express = require("express");**

**const app = express();**

**const port = 3000;**

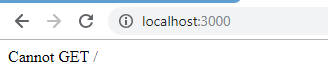
**app.listen(port,()=>console.log(`Server is number on port number ${port}`));**

After running this program



Now if you hit on the browser with the port number

**http://localhost:3000**

****

**Sample program to display the get method invocation.**

**const express = require("express");**

**const app= express();**

**const port = 3000;**

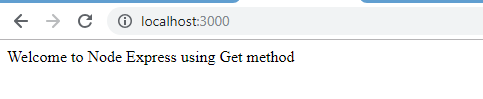
**app.get("/",(req,res)=>**

**res.send("Welcome to Node Express using Get method")**

**);**

**app.listen(port,()=>console.log(`Server running on port number ${port}`));**

After running this program



This app starts a server and listens on a port 3000 for connections, The app responds with msg for requesting to the root URL (/) or route. For every other path, it will respond with a 404 Not found.

The example above is actually working serer. Go ahead and click on the URL shown, You will get a response, with real-time logs on the page, and any changes you make will be reflected in real time. This is powered by Runkit, which provides an interactive JavaScript playground connected to a complete node environment that runs in your web browser.

**Basic routing**

Routing refers to determining how an application responds to a client to a particular endpoint, which is a URL (or Path) and a specific HTTP request method (GET,POST and so on).

Each route can have one or more handler functions, which are executed when the route is matched.

Route definitions takes the following structure.

**app.method(Path,Handler)**

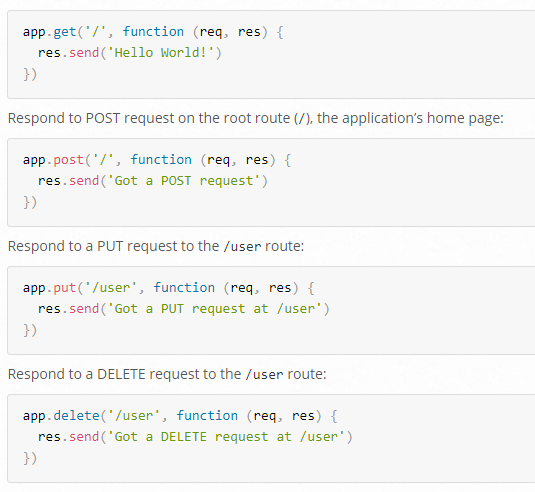
Where

app is an instance of express

Method is an HTTP request method, in lower case.

Path is path on the server

Handler is the function executed when the route is matched.



**allHttpMethod.js**

**const express = require("express");**

**const app = express();**

**const port = 3000;**

**app.get("/",(req,res)=>**

**{**

**console.log("Get method called....");**

**res.send("Get method called..")**

**});**

**app.post("/",(req,res)=>{**

**console.log("post method called....");**

**res.send("post method called..")**

**});**

**app.put("/",(req,res)=>{**

**console.log("Put method called....");**

**res.send("Put method called..")**

**});**

**app.delete("/",(req,res)=>{**

**console.log("Delete method called....");**

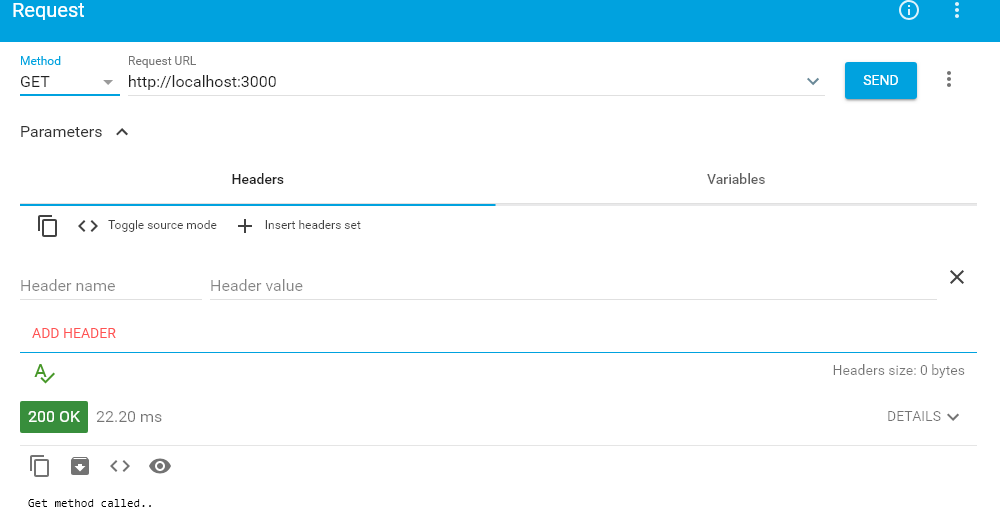
**res.send("Delete method called..")**

**});**

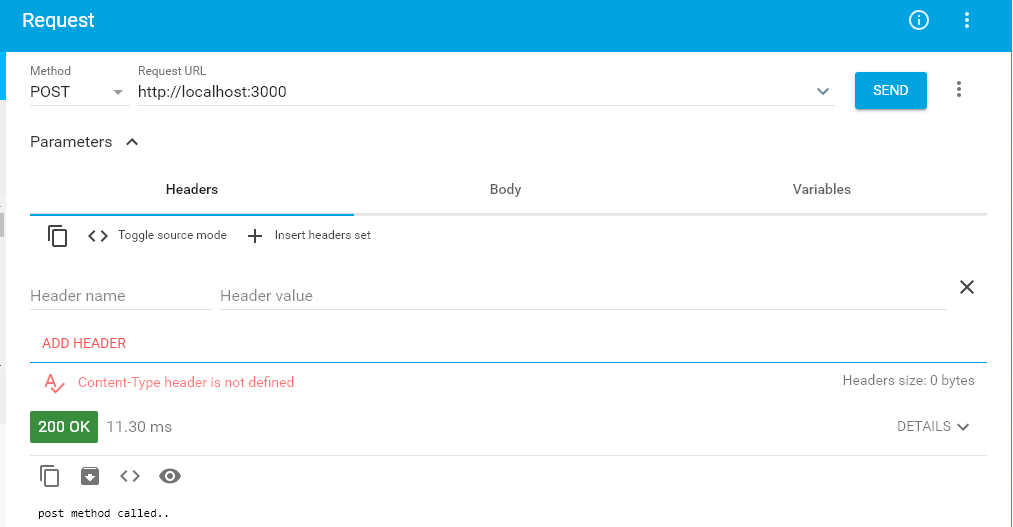
**app.listen(port,()=>console.log(`Server is running on port number ${port}`));**

After running this program using Rest Client browser Features

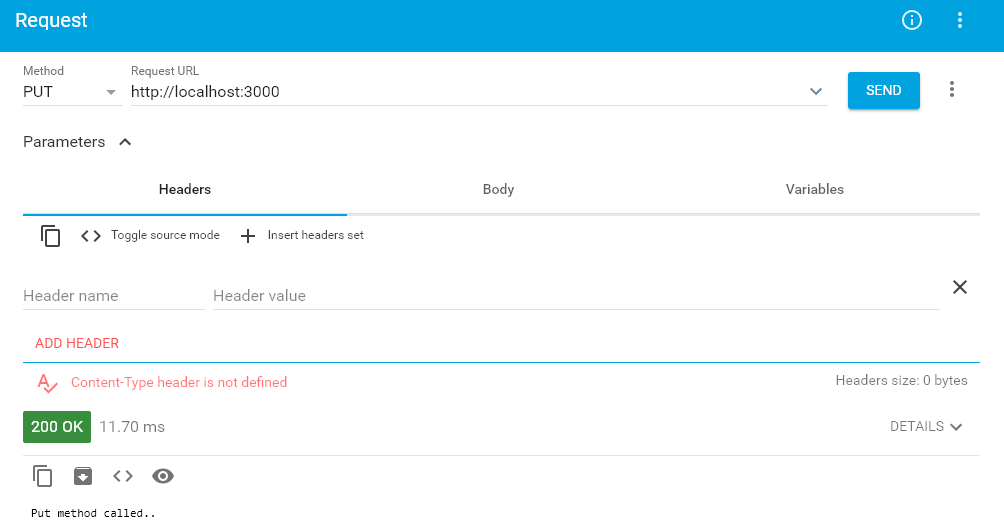
**Get Method**



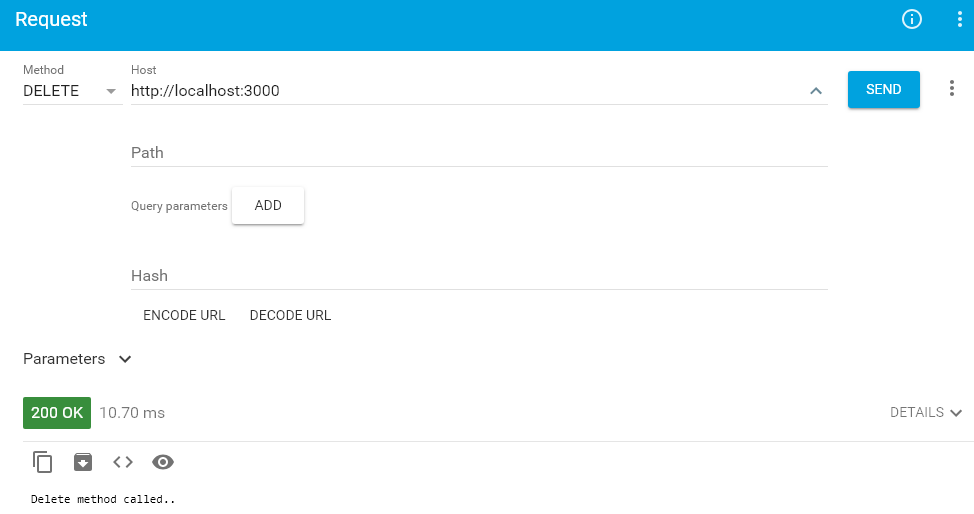
**Post Method**



**Put Method**



**Delete method**



**Node.js with Passing the value through Get method**

**Login.html**

**<<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8" />**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<title>Page Title</title>**

**<meta name="viewport" content="width=device-width, initial-scale=1">**

**<link rel="stylesheet" type="text/css" media="screen" href="main.css" />**

**<script src="main.js"></script>**

**</head>**

**<body>**

**<h2>Login Page</h2>**

**<form action="checkInfo" method="GET">**

**UserName:<input type="text" name="user"><br>**

**Password :<input type="password" name="pass"><br>**

**<input type="submit" value="submit">**

**</form>**

**</body>**

**</html>**

**getMethodLogin.html**

**const express = require("express");**

**const app = express();**

**const path = require("path");**

**app.get("/",(req,res)=>{**

**console.log("Req receive...")**

**res.sendFile(path.join(\_\_dirname+"/Login.html"))**

**});**

**app.get("/checkInfo",(req,res)=>{**

**let user = req.query["user"];**

**let pass = req.query["pass"];**

**if(user=="Akash" && pass=="Kale"){**

**res.send("Successfully Login!");**

**}else {**

**res.send("Failure try once again!");**

**}**

**})**

**const server = app.listen(8989,"localhost",()=>{**

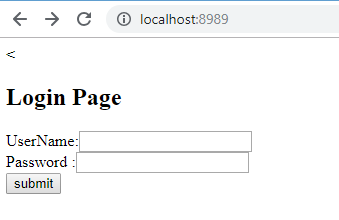
**let host = server.address().address;**

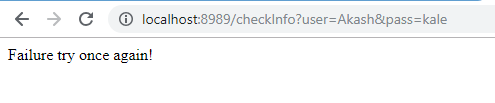
**let port = server.address().port;**

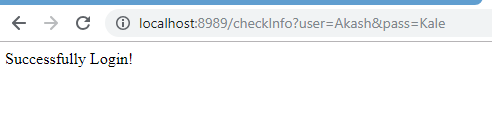
**console.log(`Server address is ${host} and port number is ${port}`);**

**});**

**After hitting on browser**

****

****

****

**Node.js with Passing the value through Post method**

**Login.html**

**<<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8" />**

**<meta http-equiv="X-UA-Compatible" content="IE=edge">**

**<title>Page Title</title>**

**<meta name="viewport" content="width=device-width, initial-scale=1">**

**<link rel="stylesheet" type="text/css" media="screen" href="main.css" />**

**<script src="main.js"></script>**

**</head>**

**<body>**

**<h2>Login Page</h2>**

**<form action="checkInfo" method="POST">**

**UserName:<input type="text" name="user"><br>**

**Password :<input type="password" name="pass"><br>**

**<input type="submit" value="submit">**

**</form>**

**</body>**

**</html>**

**postLoginMethod.js**

**const express = require("express");**

**const bodyParser = require('body-parser')**

**const app = express();**

**const path = require("path");**

**app.get("/",(req,res)=>{**

**console.log("Req receive...")**

**res.sendFile(path.join(\_\_dirname+"/Login.html"))**

**});**

**app.use(bodyParser.urlencoded({ extended: false }))**

**app.post("/checkInfo",(req,res)=>{**

**let user = req.body.user;**

**let pass = req.body.pass;**

**if(user=="Akash" && pass=="Kale"){**

**res.send("Successfully Login!");**

**}else {**

**res.send("Failure try once again!");**

**}**

**})**

**const server = app.listen(8989,"localhost",()=>{**

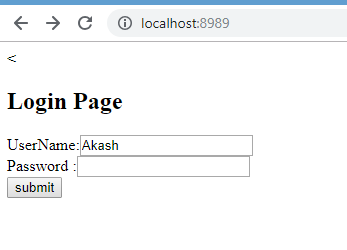
**let host = server.address().address;**

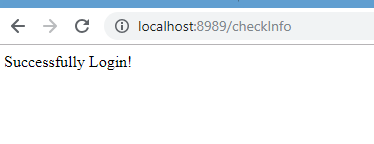
**let port = server.address().port;**

**console.log(`Server address is ${host} and port number is ${port} with post method`);**

**});**

After running this program





**Serving static files in Express**

To serve static files such as images, css files, and JavaScript files, use the express.statis. static build-in middleware function in express

The function signature is

express.static(root,[options])

The root arguments specifies the root directory from which to serve static assets. For more information on the options arguments

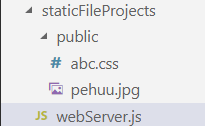
**https://expressjs.com/en/4x/api.html#express.static**

For example, use the following code to serve images, CSS files and JavaScript files in a directory named public.

app.use(express.static("public"))

Now, you can load the files that are in the public directory

Sample projects



webServer.js

**const express = require("express");**

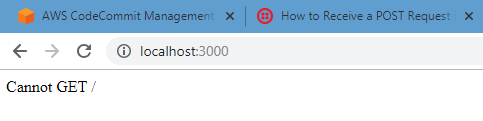
**const app = express();**

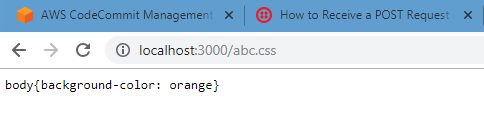
**app.use(express.static("public"));**

**const port = 3000;**

**app.listen(port,()=>console.log(`Server is number on port number ${port}`));**

After run the command as





**pehuu.jsp**

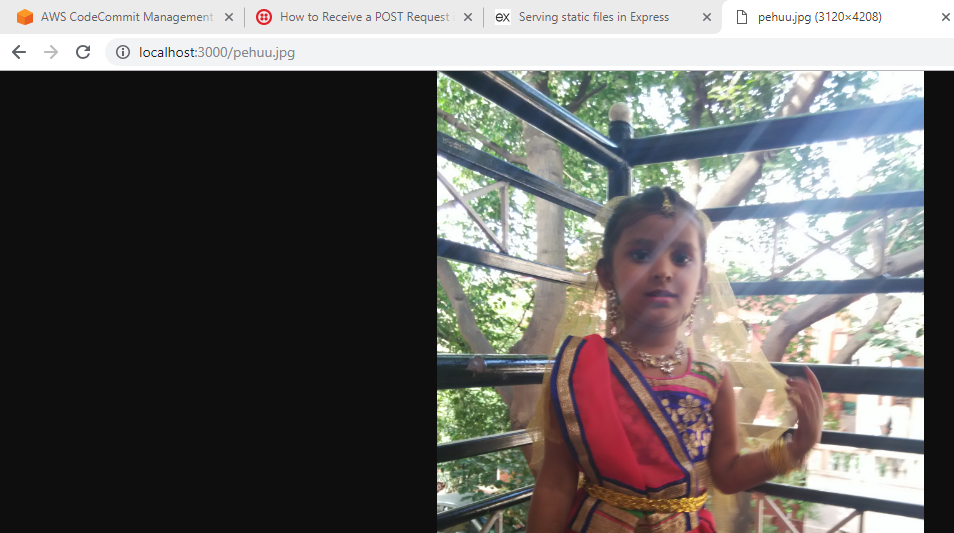


Image contains

**abc.css**

**body{background-color: orange}**

To use multiple static asserts directories, all the express.static middleware function multiple times.

app.use(express.static("public"))

app.use(express.static("files"))

Express looks up the files in the order in which your set the static directories with the express.static middleware function.

Note : for the best results, user a reverse proxy cache to improve performance of serving static assets.

https://expressjs.com/en/advanced/best-practice-performance.html#use-a-reverse-proxy

To create a virtual path prefix (where the path does not actually exist in the file system) for files that are served by the expres.static function, specify a mount path for the static directory, as show below.

app.use("/static",express.static("public"))

Now you can load the files that in the public directory form the /static path prefix.

**http://localhost:3000/static/images/kitten.jpg**

However, the path that you provide to the express.static function is relative to the directory from where you launch your node process. If you run the express app from another directory, it is safer to use the absolute path of the directory that you want to serve.

app.use("/static",express.static(path.join(\_dirname,"public")))

**Express application generator**

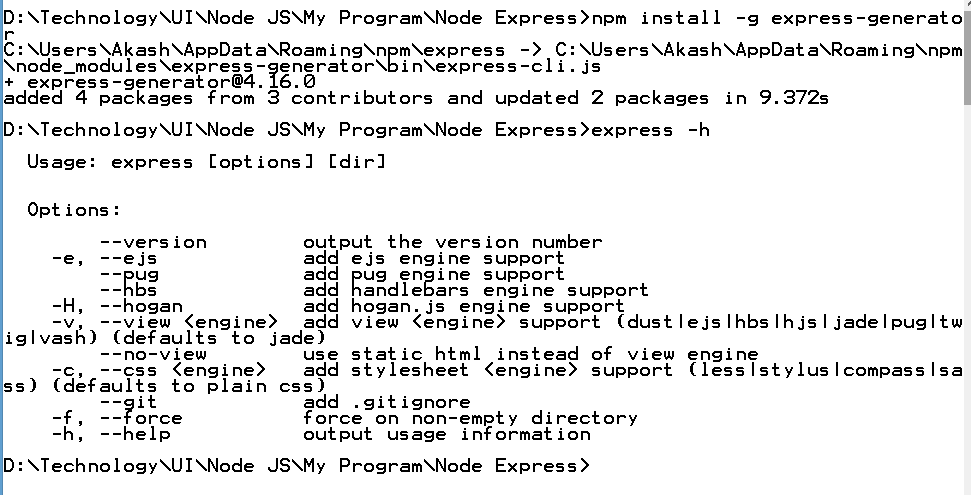
Use the application generator tool, express-generator, to quickly create an application skeleton.

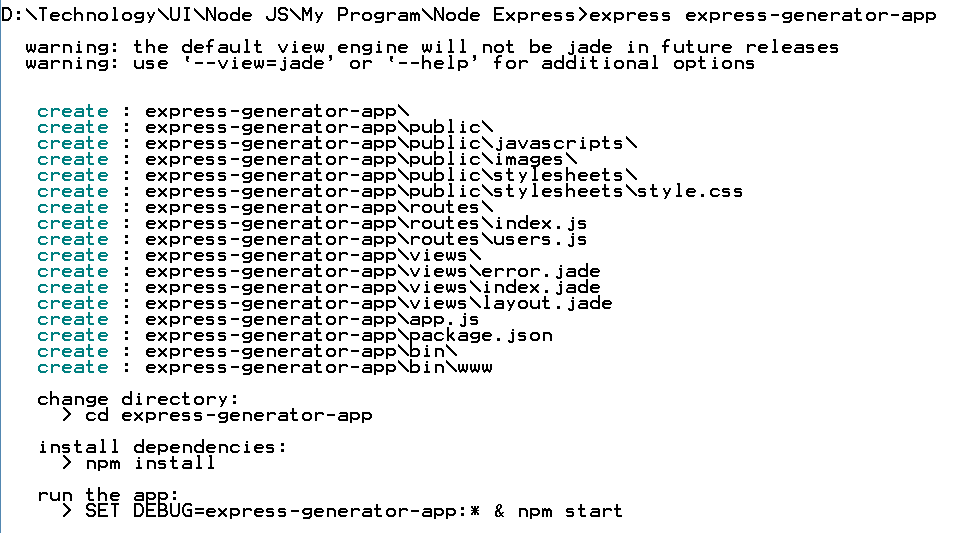
The express-generator package installs the express command-line tool. Use the following command to do so.

To install the express-generator we have to use the command as

**npm install-g express-generator**

With the exress-generator we can see more then options.





**Routing**

Routing refers to how an application's endpoints (URIs) respond to client requests.

You define routing using methods of the Express app object that correspond to HTTP methods. for example app.get() to handle GET request and app.post() to handle POST requests. You can also use app.all() to handle all HTTP methods and app.user() to specify middleware as the callback functions.

These routing methods specify a callback functions (sometimes called "handler functions") called when the application receives a request to the specified route (endpoint) and HTTP method. In other words, the application listens for requests that match the specified routes(s) and methods, and when it detects a match it calls the specifies callback functions.

In fact, the routing methods can have more than one callback function as arguments. With multiple callback functions, it is important to provide next as a arguments to the callback function and then call next() within the body of the function to hand off control to the next callback.

var express = require("express")

var app = express()

app.get("/", function(req,res){

res.send("hello world");

})

app.post("/",function(req,res){

res.send("Post method");

})

Express support methods that correspond to all HTTP request methods. get, post, and so on.

There is a special routing method app.all(), use to load middleware functions at a path for all HTTP request methods. for example the following handler is executed for request to the route "/" whether using GET,POST, PUT or DELETE or any other HTTP request method supported by http module.

**app with all method example**

**allMethod.js**

**const express = require("express");**

**const app = express();**

**const port = 3000;**

**app.all("/",(req,res)=>**

**{**

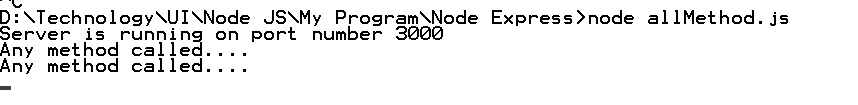
**console.log("Any method called....");**

**res.send("All Method "+req.method)**

**});**

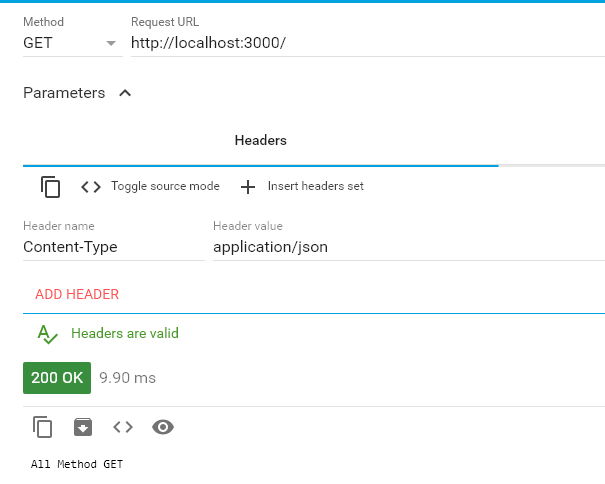
**app.listen(port,()=>console.log(`Server is running on port number ${port}`));**

After running the program using command as

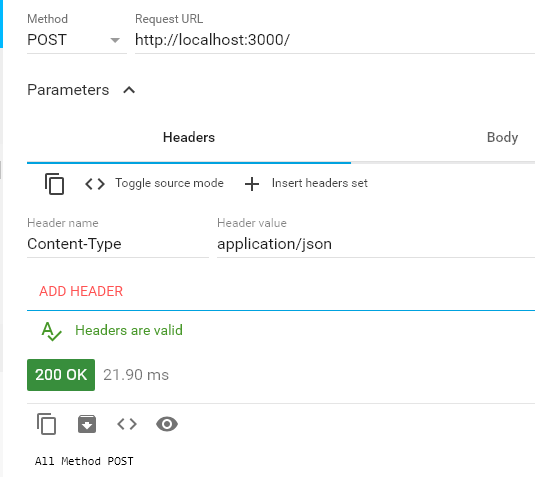


Sending the request using post man client.

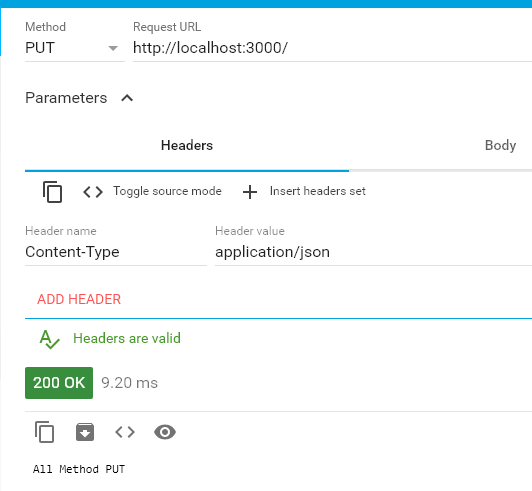
Get method



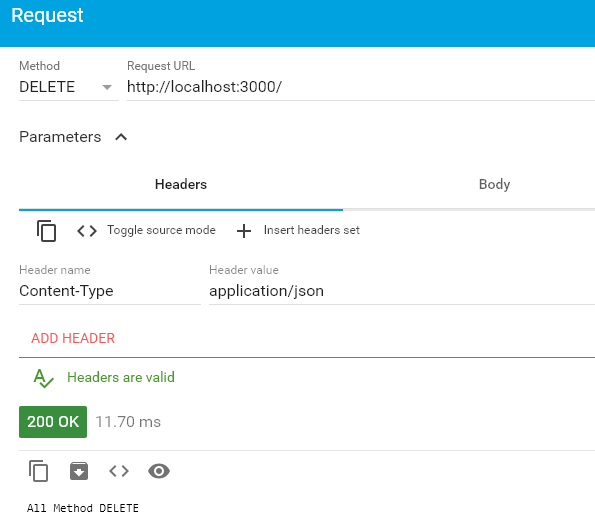
Post method



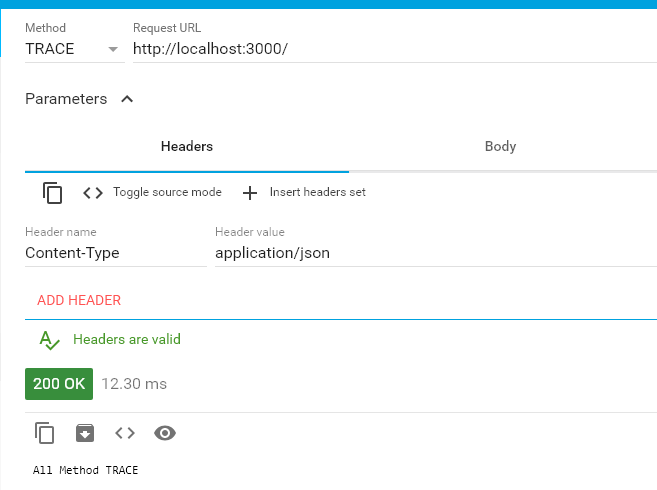
Put method



Delete method



Trace method



**Route paths**

Route paths, in combination with a request method. define the endpoints at which requests can be made. Route paths can be string, string pattern or regular expression.

The character ?,+, \* and () are subsets of their regular expression counter parts. The hyphen (-) and the (.) are interpreted liberally by sting - based paths.

If you need to use the dollar character (&) in a path string, enclose if escaped within ([ and ]). For example, the path string for requests at "/data/$book", would be "/data/([\$])book.

The route path with ?

**getQuestionMark.js**

**const express = require("express");**

**const app = express();**

**const port = 3000;**

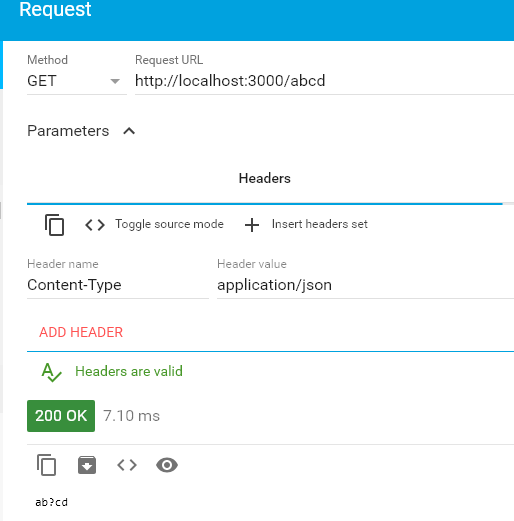
**app.get('/ab?cd', function (req, res) {**

**res.send('ab?cd')**

**})**

**app.listen(port,()=>console.log(`Server is running on port number ${port}`));**

This route path will match abcd and acd



**getPlusMark.js**

**const express = require("express");**

**const app = express();**

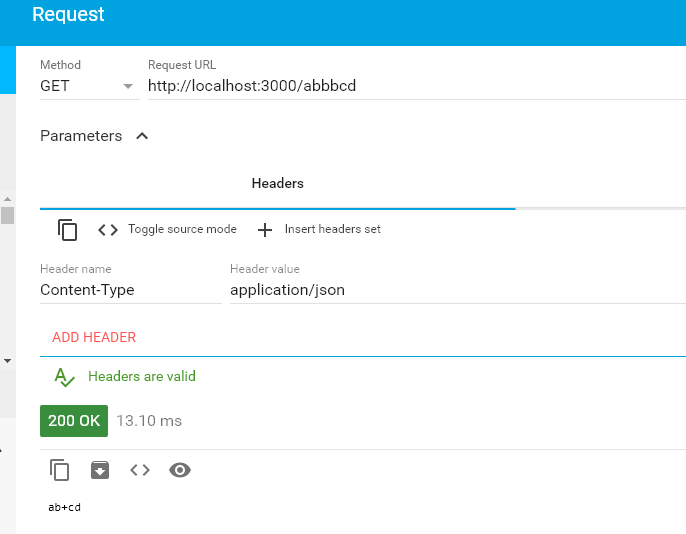
**const port = 3000;**

**app.get('/ab+cd', function (req, res) {**

**res.send('ab+cd')**

**})**

**app.listen(port,()=>console.log(`Server is running on port number ${port}`));**



This route path will match abcd, abbcd, abbbcd and so on.

**getStarMark.js**

**const express = require("express");**

**const app = express();**

**const port = 3000;**

**app.get('/ab\*cd', function (req, res) {**

**res.send('ab\*cd')**

**})**

**app.listen(port,()=>console.log(`Server is running on port number ${port}`));**

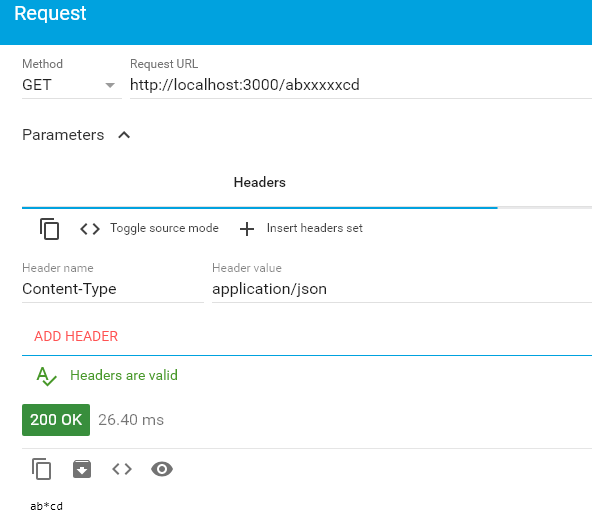
This route path will match abcd, abxyz, absafasfcd and so on.

This route path will match /abe and /abcde

app.get("/ab(cd)?e", function(req,res){

res.send("ab(cd)?e");

})



Example of route paths based on regular expression:

This route path will match anything with an "a" in it.

app.get(/a/,funtion(req,res){

res.send("/a/");

})

This route path will match butterfly and dragonfly, but not butterflyman, dragonflyman and so on.

app.get(/.\*fly$/,function(req,res){

res.send("/.\*fly$");

})

**Route parameters**

https://developer.okta.com/blog/2018/09/13/build-and-understand-express-middleware-through-examples