**NodeJS**

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code on server-side.

**Installation:**

**Linux:**

Open Terminal and type the following:

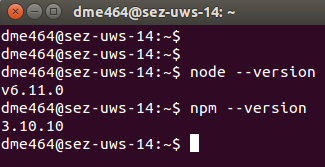
sudo apt-get update

curl -sL https://deb.nodesource.com/setup\_6.x | sudo -E bash

sudo apt-get install nodejs

node --version

npm --version



**Express:**

Express is a minimal and flexible Node.js web application framework.

Before installing express we need create a project directory, so in Terminal create a directory.

mkdir test

cd test

Now we need create a package.json file.

A package.json - file holds various metadata relevant to the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

Open Terminal and type the command below to create a package.json file.

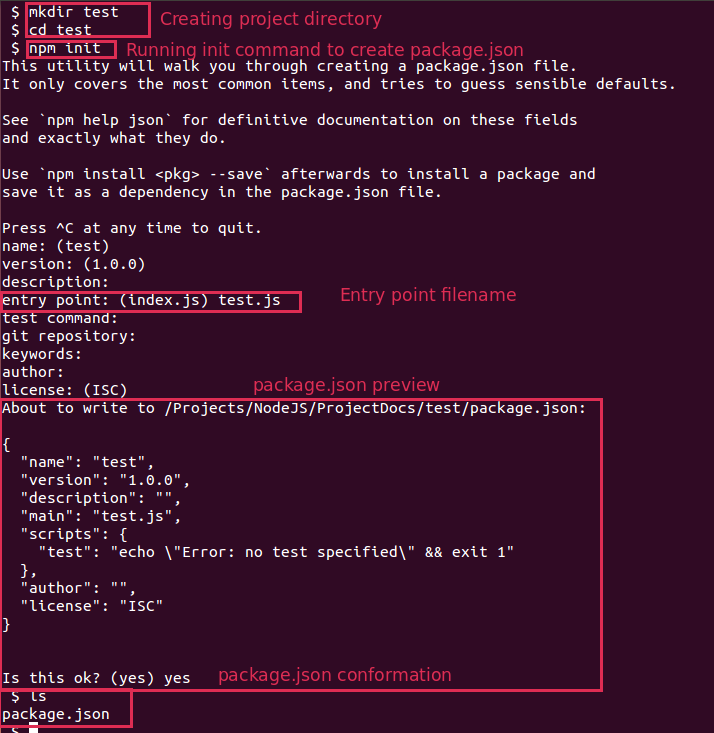
npm init

Now npm will ask for some information like name, version, description, entry point, etc. We will skip all other fields except ‘entry point’ field. In the entry point type test.js.

Note that, all these fields are optional so its ok to skip the fields.

At the end npm will show a preview of our package.json and ask confirm to create the file. Type ‘yes’ and press return key to create the package.json file.

Now type ls command in terminal to verify if the package.json file is created.



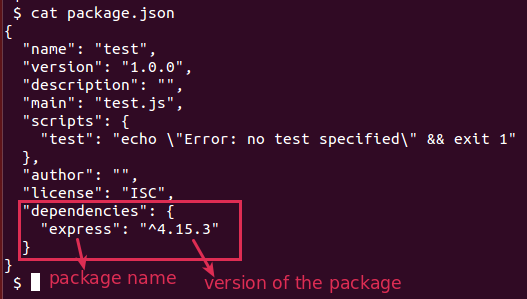
Now that we created the package.json file, we can now install express framework.

Type the following command

npm install express --save

This will create the ‘node\_modules’ directory in the current directory(if one doesn't exist yet), and will download the express package to that directory.

Now open package.json file and see if express listed in the ‘dependencies’.



**Note:-**

* Npm will install all it’s packages in the ‘node\_modules’ directory.
* The ‘-- save’ command at the end of the npm install command is used to save the package details in package.json file. All the packages installed with ‘--save’ command will be listed under ‘dependencies’ property in the package.json file.

**Working:**

1. **Intro to ES6 features**

EcmaScript is the "official" name for JavaScript and ES6 is short-hand for EcmaScript 6. ie, ES6 is the 6th version of JavaScript.

Following are the some of the new ES6 features that we will encounter in NodeJS:

**let and const keywords:**

let and const are newly introduced in ES6. They mainly used for block scoping.

Declaring variables with let or const within a code block restricts their visibility to that block only. This is called block scoping.

A variable declared with let keyword will only available inside the block. example

example 1:

for (var i=0; i < 5; i++) {}

console.log('i= ' + i) // i= 5

for (let i=0; i < 5; i++) {}

console.log('i= ' + i) // Uncaught ReferenceError: i is not defined

const allows us to declare constant variables. const does not indicate that the assigned value will be

constant, but that the binding with the value is constant.

const person = {}

person.name = 'joe' // "joe"

person = null // Uncaught TypeError: Assignment to constant variable.

**Arrow functions:**

An arrow function expression has a shorter syntax than a function expression.

Example1:

const numbers = [2, 6, 7, 8, 1];

const even = numbers.filter(function(x) {

return x%2 === 0;

});

const numbers = [2, 6, 7, 8, 1];

const even = numbers.filter(x => x%2 === 0);

1. **Creating Web Server**

NodeJS doesn’t need a server application to serve pages, instead it can create its own server.

Lets create a webserver to say ‘hello world’. Open the project folder and paste the following lines

// Gets http system module

*const* http = require('http')

// Creates a web server and listens on port 4000

http.createServer((*req*, *res*) *=>* {

// Sets status code and header

res.writeHead(200, {'content-type': 'text/plain'})

// Writes the content

res.end('Hello world')

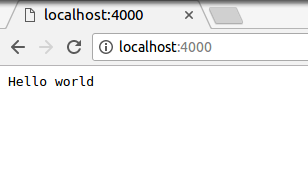
}).listen(4000)

To run the code open terminal in the project folder and type

node test.js

Now this url <http://localhost:4000/> in a browser and you should see page like below.

**Note:-** To stop the server press Ctrl + C



1. **Routing using express**

**Simple routing:**

Stop the server if running and paste the code below in test.js

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

http.createServer(*function* (*req*, *res*) {

// normalize url by removing querystring, optional

// trailing slash, and making it lowercase

*var* path = req.url.replace(/\/?(?:\?.\*)?$/, '').toLowerCase();

switch (path) {

case '':

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

res.end('<h1>Homepage</h1>');

break;

case '/about':

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

res.end('<h1>About</h1>');

break;

default:

res.writeHead(404, { 'Content-Type': 'text/html' });

res.end('<h1>Not Found</h1>');

break;

}

}).listen(4000);

*console*.log('Server started on localhost:4000; press Ctrl-C to terminate....');

Now open the following links:

<http://localhost:4000/> will serve homepage

<http://localhost:4000/about> will serve about page

<http://localhost:4000/foo> will serve not found page with 400 status

Here, req.url property will gets the full url. Then with the using a regular expression we will filter the url to get the only the path(eg: /about) of the url. This path then matched to get correct route.

**Routing Using Express:**

The above example using node will seems simple but its get complex when we want a pass data with the url. Also setting header and content type for each request break the DRY coding principles. This is where the express come into the picture. Now replace the existing code with the following code and run server.

// getting express module

*const* express = require('express')

*const* app = express()

*const* port = 4000 // port

// Home page route

app.get('/', *function* (*req*, *res, next*) {

res.send('<h1>Homepage</h1>')

})

// About page route

app.get('/about', *function* (*req*, *res, next*) {

res.send('<h1>About</h1>')

})

// Hello page route

app.get('/hello/:name', *function* (*req*, *res, next*) {

// Gets name from query string

*let* name = req.params.name

res.send('<h1>Hello ' + name + '</h1>')

})

// Error page route

app.get('/error', *function* (*req*, *res*, *next*) {

// passing a custom error to error handler route

next(new Error('testing error'))

})

// Error handler route

app.use(*function* (*err*, *req*, *res*, *next*) {

// shows error stack in console

*console*.error(err.stack)

res.status(500)

// shows error message

res.send('<h1>Server Error</h1> <br/> Error Message: <b>' +

err.message + '</b>'

)

})

// 404 page route

app.use(*function* (*req*, *res, next*) {

res.status(404);

res.send('<h1>Not Found</h1>')

})

// Creates server and listens to the port

app.listen(port, *function* () {

*console*.log('Express started on http://localhost:' +

port + '; press Ctrl-C to terminate.')

})

Now open the following links:

<http://localhost:4000/> will serve homepage

<http://localhost:4000/about> will serve about page

<http://localhost:4000/foo> will serve not found page

[http://localhost:4000/hello/Joe](http://localhost:4000/foo) will serve hello page

[http://localhost:4000/error](http://localhost:4000/foo) will serve error handler page

Notice that we didn’t filter the url, content-type header and status code; all this will be handled by the express framework.

**Express Routing explained:**

app.get() function is used to handle HTTP GET request.

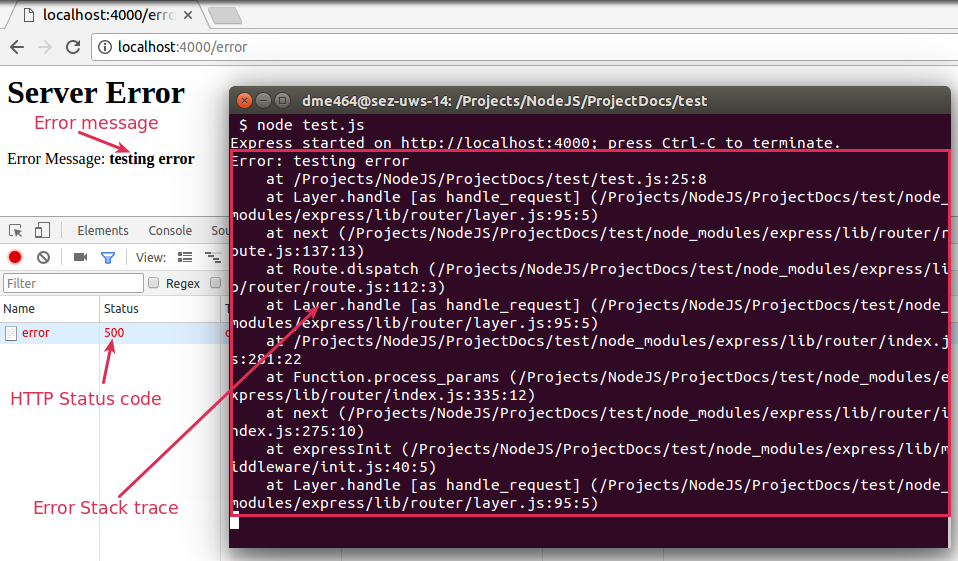
The first argument to get() function is path and second argument is the a callback function.

When a url is called in the browser, then express will get that request and search for the function that matches the path.

If the express finds the match, then then the callback function of the corresponding get() function is executed and if express couldn't find the match, then last 404 route will be executed.

The callback function of the get() function have must have three arguments (req and res are required and next is optional). The three arguments are req - which is the request object, res - which is the response object and next - which is a callback function.

In example url [http://localhost:4000/error](http://localhost:4000/foo) , an error object is passed to the ‘next’ callback function. The express will call get this error object in the ‘Error handler route’ which is the second last route.



The error handler route will then console stack to the terminal and then displays the error message in the error page. This is the common way of handling error in nodeJS.

In example url <http://localhost:4000/hello/Joe> ‘Joe’ is a query string data passed with url. In the app.get () function the path string ‘:name’ indicates that ‘name’ part is query string. We can access query strings using ‘req.params’ property

**Important Note:-**

The order of the routes are important. The 404 page route should be place at the end.

Always pass the error to the error handler as an error object using the ‘next’ function. When we pass error as an error object, the error stack will contain the line number from where it was called.

1. **Templating using ejs package**

EJS is a simple templating language that combines data and a template to produce HTML.

**Installation:**

npm install ejs --save

To use ejs we must first set the view path and view engine in express. Then by calling ‘res.render()’ function will render the page. The following code will demonstrate the basic working of ejs.

Open test.js and paste the following code.

// getting express module

*const* express = require('express')

*const* app = express()

*const* port = 4000 // port

// Setting view directory

app.set('views', './views')

// Setting view engine

app.set('view engine', 'ejs')

// Home page route

app.get('/', *function* (*req*, *res*, *next*) {

res.send('<h1>Homepage</h1>')

})

// About page route

app.get('/about', *function* (*req*, *res*, *next*) {

res.send('<h1>About</h1>')

})

// Hello page route

app.get('/hello/:name', *function* (*req*, *res*, *next*) {

// Gets name from query string

*let* name = req.params.name

// Renders view using ejs template

res.render('hello', { name: name })

})

// Error page route

app.get('/error', *function* (*req*, *res*, *next*) {

// passing a custom error to error handler route

next(new Error('testing error'))

})

// 404 page route

app.use(*function* (*req*, *res*, *next*) {

res.status(404)

res.send('<h1>Not Found</h1>')

})

// Error handler route

app.use(*function* (*err*, *req*, *res*, *next*) {

// shows error stack in console

*console*.error(err.stack)

res.status(500)

// shows error message

res.send('<h1>Server Error</h1> <br/> Error Message: <b>' + err.message +

'</b>')

})

app.listen(port, *function* () {

*console*.log('Express started on http://localhost:' +

port + '; press Ctrl-C to terminate.')

})

Now create a directory named ‘views’, then in that directory create new file and name it hello.ejs.

In hello.ejs paste the following code:

<% if (name) {%>

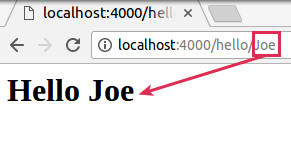
<h1>Hello <%= name %></h1>

<% } else { %>

<h1>Hello</h1>

<% } %>

Now start the server and open this url <http://localhost:4000/hello/Joe> , you will see the following message.



In test.js ‘res.render()’ function has two arguments, first is the name of the view page and second is the data object. The properties set in the data object will be accessible in view page.

The view file is should be save as .ejs extension, otherwise an error will be thrown.

‘<%= %>’ is the ejs tag to outputs the value into the template and ‘<% %>’ is the tag, for control-flow.

Ref:

<https://expressjs.com/en/guide/using-template-engines.html>

<https://www.npmjs.com/package/ejs#usage>

1. **Template Layouts with EJS**

EJS does not specifically support layouting, but layouts can be implemented by including headers and footers.

First create a directory in the root called ‘public’ and

Open ‘views’ directory and create a new directly called ‘layouts’ and in that directory create two ejs files names ‘header.ejs’ and ‘footer.ejs’

1. **Database connection using sequelize package**

Sequelize is a ORM for Nodejs. It supports the dialects PostgreSQL, MySQL, SQLite and MSSQL and features solid transaction support, relations, read replication and more.

**Installation:**

npm install mysql --save

npm install sequelize --save

sd

1. CURD operation
2. Validation using validate.js
3. Realtime time updates using socket.io package
4. Virtual hosting using vhost package