NodeJS

1. <https://code.tutsplus.com/tutorials/authenticating-nodejs-applications-with-passport--cms-21619>
2. Passport also needs to serialize and deserialize user instance from a session store in order to support login sessions, so that every subsequent request will not contain the user credentials. It provides two methods serializeUser and deserializeUser for this purpose.
3. body-parser

* <https://medium.com/@adamzerner/how-bodyparser-works-247897a93b90>
* if you post form data as content-type: application/json, you need to set app.use(bodyparser.json())
* if you post data as url-encoded (default one), you need to set app.use(bodyparser.urlencoded({extended: true}))
* <https://stackoverflow.com/questions/29960764/what-does-extended-mean-in-express-4-0>
* body-parser extracts the entire body portion of an incoming request stream and exposes it on req.body as something easier to interface with. You don't need it per se, because you could do all of that yourself. However, it will most likely do what you want and save you the trouble.
* To go a little more in depth; body-parser gives you a middleware which uses [nodejs/zlib](https://nodejs.org/api/zlib.html) to unzip the incoming request data if it's zipped and [stream-utils/raw-body](https://github.com/stream-utils/raw-body) to await the full, raw contents of the request body before "parsing it" (this means that if you weren't going to use the request body, you just wasted some time).
* [bodyParser.urlencoded()](https://github.com/expressjs/body-parser#bodyparserurlencodedoptions): Parses the text as URL encoded data (which is how browsers tend to send form data from regular forms set to POST) and exposes the resulting object (containing the keys and values) on req.body. For comparison; in PHP all of this is automatically done and exposed in $\_POST.
* [bodyParser.json()](https://github.com/expressjs/body-parser#bodyparserjsonoptions): Parses the text as JSON and exposes the resulting object on req.body.

1. Different Views Template?
2. Maintain sessions – express-session?

* <https://stormpath.com/blog/everything-you-ever-wanted-to-know-about-node-dot-js-sessions>

1. JWT?
2. Express-session vs passport authentication?
3. MEAN stack application dir structure
4. Nodejs application dir structure
5. Passport – facebook, twitter authentication?
6. Why we need to serialize and deserialize user in passport authentication?
7. Use of secret in express nodejs

* ‘**secret**‘ is used for cookie handling etc but we have to put some secret for managing Session in Express.
* The secret is a random, high-entropy string you create to encrypt the cookie. We need to take this step because the browser is an inherently untrusted environment; anyone with access can open it up and see what’s stored in there. Client-sessions will encrypt and decrypt all the cookie values, so you don’t have to worry about prying eyes.

1. Unzip and read file content

* fs.createReadStream('path/to/archive.zip').pipe(unzip.Extract({ path: 'output/path' }));

fs.readFile(FILE\_LOCATION, function (err, data) {

if (err) throw err;

if(data.indexOf('search string') >= 0){

console.log(data)

}

});

1. Difference between dependencies and dev-dependencies

* <https://www.linkedin.com/pulse/npm-dependencies-vs-devdependencies-daniel-tonon>
* <https://stackoverflow.com/questions/18875674/whats-the-difference-between-dependencies-devdependencies-and-peerdependencies>

1. Difference betwwen tilde (~) and ^ operator

* "express": "^4.0.0" will install latest version express 4.\*.\*
* "express": "~4.0.0" will install latest version of express 4.0.\*

1. Why alert not working in nodejs? - The alert() function is a property of browser window objects. It is not really part of JavaScript.
2. Concurrency while multiple users modifying file simultaneously?
3. Gulp?
4. Create a module that returns the current date and time:

exports.myDateTime = function () {  
    return Date();  
};

**var dt = require('./myfirstmodule');**

http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write("The date and time are currently: " + **dt.myDateTime()**);  
    res.end();  
}).listen(8080);

1. we use ./ to locate the module, that means that the module is located in the same folder as the Node.js file.
2. Nodemailer
3. Template Engines

app.use(‘views’, ‘./src/views’);

app.set(‘view engine’, ‘jade’);

1. Jade

* Without tags
* Hierarchy using tabs
* H1 (class=’h1class’)
* Ul

Each value in [‘1’, ‘2’]

li = value

1. Handlebars

* {{title}}
* {{#each [‘1’, ‘2’]}}

<li>{{this}}</li>

{{/each}}

1. Ejs

* <%=title%>
* <% for (var I = 0; I < list.length; i++) { %>

<li><%=list[i]%></li>

<%}%>

* <%- include('partials/header') %>

1. \_\_dirname
2. MongoDB stores data in a JSON-like format,
3. Create sample project using angular –cli
4. If you have installed the "MongoDB" then open a command prompt window and run the "mongod" command, this command run the "MongoDB" server at "27017" port and ready to connection with a client.
5. run the "mongo" command, this command create a client and establish the connection to server running at port "27017".
6. <https://github.com/Pankajmalhan/MeanStack2>
7. <https://scotch.io/bar-talk/setting-up-a-mean-stack-single-page-application>
8. <https://www.codeproject.com/Articles/1185294/Build-Modern-App-with-MEAN-Stack-Part>
9. How to make sure all developers using same version of npm modules - .npmrc?

* Use npm config set save=true
* Use npm config set save-exact=true

1. How to create package for angular 2 application?
2. Change location globaly installed modules – npm config set –prefix /path/
3. Use of tsconfig
4. Node does not have a window global object, but instead has two others: globals and process.
5. <https://javascriptplayground.com/blog/2013/06/think-async/>
6. Use of next() - <https://stackoverflow.com/questions/10695629/what-is-the-parameter-next-used-for-in-express>

app.get("/name", function(*httpRequest*, *httpResponse*, *next*){

httpResponse.write("Hello");

next(); //remove this and see what happens

});

app.get("/name", function(*httpRequest*, *httpResponse*, *next*){

httpResponse.write(" World !!!");

httpResponse.end();

});

Will print Hello World !!!

1. Debug nodejs application
2. Secure route
3. Async
4. <https://www.journaldev.com/7462/node-js-architecture-single-threaded-event-loop>
5. <https://stackoverflow.com/questions/15349733/setimmediate-vs-nexttick>
6. <https://stackoverflow.com/questions/17502948/nexttick-vs-setimmediate-visual-explanation>

* nextTick handlers are run right after each call from C++ into JavaScript. That means that, if your JavaScript code calls process.nextTick, then the callback will fire as soon as the code runs to completion, but before going back to the event loop. The race is over, and all is good.
* However, there are programs out in the wild that use recursive calls to process.nextTick to avoid pre-empting the I/O event loop for long-running jobs. In order to avoid breaking horribly right away, Node will now print a deprecation warning, and ask you to use setImmediate for these kinds of tasks instead.
* setImmediate callbacks are fired off the event loop, once per iteration in the order that they were queued.
* <http://voidcanvas.com/setimmediate-vs-nexttick-vs-settimeout/>

1. Single thread means single call stack
2. <https://stackoverflow.com/questions/5062614/how-to-decide-when-to-use-node-js?rq=1>
3. <https://javascriptplayground.com/blog/2013/06/think-async/>
4. <https://stackoverflow.com/questions/20186081/understanding-node-js-async-parallel>
5. Express.js is based on the Node.js middleware module called *connect* which in turn uses **http**module. So, any middleware which is based on connect will also work with Express.js.
6. Zip file

var fs = require("fs");

var zlib = require('zlib');

// Compress the file input.txt to input.txt.gz

fs.createReadStream('input.txt')

  .pipe(zlib.createGzip())

  .pipe(fs.createWriteStream('input.txt.gz'));

  console.log("File Compressed.");

1. Unzip file

var fs = require("fs");

var zlib = require('zlib');

// Decompress the file input.txt.gz to input.txt

fs.createReadStream('input.txt.gz')

  .pipe(zlib.createGunzip())

  .pipe(fs.createWriteStream('input.txt'));

  console.log("File Decompressed.");

1. Async. Queue
2. Async.parallel
3. Generators - <https://www.guru99.com/node-js-generators-compare-callbacks.html>
4. How process object turn sync call to async callback
5. File descriptor
6. Unit testing
7. <https://nodejs.org/en/blog/npm/managing-node-js-dependencies-with-shrinkwrap/>
8. Nodejs performance optimization techniques

* Use Cluster
* Use compression to compress request payload
* Use cache
* Use asynchronous library
* Remove unnecessary modules
* Client side rendering (angular instead of express)

1. Nodejs security

* Bcrypt
* Session management using express
* Authentication using passport

1. Morgan – http request logger middleware
2. <https://www.sitepoint.com/configuring-nginx-ssl-node-js/>
3. Express-validator - <https://booker.codes/input-validation-in-express-with-express-validator/>
4. You can enforce validations at database layer as well.
5. HTTP is stateless protocol. It stores no information about previous visit and Express solves this problem very beautifully using session
6. Express-session

* Secret mandatory to set while initialize session – app.use(session({secret: ‘df’}));
* **Secret.** This is a value used in the signing of the session ID that is stored in the cookie. (encrypt cookie)
* **Set secure true and httponly.**
* **Set maxAge for limiting session**

1. Cluster

* worker is connected to its master via its IPC channel

1. Child Process

* <https://medium.freecodecamp.org/node-js-child-processes-everything-you-need-to-know-e69498fe970a>
* Every child process also gets the three standard stdio streams, which we can access using child.stdin, child.stdout, and child.stderr.
* When those streams get closed, the child process that was using them will emit the close event. This close event is different than the exit event because multiple child processes might share the same stdio streams and so one child process exiting does not mean that the streams got closed.
* stdout/stderr streams are readable streams while the stdin stream is a writable one.
* Spawn
* The spawn function is a much better choice when the size of the data expected from the command is large, because that data will be streamed with the standard IO objects.
* Fork
* The fork function is a variation of the spawn function for spawning node processes. The biggest difference between spawn and fork is that a communication channel is established to the child process when using fork, so we can use the send function on the forked process along with the global process object itself to exchange messages between the parent and forked processes.
* execFile
* If you need to execute a file without using a shell, the execFile function is what you need. It behaves exactly like the exec function, but does not use a shell, which makes it a bit more efficient. On Windows, some files cannot be executed on their own, like .bat or .cmd files. Those files cannot be executed with execFile and either exec or spawn with shell set to true is required to execute them.
* Exec
* the spawn function does not create a shell to execute the command we pass into it. This makes it slightly more efficient than the exec function, which does create a shell. The exec function has one other major difference. It buffers the command’s generated output and passes the whole output value to a callback function (instead of using streams, which is what spawn does). The exec function is a good choice if you need to use the shell syntax and if the size of the data expected from the command is small. (Remember, execwill buffer the whole data in memory before returning it.) Note that using the shell syntax comes at a [security risk](https://blog.liftsecurity.io/2014/08/19/Avoid-Command-Injection-Node.js/) if you’re executing any kind of dynamic input provided externally. A user can simply do a command injection attack using shell syntax characters like ; and $ (for example, command + ’; rm -rf ~’ )

1. Node is named Node to emphasize the idea that a Node application should comprise multiple small distributed nodes that communicate with each other.
2. Each worker process here will have its own event loop and memory space.
3. readFile or createReadStream – readfile loaded all content into memory before writing.so in case of reading big file it will eat good amount of memory. While createreadstream gives content in chunk.
4. A duplex streams is both Readable and Writable. An example of that is a TCP socket.
5. File upload?
6. Microservice?
7. When you do require(‘module’), it will look for module in all directories which you can find listed using module.paths in REPL.
8. The whole process of requiring/loading a module is synchronous. That’s why we were able to see the modules fully loaded after one cycle of the event loop.
9. If a file extension was not specified, the first thing Node will try to resolve is a .js file. If it can’t find a .js file, it will try a .json file and it will parse the .json file if found as a JSON text file. After that, it will try to find a binary .node file.
10. We can use the exports object to export properties, but we cannot replace the exports object directly because it’s just a reference to module.exports

exports.id = 42; // This is ok.

exports = { id: 42 }; // This will not work.

module.exports = { id: 42 }; // This is ok.

1. There is nothing special about require. It’s an object that acts mainly as a function that takes a module name or path and returns the module.exportsobject.
2. If you need to define a new listener, but have that listener invoked first, you can use the prependListener method:

withTime.on('data', (data) => {  
 console.log(`Length: ${data.length}`);  
});  
  
withTime.prependListener('data', (data) => {  
 console.log(`Characters: ${data.toString().length}`);  
});

1. net module ?
2. <https://stackoverflow.com/questions/42616120/what-is-the-relationship-between-node-js-and-v8>
3. exports variable inside each module is just a reference to module.exports which manages the exported properties.
4. Can different versions of the same package be used in the same application?
5. Use REPL in strict mode = node –use strict
6. node-gyp
7. <https://stackoverflow.com/questions/22213980/could-someone-explain-what-process-argv-means-in-node-js-please>
8. Nodejs performance tuning

* Use cluster
* Use async apis
* Serve static content using nginx or cdn
* Use gzip to compress request response data
* Use es6 features
* Remove unused middleware
* Use client side rendering
* Use indexing

1. Project description?
2. New relic for performance monitoring?
3. Set up nginx?
4. Session info stored in db rather than in memory?
5. What you can improve in your application?
6. How to deliver nodejs application to client?
7. Microservice?
8. Nodejs testing
9. <https://medium.freecodecamp.org/8-npm-tricks-you-can-use-to-impress-your-colleagues-dbdae1ef5f9e>
10. NodeJs Performance Optimization