**1) What is node.js?**

Node.js is a Server side scripting which is used to build scalable programs. Its multiple advantages over other server side languages, the prominent being non-blocking I/O.

**2) How node.js works?**

Node.js works on a v8 environment, it is a virtual machine that utilizes JavaScript as its scripting language and achieves high output via non-blocking I/O and single threaded event loop.

**3) What do you mean by the term I/O ?**

I/O is the shorthand for input and output, and it will access anything outside of your application. It will be loaded into the machine memory to run the program, once the application is started.



**4) What does event-driven programming mean?**

In computer programming, event driven programming is a programming paradigm in which the flow of the program is determined by events like messages from other programs or threads. It is an application architecture technique divided into two sections 1) Event Selection 2) Event Handling

**5) Where can we use node.js?**

Node.js can be used for the following purposes

a) Web applications ( especially real-time web apps )

b) Network applications

c) Distributed systems

d) General purpose applications

**6) What is the advantage of using node.js?**

a) It provides an easy way to build scalable network programs

b) Generally fast

c) Great concurrency

d) Asynchronous everything

e) Almost never blocks

**7) What are the two types of API functions in Node.js ?**

The two types of API functions in Node.js are

a) Asynchronous, non-blocking functions

b) Synchronous, blocking functions

**8) What is control flow function?**

A generic piece of code which runs in between several asynchronous function calls is known as control flow function.

**9) Explain the steps how “Control Flow” controls the functions calls?**

a) Control the order of execution

b) Collect data

c) Limit concurrency

d) Call the next step in program

**10) Why Node.js is single threaded**?

For async processing, Node.js was created explicitly as an experiment. It is believed that more performance and scalability can be achieved by doing async processing on a single thread under typical web loads than the typical thread based implementation.

**11) Does node run on windows?**

Yes – it does. Download the MSI installer from <http://nodejs.org/download/>

**12) Can you access DOM in node?**

No, you cannot access DOM in node.

**13) Using the event loop what are the tasks that should be done asynchronously?**

a) I/O operations

b) Heavy computation

c) Anything requiring blocking

**14) Why node.js is quickly gaining attention from JAVA programmers?**

Node.js is quickly gaining attention as it is a loop based server for JavaScript. Node.js gives user the ability to write the JavaScript on the server, which has access to things like HTTP stack, file I/O, TCP and databases.

**15) What are the two arguments that async.queue takes?**

The two arguments that async.queue takes

a) Task function

b) Concurrency value

**16) What is an event loop in Node.js ?**

To process and handle external events and to convert them into callback invocations an event loop is used. So, at I/O calls, node.js can switch from one request to another .

**17) Mention the steps by which you can async in Node.js?**

By following steps you can async Node.js

a) First class functions

b) Function composition

c) Callback Counters

d) Event loops

**18) What are the pros and cons of Node.js?**

**Pros:**

a) If your application does not have any CPU intensive computation, you can build it in Javascript top to bottom, even down to the database level if you use JSON storage object DB like MongoDB.

b) Crawlers receive a full-rendered HTML response, which is far more SEO friendly rather than a single page application or a websockets app run on top of Node.js.

**Cons:**

a) Any intensive CPU computation will block node.js responsiveness, so a threaded platform is a better approach.

b) Using relational database with Node.js is considered less favourable

**19) How Node.js overcomes the problem of blocking of I/O operations?**

Node.js solves this problem by putting the event based model at its core, using an event loop instead of threads.

**20) What is the difference between Node.js vs Ajax?**

The difference between Node.js and Ajax is that, Ajax (short for Asynchronous Javascript and XML) is a client side technology, often used for updating the contents of the page without refreshing it. While,Node.js is Server Side Javascript, used for developing server software. Node.js does not execute in the browser but by the server.

**21) What are the Challenges with Node.js ?**

Emphasizing on the technical side, it’s a bit of challenge in Node.js to have one process with one thread to scale up on multi core server.

**22)**  **What does it mean “non-blocking” in node.js?**

In node.js “non-blocking” means that its IO is non-blocking. Node uses “libuv” to handle its IO in a platform-agnostic way. On windows, it uses completion ports for unix it uses epoll or kqueue etc. So, it makes a non-blocking request and upon a request, it queues it within the event loop which call the JavaScript ‘callback’ on the main JavaScript thread.

**23) What is the command that is used in node.js to import external libraries?**

Command “require” is used for importing external libraries, for example, “var http=require (“http”)”. This will load the http library and the single exported object through the http variable.

**24) Mention the framework most commonly used in node.js?**

“Express” is the most common framework used in node.js

**25) What is ‘Callback’ in node.js?**

Callback function is used in node.js to deal with multiple requests made to the server. Like if you have a large file which is going to take a long time for a server to read and if you don’t want a server to get engage in reading that large file while dealing with other requests, call back function is used. Call back function allows the server to deal with pending request first and call a function when it is finished.

---------

**What tools and IDEs are used for Node.js?**

► Atom (free open-source)<br />  
► Nodeclipse Enide Studio (free open-source, Eclipse-based)<br />  
► JetBrains WebStorm (commercial)<br />  
► JetBrains IntelliJ IDEA (commercial)<br />  
► Microsoft Visual Studio with TypeScript<br />  
► NoFlo – flow-based programming environment integrated with GNOME APIs

**How to get started with Node.js**

First, learn the core concepts of Node.js:

You'll want to understand the asynchronous coding style that Node encourages.

Async != concurrent. Understand Node's event loop!

Node uses CommonJS-style require() for code loading; it's probably a bit different from what you're used to.

Familiarize yourself with Node's standard library.

Then, you're going to want to see what the community has to offer:

The gold standard for Node package management is NPM.

It is a command line tool for managing your project's dependencies.

Make sure you understand how Node and NPM interact with your project via the node\_modules folder and package.json.

NPM is also a registry of pretty much every Node package out there

Finally, you're going to want to know what some of the more popular packages are for various tasks:

Useful Tools for Every Project:

Underscore contains just about every core utility method you want.

CoffeeScript makes JavaScript considerably more bearable, while also keeping you out of trouble!

Caveat: A large portion of the community frowns upon it. If you are writing a library, you should consider regular JavaScript, to benefit from wider collaboration.

Unit Testing:

Mocha is a popular test framework.

Vows is a fantastic take on asynchronous testing, albeit somewhat stale.

Expresso is a more traditional unit testing framework.

node-unit is another relatively traditional unit testing framework.

Web Frameworks:

Express is by far the most popular framework.

Meteor bundles together jQuery, Handlebars, Node.js, websockets, mongoDB, and DDP and promotes convention over configuration without being a Rails clone.

Tower is an abstraction of top of Express that aims to be a Rails clone.

Geddy is another take on web frameworks.

RailwayJS is a Ruby-on-Rails inspired MVC web framework.

SailsJS is a realtime MVC web framework.

Sleek.js is a simple web framework, built upon express.js.

Hapi is a configuration-centric framework with built-in support for input validation, caching, authentication, etc.

Koa Koa is a new web framework designed by the team behind Express, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs.

Web Framework Tools:

Jade is the HAML/Slim of the Node world

EJS is a more traditional templating language.

Don't forget about Underscore's template method!

Networking:

Connect is the Rack or WSGI of the Node world.

Request is a very popular HTTP request library.

socket.io is handy for building WebSocket servers.

Command Line Interaction:

Optimist makes argument parsing a joy.

Commander is another popular argument parser.

Colors makes your CLI output pretty.

**Is Node.js on multi-core machines?**

Yes, Node.js is one-thread-per-process. This is a very deliberate design decision and eliminates the need to deal with locking semantics. If you don't agree with this, you probably don't yet realize just how insanely hard it is to debug multi-threaded code. For a deeper explanation of the Node.js process model and why it works this way (and why it will NEVER support multiple threads), read my other post.

**Tell me how to decide when to use Node.js?**

You did a great job of summarizing what's awesome about Node.js. My feeling is that Node.js is especially suited for applications where you'd like to maintain a persistent connection from the browser back to the server. Using a technique known as "long-polling", you can write an application that sends updates to the user in real time. Doing long polling on many of the web's giants, like Ruby on Rails or Django, would create immense load on the server, because each active client eats up one server process. This situation amounts to a tarpit attack. When you use something like Node.js, the server has no need of maintaining separate threads for each open connection.

**Can we use jQuery with Node.js?**

No. It's going to be quite a big effort to port a browser environment to node. Another approach, that I'm currently investigating for unit testing, is to create "Mock" version of jQuery that provides callbacks whenever a selector is called. This way you could unit test your jQuery plugins without actually having a DOM. You'll still have to test in real browsers to see if your code works in the wild, but if you discover browser specific issues, you can easily "mock" those in your unit tests as well. I'll push something to github.com/felixge once it's ready to show.

**How to extract POST data in node.js?**

If you use Express (High performance, high class web development for Node.js), you can do this:

HTML:

<form method="post" action="/">

<input type="text" name="user[name]">

<input type="text" name="user[email]">

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

</form>

Javascript:

app.use(express.bodyParser();

app.post('/', function(request, response){

console.log(request.body.user.name);

console.log(request.body.user.email);

});

This gets a lot easier if you use the request library.

var request = require('request');

request.post(

'http://www.abcsite.com/formpage',

{ form: { key: 'value' } },

function (error, response, body) {

if (!error && response.statusCode == 200) {

console.log(body)

}

}

);

Aside from providing a nice syntax it makes json requests easy, handles oauth signing (for twitter, etc.), can do multi-part forms (e.g. for uploading files) and streaming.

---------

**What is “callback hell” and how can it be avoided?**

“Callback hell” refers to heavily nested callbacks that have become unweildy or unreadable.

An example of heavily nested code is below:

query("SELECT clientId FROM clients WHERE clientName='picanteverde';", function(id){  
 query("SELECT \* FROM transactions WHERE clientId=" + id, function(transactions){  
 transactions.each(function(transac){  
 query("UPDATE transactions SET value = " + (transac.value\*0.1) + " WHERE id=" + transac.id, function(error){  
 if(!error){  
 console.log("success!!");  
 }else{  
 console.log("error");  
 }  
 });  
 });  
 });  
});

The primary method to fix callback hell is usually referred to as **modularization**. The callbacks are broken out into independent functions which can be called with some parameters. So the first level of improvement might be:

var logError = function(error){  
 if(!error){  
 console.log("success!!");  
 }else{  
 console.log("error");  
 }  
 },  
 updateTransaction = function(t){  
 query("UPDATE transactions SET value = " + (t.value\*0.1) + " WHERE id=" + t.id, logError);  
 },  
 handleTransactions = function(transactions){  
 transactions.each(updateTransaction);  
 },  
 handleClient = function(id){  
 query("SELECT \* FROM transactions WHERE clientId=" + id, handleTransactions);  
 };  
  
query("SELECT clientId FROM clients WHERE clientName='picanteverde';",handleClient);

Even though this code is much easier to read, and we created some functions that we can even reuse later, in some cases it may be appropriate to use a more robust solution in the form of **promises**. Promises allow additional desirable behavior such as error propogation and chaining. Node.js doesn’t include much core support for promises, so one of the popular promise libraries should be used. One of the most popular is the [Q promise library](https://www.npmjs.org/package/q).

More information about promises and how they work can be found [here](http://www.html5rocks.com/en/tutorials/es6/promises/).

Additionally, a more supercharged solution to callback hell is provided by **generators**, as these can resolve execution dependency between different callbacks. However, generators are much more advanced and it might be overkill to use them for this purpose. To read more about generators you can start with [this post](http://strongloop.com/strongblog/how-to-generators-node-js-yield-use-cases/).

[Comment](http://www.toptal.com/nodejs/interview-questions#)

What is the preferred method of resolving unhandled exceptions in Node.js?

Hide answer

Unhandled exceptions in Node.js can be caught at the Process level by attaching a handler for uncaughtException event.

process.on('uncaughtException', function(err) {  
 console.log('Caught exception: ' + err);  
});

However, uncaughtException is a very crude mechanism for exception handling and may be removed from Node.js in the future. An exception that has bubbled all the way up to the Process level means that your application, and Node.js may be in an undefined state, and the only sensible approach would be to restart everything.

The preferred way is to add another layer between your application and the Node.js process which is called the [domain](http://nodejs.org/api/domain.html).

Domains provide a way to handle multiple different I/O operations as a single group. So, by having your application, or part of it, running in a separate domain, you can safely handle exceptions at the domain level, before they reach the Process level.

**What is typically the first argument passed to a Node.js callback handler?**

Node.js core modules, as well as most of the community-published ones, follow a pattern whereby the first argument to any callback handler is an optional error object. If there is no error, the argument will be null or undefined.

A typical callback handler could therefore perform error handling as follows:

function callback(err, results) {  
 // usually we'll check for the error before handling results  
 if(err) {  
 // handle error somehow and return  
 }  
 // no error, perform standard callback handling  
}

**How does Node.js support multi-processor platforms, and does it fully utilize all processor resources?**

Since Node.js is by default a **single thread** application, it will run on a single processor core and will not take full advantage of multiple core resources. However, Node.js provides support for deployment on multiple-core systems, to take greater advantage of the hardware. The [Cluster](http://nodejs.org/api/cluster.html) module is one of the core Node.js modules and it allows running multiple Node.js worker processes that will share the same port.

**How does Node.js handle child threads?**

Node.js, in its essence, is a **single thread** process. It does not expose child threads and thread management methods to the developer. Technically, Node.js *does* spawn child threads for certain tasks such as asynchronous I/O, but these run behind the scenes and do not execute any application JavaScript code, nor block the main event loop.

If threading support is desired in a Node.js application, there are tools available to enable it, such as the [ChildProcess](http://nodejs.org/api/child_process.html) module.

Consider the following JavaScript code:

console.log("first");  
setTimeout(function() {  
 console.log("second");  
}, 0);  
console.log("third");

The output will be:

first  
third  
second

**Assuming that this is the desired behavior, and that we are using Node.js version 0.10 or higher, how else might we write this code?**

Node.js version 0.10 introduced setImmediate, which is equivalent to setTimeout(fn, 0), but with some slight advantages.

**setTimeout(fn, delay)** calls the given callback fn after the given delay has ellapsed (in milliseconds). However, the callback is not executed immediately at this time, but added to the function queue so that it is executed **as soon as possible**, after all the currently executing and currently queued event handlers have completed. Setting the delay to 0 adds the callback to the queue immediately so that it is executed as soon as all currently-queued functions are finished.

**setImmediate(fn)** achieves the same effect, except that it doesn’t use the queue of functions. Instead, it checks the queue of I/O event handlers. If all I/O events in the current snapshot are processed, it executes the callback. It queues them immediately after the last I/O handler somewhat like process.nextTick. This is faster than setTimeout(fn, 0).

So, the above code can be written in Node as:

console.log("first");  
setImmediate(function(){  
 console.log("second");  
});  
console.log("third");

-------------------------

**2. Can you explain how Nodejs works**

It uses Google V8 Javascript engine to execute code. It contains built-in asynchronous I/O library for file, socket and HTTP communication. Node.js encapsulates libuv to handle asynchronous events.

**3. Is Nodejs really Single-Threaded**

Node.js operates on single-thread, but using non-blocking I/O calls allows it to support many concurrent connections. That means node doen't process the requests in parallel but all the back-end stuffs which actually takes lot of time run in parallel.

**4. Can you explain the Asynchronous approach in Nodejs**

Nodejs operats asynchronously using event loop and callback functions.An Event Loop is a functionality which handles and processes all your external events and just converts them to a callback function. It invokes all your event handlers at a proper time. So, that means while executing a single request, it does a lot of things in the back-end so that the current request or the coming request doesn't take much time.

You can read more here - [Asynchronous approach in Nodejs](http://www.codingdefined.com/2014/06/nodejs-asynchronous-approach.html)

**5. Can you explain what is Globals in Nodejs**

Global, Process and Buffer are combinedly termed as Globals.

Global : Its a global namespace object

Process : Its also a global object but it provides essential functionality to transform a synchronous function into a asynchronous callback.

Buffer : Raw data is stored in instances of the Buffer class.

You can read more here - [Globals in Nodejs](http://www.codingdefined.com/2014/07/globals-in-nodejs.html)

**6. What is the Use of underscore in Nodejs?**

To access the last expression, we have to use the (\_) underscore/underline character.

You can read more here - [Use of Underscore (\_) in Nodejs](http://www.codingdefined.com/2014/06/use-of-underscore-in-nodejs.html)

**7. Can you create Http Server in Nodejs, explain with code**

Yes, we can create Http Server in Nodejs. We can use http-server command to do so.

Code :

var http = require('http');

var requestListener = function (request, response) {

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

response.end('Hello You\n');

}

var server = http.createServer(requestListener);

server.listen(8080); // The port where you want to start with.

**8. How to load HTML in Nodejs**

To load HTML in Nodejs we have to change the Content-type from text/plain to text/html.

You can read more here - [Loading HTML in NodeJS](http://www.codingdefined.com/2014/06/loading-html-in-nodejs.html)

**9. Can you explain the difference between Node.js vs Ajax**

The difference between Node.js and Ajax is that Ajax is a client side technology whereas Nodejs is server side technology. Ajax is used for updating the contents of the page without refreshing it whereas Nodejs is used for developing server software. Nodejs is executed by the server whereas Ajax is executed by the browser.

**10. Can you explain the difference between readFile vs createReadStream in Nodejs**

readFile - It will read the file completely into memory before making it available to the User.

createReadStream - It will read the file in chunks of the size which is specified before hand.

You can read more here - [readFile vs createReadStream in Nodejs](http://www.codingdefined.com/2014/07/readfile-vs-createreadstream-in-nodejs.html)

Please Like and Share the Blog, if you find it interesting and helpful.

**MEAN.js:-**

MEAN.JS is a full-stack JavaScript open-source solution, which provides a solid starting point for [MongoDB](http://www.mongodb.org/), [Node.js](http://www.nodejs.org/), [Express](http://expressjs.com/), and [AngularJS](http://angularjs.org/) based applications. The idea is to solve the common issues with connecting those frameworks, build a robust framework to support daily development needs, and help developers use better practices while working with popular JavaScript components.

## Before You Begin

Before you begin we recommend you read about the basic building blocks that assemble a MEAN.JS application:

* MongoDB - Go through [MongoDB Official Website](http://mongodb.org/) and proceed to their [Official Manual](http://docs.mongodb.org/manual/), which should help you understand NoSQL and MongoDB better.
* Express - The best way to understand express is through its [Official Website](http://expressjs.com/), which has a [Getting Started](http://expressjs.com/starter/installing.html) guide, as well as an [ExpressJS Guide](http://expressjs.com/guide/error-handling.html) guide for general express topics. You can also go through this [StackOverflow Thread](http://stackoverflow.com/questions/8144214/learning-express-for-node-js) for more resources.
* AngularJS - Angular's [Official Website](http://angularjs.org/) is a great starting point. You can also use [Thinkster Popular Guide](http://www.thinkster.io/), and the [Egghead Videos](https://egghead.io/).
* Node.js - Start by going through [Node.js Official Website](http://nodejs.org/) and this [StackOverflow Thread](http://stackoverflow.com/questions/2353818/how-do-i-get-started-with-node-js), which should get you going with the Node.js platform in no time.

## Prerequisites

Make sure you have installed all these prerequisites on your development machine.

* Node.js - [Download & Install Node.js](http://www.nodejs.org/download/) and the npm package manager, if you encounter any problems, you can also use this [GitHub Gist](https://gist.github.com/isaacs/579814) to install Node.js.
* MongoDB - [Download & Install MongoDB](http://www.mongodb.org/downloads), and make sure it's running on the default port (27017).
* Bower - You're going to use the [Bower Package Manager](http://bower.io/) to manage your front-end packages, in order to install it make sure you've installed Node.js and npm, then install bower globally using npm:

$ npm install -g bower

* Grunt - You're going to use the [Grunt Task Runner](http://gruntjs.com/) to automate your development process, in order to install it make sure you've installed Node.js and npm, then install grunt globally using npm:

$ sudo npm install -g grunt-cli

## Downloading MEAN.JS

There are several ways you can get the MEAN.JS boilerplate:

### Yo Generator

The recommended way would be to use the [Official Yo Generator](http://meanjs.org/generator.html) which will generate the latest stable copy of the MEAN.JS boilerplate and supplies multiple sub-generators to ease your daily development cycles.

### Cloning The GitHub Repository

You can also use Git to directly clone the MEAN.JS repository:

$ git clone https://github.com/meanjs/mean.git meanjs

This will clone the latest version of the MEAN.JS repository to a **meanjs** folder.

### Downloading The Repository Zip File

Another way to use the MEAN.JS boilerplate is to download a zip copy from the [master branch on GitHub](https://github.com/meanjs/mean/archive/master.zip). You can also do this using wget command:

$ wget https://github.com/meanjs/mean/archive/master.zip -O meanjs.zip; unzip meanjs.zip; rm meanjs.zip

Don't forget to rename **mean-master** after your project name.

## Quick Install

Once you've downloaded the boilerplate and installed all the prerequisites, you're just a few steps away from starting to develop you MEAN application.

The first thing you should do is install the Node.js dependencies. The boilerplate comes pre-bundled with a package.json file that contains the list of modules you need to start your application, to learn more about the modules installed visit the NPM & Package.json section.

To install Node.js dependencies you're going to use npm again, in the application folder run this in the command-line:

$ npm install

This command does a few things:

* First it will install the dependencies needed for the application to run.
* If you're running in a development environment, it will then also install development dependencies needed for testing and running your application.
* Finally, when the install process is over, npm will initiate a bower install command to install all the front-end modules needed for the application

## Running Your Application

After the install process is over, you'll be able to run your application using Grunt, just run grunt default task:

$ grunt

Your application should run on the 3000 port so in your browser just go to<http://localhost:3000>

That's it! your application should be running by now, to proceed with your development check the other sections in this documentation. If you encounter any problem try the Troubleshooting section.

## Development and deployment With Docker

* Install [Docker](http://www.docker.com/)
* Install [Fig](https://github.com/orchardup/fig)
* Local development and testing with fig:

$ fig up

* Local development and testing with just Docker:

$ docker build -t mean .  
$ docker run -p 27017:27017 -d --name db mongo  
$ docker run -p 3000:3000 --link db:db\_1 mean  
$

* To enable live reload forward 35729 port and mount /app and /public as volumes:

$ docker run -p 3000:3000 -p 35729:35729 -v /Users/mdl/workspace/mean-stack/mean/public:/home/mean/public -v /Users/mdl/workspace/mean-stack/mean/app:/home/mean/app --link db:db\_1 mean

## Running in a secure environment

To run your application in a secure manner you'll need to use OpenSSL and generate a set of self-signed certificates. Unix-based users can use the following commnad:

$ sh generate-ssl-certs.sh

Windows users can follow instructions found [here](http://www.websense.com/support/article/kbarticle/How-to-use-OpenSSL-and-Microsoft-Certification-Authority) To generate the key and certificate and place them in the *config/sslcert* folder.

## Getting Started With MEAN.JS

You have your application running but there are a lot of stuff to understand, we recommend you'll go over the [Official Documentation](http://meanjs.org/docs.html). In the docs we'll try to explain both general concepts of MEAN components and give you some guidelines to help you improve your development process. We tried covering as many aspects as possible, and will keep update it by your request, you can also help us develop the documentation better by checking out the *gh-pages* branch of this repository.

## Community

* Use to [Offical Website](http://meanjs.org) to learn about changes and the roadmap.
* Join #meanjs on freenode.
* Discuss it in the new [Google Group](https://groups.google.com/d/forum/meanjs)
* Ping us on [Twitter](http://twitter.com/meanjsorg) and [Facebook](http://facebook.com/meanjs)