Q. What is Node.js? What is it used for?

Node.js is a run-time JavaScript environment built on top of Chrome’s V8 engine. It uses an event-driven, non-blocking I/O model. It is lightweight and so efficient. Node.js has a package ecosystem called npm.

Node.js can be used to build different types of applications such as web application, real-time chat application, REST API server etc. However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET. Node.js was developed by Ryan Dahl in 2009.

Q. Why use Node.js?

Node.js makes building scalable network programs easy. Some of its advantages include:

It is generally fast

It almost never blocks

It offers a unified programming language and data type

Everything is asynchronous

It yields great concurrency

Q. What is Event loop in Node.js work? And How does it work?

The Event loop handles all async callbacks. Node.js (or JavaScript) is a single-threaded, event-driven language. This means that we can attach listeners to events, and when a said event fires, the listener executes the callback we provided.

Whenever we are call setTimeout, http.get and fs.readFile, Node.js runs this operations and further continue to run other code without waiting for the output. When the operation is finished, it receives the output and runs our callback function.

So all the callback functions are queued in an loop, and will run one-by-one when the response has been received.

Q. How does Node.js work?

A web server using Node.js typically has a workflow that is quite similar to the diagram illustrated below. Let’s explore this flow of operations in detail.

• Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking:

• Querying for data

• Deleting data

• Updating the data

• Node.js retrieves the incoming requests and adds those to the Event Queue

• The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough to not require any external resources

• The Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients

A single thread from the Thread Pool is assigned to a single complex request. This thread is responsible for completing a particular blocking request by accessing external resources, such as computation, database, file system, etc.

Once the task is carried out completely, the response is sent to the Event Loop that in turn sends that response back to the client.

Q. What is REPL in Node.js?

REPL means Read-Eval-Print-Loop. It is a virtual environment that comes with Node.js. We can quickly test our JavaScript code in the Node.js REPL environment.

To launch the REPL in Node.js, just open the command prompt and type node. It will change the prompt to > in Windows and MAC.

Now we can type and run our JavaScript easily. For example, if we type 10 + 20, it will print 30 in the next line.

REPL stands for Read Eval Print Loop, and it represents a computer environment. It’s similar to a Windows console or Unix/Linux shell in which a command is entered. Then, the system responds with an output

Q. How would you define the term I/O?

The term I/O is used to describe any program, operation or device that transfers data to or from a medium and to or from another medium

• Every transfer is an output from one medium and an input into another. The medium can be a physical device, network, or files within a system

Q. How is Node.js most frequently used?

Node.js is widely used in the following applications:

1. Real-time chats

2. Internet of Things

3. Complex SPAs (Single-Page Applications)

4. Real-time collaboration tools

5. Streaming applications

6. Microservices architecture

Q. What is NPM?

NPM stands for Node Package Manager, which is responsible for managing all the packages and modules for Node.js.

Node Package Manager provides two main functionalities:

• Provides online repositories for node.js packages/modules, which are searchable on search.nodejs.org

• Provides command-line utility to install Node.js packages, and also manages Node.js versions and dependencies

Q. What is the purpose of module.exports in Node.js?

A module in Node.js is used to encapsulate all the related codes into a single unit of code, which can be interpreted by shifting all related functions into a single file. You can export a module using the module.exports, which allows it to be imported into another file using a required keyword

A module encapsulates related code into a single unit of code. This can be interpreted as moving all related functions into a file. Imagine that we created a file called greetings.js and it contains the following two functions:

In the above code, module.exports exposes two functions to the outer world. We can import them in another file as follow:

Q. What is Tracing in Node.js?

Tracing provides a mechanism to collect tracing information generated by V8, Node core and userspace code in a log file. Tracing can be enabled by passing the --trace-events-enabled flag when starting a Node.js application.

The set of categories for which traces are recorded can be specified using the --trace-event-categories flag followed by a list of comma separated category names. By default the node and v8 categories are enabled.

Running Node.js with tracing enabled will produce log files that can be opened in the chrome://tracing tab of Chrome.

Q. What is package.json? What is it used for?

This file holds various metadata information about 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.

Some of the fields are: name, name, description, author and dependencies.

When someone installs our project through npm, all the dependencies listed will be installed as well. Additionally, if someone runs npm install in the root directory of our project, it will install all the dependencies to ./node\_modules directory.

Q. What is EventEmitter in Node.js?

All objects that emit events are instances of the EventEmitter class. These objects expose an eventEmitter.on() function that allows one or more functions to be attached to named events emitted by the object.

When the EventEmitter object emits an event, all of the functions attached to that specific event are called synchronously.

Q. What are the two types of API functions in Node.js?

The two types of API functions in Node.js are:

• Asynchronous, non-blocking functions

• Synchronous, blocking functions

Q. What is the package.json file?

The package.json file is the heart of a Node.js system. This file holds the metadata for a particular project. The package.json file is found in the root directory of any Node application or module

This is what a package.json file looks like immediately after creating a Node.js project using the command: npm init

You can edit the parameters when you create a Node.js project.

Q. What are streams in Node.js?

Streams are objects that enable you to read data or write data continuously.

There are four types of streams:

Readable – Used for reading operations

Writable − Used for write operations

Duplex − Can be used for both reading and write operations

Transform − A type of duplex stream where the output is computed based on input

Streams are pipes that let you easily read data from a source and pipe it to a destination. Simply put, a stream is nothing but an EventEmitter and implements some specials methods. Depending on the methods implemented, a stream becomes Readable, Writable, or Duplex (both readable and writable).

Q. What is a callback function in Node.js?

A callback is a function called at the completion of a given task. This prevents any blocking and enables other code to run in the meantime.

Node.js, being an asynchronous platform, doesn’t wait around for things like file I/O to finish — Node.js uses callbacks. A callback is a function called at the completion of a given task; this prevents any blocking, and allows other code to be run in the meantime.

Callbacks are the foundation of Node.js. Callbacks give us an interface with which to say, “and when you’re done doing that, do all this.” This allows us to have as many IO operations as our OS can handle happening at the same time. For example, in a web server with hundreds or thousands of pending requests with multiple blocking queries, performing the blocking queries asynchronously gives you the ability to be able to continue working and not just sit still and wait until the blocking operations come back.

Q. What is the control flow function?

The control flow function is a piece of code that runs in between several asynchronous function calls.

Q. What is the buffer class in Node.js?

Buffer class stores raw data similar to an array of integers, but corresponds to a raw memory allocation outside the V8 heap. Buffer class is used because pure JavaScript is not compatible with binary data

Q. What is piping in Node.js?

Piping is a mechanism used to connect the output of one stream to another stream. It is normally used to retrieve data from one stream, and pass output to another stream

Q. What are some of the flags used in the read/write operations in files?

Q. What is callback hell?

Callback hell, also known as the pyramid of doom, is the result of intensively nested, unreadable, and unmanageable callbacks, which in turn makes the code harder to read and debug

• improper implementation of the asynchronous logic causes callback hell

Q. What is a reactor pattern in Node.js?

A reactor pattern is a concept of non-blocking I/O operations. This pattern provides a handler that is associated with each I/O operation. As soon as an I/O request is generated, it is then submitted to a demultiplexer

Q. Describe Node.js exit codes.

Q. Explain the concept of middleware in Node.js.

Middleware is a function that receives the request and response objects. Most tasks that the middleware functions perform are:

• Execute any type of code

• Update or modify the request and the response objects

• Finish the request-response cycle

• Invoke the next middleware in the stack

Q. What are the different types of HTTP requests?

Http defines a set of request methods used to perform desired actions. The request methods include:

GET: Used to retrieve the data

POST: Generally used to make a change in state or reactions on the server

HEAD: Similar to the GET method, but asks for the response without the response body

DELETE: Used to delete the predetermined resource

Q. What is the difference between readFile vs createReadStream in Node.js?

readFile — is for asynchronously reads the entire contents of a file. It will read the file completely into memory before making it available to the User. readFileSync is synchronous version of readFile.

createReadStream — It will read the file in chunks of the default size 64 kb which is specified before hand.

Q. What is the use of DNS module in Node.js?

dns module which provide underlying system's name resolution and DNS look up facilities. DNS module consists of an asynchronous network wrapper.

The most commonly used functions in DNS module are:

• dns.lookup(adress, options, callback) - The dns lookup method takes any website address as its first parameter and returns the corresponding first IPV4 or IPV6 record. The options parameter can be an integer or object. If no options are provided both IPV4 and IPV6 are valid inputs. The third parameter is the callback functions.

• dns.lookupservice(address, port, callback) - This function converts any physical address such as “www.knowledgehills.com” to array of record types. The record types are specified by the second parameter “rrbyte”. Finally the third method is the callback function.

• dns.getServers() - This function returns an array of IP address strings, formatted according to rfc5952, that are currently configured for DNS resolution. A string will include a port section if a custom port is used.

• dns.setServers() - This function sets the IP address and port of servers to be used when performing DNS resolution. The dns.setServers() method must not be called while a DNS query is in progress.

Q. What are the security mechanisms available in Node.js?

We can secure our Node.js application in the following ways:

Authentication — Authentication is one of the primary security stages at which user is identified as permitted to access the application at all. Authentication verifies the user’s identity through one or several checks. In Node.js, authentication can be either session-based or token-based. In session-based authentication, the user’s credentials are compared to the user account stored on the server and, in the event of successful validation, a session is started for the user. Whenever the session expires, the user needs to log in again. In token-based authentication, the user’s credentials are applied to generate a string called a token which is then associated with the user’s requests to the server.

Error Handling — Usually, the error message contains the explanation of what’s actually gone wrong for the user to understand the reason. At the same time, when the error is related to the application code syntax, it can be set to display the entire log content on the frontend. For an experienced hacker, the log content can reveal a lot of sensitive internal information about the application code structure and tools used within the software.

Request Validation — Another aspect which has to be considered, while building a secure Node.js application, is a validation of requests or, in other words, a check of the incoming data for possible inconsistencies. It may seem that invalid requests do not directly affect the security of a Node.js application, however, they may influence its performance and robustness. Validating the incoming data types and formats and rejecting requests not conforming to the set rules can be an additional measure of securing your Node.js application.

Node.js Security Tools and Best Practices — We can use tools like helmet (protects our application by setting HTTP headers), csurf (validates tokens in incoming requests and rejects the invalid ones), node rate limiter (controls the rate of repeated requests. This function can protect you from brute force attacks) and cors (enables cross-origin resource sharing).

Q. What is blocking and nonblocking API in Node.js and how does it affect the

architecture of your applications?

Blocking is when the execution of JavaScript in the Node.js process must wait until a non-JavaScript operation completes. This happens because the event loop is unable to continue running JavaScript while a blocking operation is occurring. Synchronous methods in the Node.js standard library that use libuv are the most commonly used blocking operations.

To simply put, Blocking methods execute synchronously and non-blocking methods execute asynchronously.

The following example reads a file from File System synchronously.

const fs = require('fs');

const data = fs.readFileSync('/myfile.txt'); // blocks here until file is read

Here is the same operation but reads the file asynchronously.

const fs = require('fs');

fs.readFile('/myfile.txt', (err, data) => {

if (err) throw err;

});

Althought the first example is simpler than the second, it has a disadvantage of blocking the execution of additional JavaScript until the entire file is read. Also if an error is thrown it will need to be caught or the process will crash. In the asynchronous version, it is up to us to decide whether an error should throw.

Q. Why do you use forever with Node.js ?

forever is a node.js package that is used to keep the server alive even when the server crash/stops. When the node server is stopped because of some error, exception, etc. forever automatically restarts it.

Forever can be used as

forever start app.js

Finally, the purpose of forever is to keep a child process (such as our node.js web server) running continuously and automatically restart it when it exits unexpectedly. forever has some default options and configuration convention as follow:

• forever keeps track of running processes in \*.fvr files that are placed in /tmp/forever/pids

• Each forever process will generate a unique log file placed in /tmp/forever/\*.log

• Unless otherwise specified, the output of the child process' stdout and stderr will be written to the above log file.

Q. What are the main advantages and disadvantages of using Node.js?

Some of the advantages of Node.js are:

• Easy to Learn: As JavaScript is one of the most popular programming languages for front-end development, nearly every front-end developer is familiar with this universal language. Therefore, it is much easier to switch to using Node.js at the back-end.

• More Freedom: Unlike some backend like Ruby on Rails which is a framework that imposes rules and guidelines of developing software in a particular way, Node.js gives much more space and freedom in doing it our own way. Node.js is completely unopinionated, meaning we can start building everything from scratch.

• Simultaneous Request Handling: Node.js provides the non-blocking IO system that enable us process numerous requests concurrently. The system makes simultaneous request handling much better than in other languages like Ruby or Python.

• Rich Community: The Node.js developers community is a very active and vibrant group of developers who contribute to constant improvement of Node.js.

Some of the disadvantages are:

• Unstable API and Inconsistencies: One of the biggest disadvantages of Node.js is that it lacks consistency. Node.js’ API changes frequently, and the changes are often backward-incompatible.

• Not Suitable for Heavy-Computing Apps: Node.js doesn’t support multi-threaded programming yet. It is able to serve way more complicated applications than Ruby, but it’s not suitable for performing long-running calculations.

• Event Driven Model: Event driven model will confuse a lot of programmers who are new to JavaScript. The callback chain can get very long which makes it harder to maintain.

Q. What are the types of applications that can be built by using Node.js?

• If we need an application should be realtime, then Node.js is best suitable. Because it have the best supporting library Socket.IO based on websockets.

• If we need a Single Page Application, Node.js is definite go here. Because MEAN stack is most popular stack for single page applications.

• When we want to stream the data, then go with node.js, because it's good at handling i/o.

• When we want our project to be a single language from back end to front end. Then go with node.js, its completely in JavaScript.

Q. Why Node.js is based on single threaded architecture?

Node.js was created explicitly as an experiment in async processing. The theory was that doing async processing on a single thread could provide more performance and scalability under typical web loads than the typical thread-based implementation.

A node.js app that isn't doing CPU intensive stuff can run thousands more concurrent connections than Apache or IIS or other thread-based servers.

Also, V8 is a single threaded execution engine. It's built to run exactly one thread per JavaScript execution context. We can run two V8 engines in the same process, but they won't share any variables like real threads.

Q. What are the different types of APIs available in Node.js?

Node.js provides lot of API out of the box. Some of them are:

• File System API or fs for interacting with the file system.

• http API for using HTTP server.

• child\_process API for spawning child processes.

• crypto module provides cryptographic functionality

• events for event-driven programming.

• os module provides a number of operating system-related utility methods.

• v8 module exposes APIs that are specific to the version of V8 built into the Node.js binary.

Q. How will you convert a Buffer to JSON in Node.js?

Nodejs and browser based JavaScript differ because Node has a way to handle binary data even before the ES6 draft came up with ArrayBuffer. In Node, Buffer class is the primary data structure used with most I/O operations. It is a raw binary data that is allocated outside the V8 heap and once allocated, cannot be resized.

Buffers can be converted into JSON as follow.

let bufferOne = Buffer.from('This is a buffer example.');

console.log(bufferOne);

// Output:

let json = JSON.stringify(bufferOne);

console.log(json);

// Output: {"type": "Buffer", "data": [84,104,105,115,32,105,115,32,97,32,98,117,102,102,101,114,32,101,120,97,109,112,108,101,46]}

To convert a JSON into Buffer:

let bufferOriginal = Buffer.from(JSON.parse(json).data);

console.log(bufferOriginal);

Q. Why do we use \_\_filename in Node.js?

The \_\_filename is one of the Global objects in Node.js and represents the filename of the code being executed. This is the resolved absolute path of this code file.

Based on the location of your program, it will return the file name as follows:

//main.js

console.log( \_\_filename );

/web/com/1427091028\_21099/main.js

Node.js also has another Global object to get the name of the directory that the currently executing script resides in called \_\_dirname.

Q. What are the important APIs in Timers module in Node.js?

The Timers API are as follows:

setTimeout - setTimeout() can be used to schedule code execution after a designated amount of milliseconds. This function is similar to window.setTimeout() from the browser JavaScript API, however a string of code cannot be passed to be executed. setTimeout() accepts a function to execute as its first argument and the millisecond delay defined as a number as the second argument. Additional arguments may also be included and these will be passed on to the function.

function welcome(name) {

console.log("Hello " + name);

}

setTimeout(welcome, 1500, 'World!');

The above function myFunc() will execute as close to 1500 milliseconds (or 1.5 seconds) as possible due to the call of setTimeout().

setImmediate - setImmediate() will execute code at the end of the current event loop cycle. This code will execute after any I/O operations in the current event loop and before any timers scheduled for the next event loop. This code execution could be thought of as happening "right after this", meaning any code following the setImmediate() function call will execute before the setImmediate() function argument.

console.log('before immediate');

setImmediate((arg) => {

console.log("executing immediate: " + arg);

}, 'in setImmediate');

console.log('after immediate');

// before immediate

// after immediate

// executing immediate: so immediate

setInterval - If there is a block of code that should execute multiple times, setInterval() can be used to execute that code. setInterval() takes a function argument that will run an infinite number of times with a given millisecond delay as the second argument.

function intervalFunc() {

console.log('Cant stop me now!');

}

setInterval(intervalFunc, 1500);

setTimeout(), setImmediate(), and setInterval() return a timer object that can be used to reference the set Timeout or Immediate object. By passing said object into the respective clear function, execution of that object will be halted completely. The respective functions are clearTimeout(), clearImmediate(), and clearInterval().

http://blog.vigowebs.com/post/nodejs-interview-questions-part-4

Event driven programming follows mainly a publish-subscribe pattern.

function addToCart(productId){

event.send("cart.add", {id: productId});

}

event.on("cart.add", function(event){

show("Adding product " + event.id);

});

3. What is Event loop in Node.js work? And How does it work?

How does Node.js work?

A web server using Node.js typically has a workflow that is quite similar to the diagram illustrated below. Let’s explore this flow of operations in detail.

https://www.simplilearn.com/ice9/free\_resources\_article\_thumb/Node.js\_Architecture\_Workflow.png

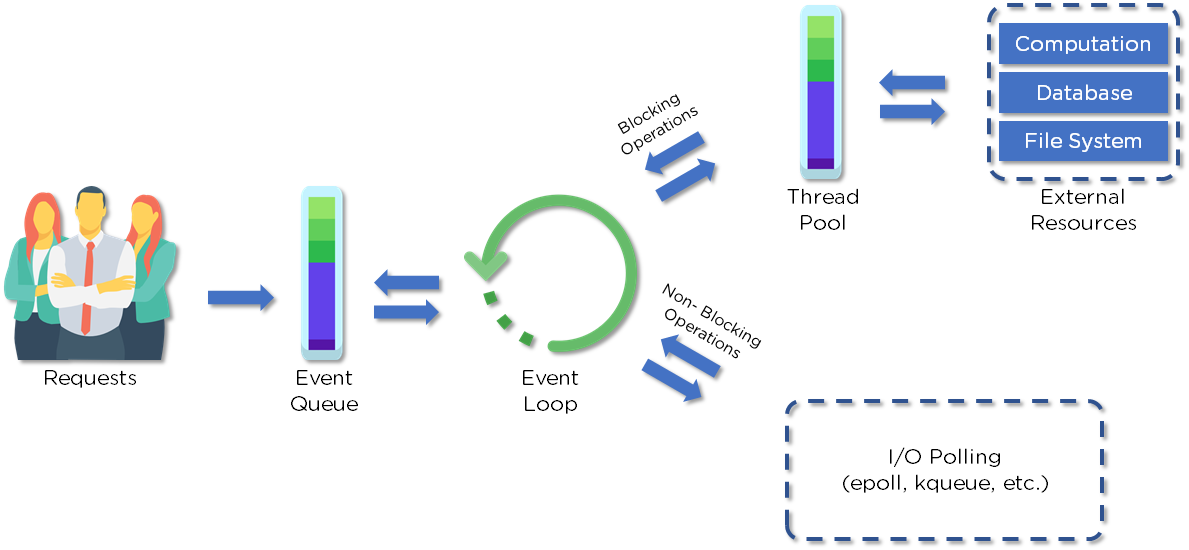
• Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking:

• Querying for data

• Deleting data

### How does Node.js work?

A web server using Node.js typically has a workflow that is quite similar to the diagram illustrated below. Let’s explore this flow of operations in detail.



* Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking:
* Querying for data
* Deleting data
* Updating the data
* Node.js retrieves the incoming requests and adds those to the Event Queue
* The requests are then passed one-by-one thr