**WPT - MN Node Js**

**What is Node.js?**

* Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.
* Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine. V8 is Google’s open source high-performance JavaScript and WebAssembly engine, written in C+) for easily building fast and scalable network applications.
* Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
* **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data.
* **Very Fast** – due to V8
* **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.
* It is not advisable to use Node.js for CPU intensive applications.

|  |  |  |
| --- | --- | --- |
|  | **Javascript** | **NodeJS** |
| 1. | Javascript is a programming language that is used for writing scripts on the website. | NodeJS is a Javascript runtime environment. |
| 2. | Javascript can only be run in the browsers. | We can run Javascript outside the browser with the help of NodeJS. |
| 3. | It is basically used on the client-side. | It is mostly used on the server-side. |
| 4. | Javascript is capable enough to add HTML and play with the DOM. | Nodejs does not have capability to add HTML tags. |

**ECMAScript 2015 (ES6)**

* ECMAScript 2015 was the second major revision to JavaScript.
* ECMAScript 2015 is also known as ES6 and ECMAScript 6.
* New Features in ES6
  + The let keyword
  + The const keyword
  + Arrow Functions
  + For/of: for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more. *for (variable of iterable) { }*
  + Map Objects
  + Set Objects
  + Classes
  + Promises: A Promise is a JavaScript object that links "Producing Code" and "Consuming Code". *const myPromise = new Promise(function(myResolve, myReject) { }*
  + Default Parameter Values: *function myFunction(x, y = 10) { }*

**What is Node JS REPL?**

The Node. js ***Read-Eval-Print-Loop (REPL)*** is an interactive shell that processes Node. js expressions. The shell reads JavaScript code the user enters, evaluates the result of interpreting the line of code, prints the result to the user, and loops until the user signals to quit. (in terminal how every line is interpreted and result printed immediately and indefinite wait for next command)

**Introduction to Asynchronous programming and callbacks**

JavaScript is synchronous by default and is single threaded. This means that code cannot create new threads and run in parallel. But JavaScript was born inside the browser, its main job, in the beginning, was to respond to user actions, like onClick, onMouseOver, onChange, onSubmit and so on.

You can't know when a user is going to click a button. So, you define an event handler for the click event. This event handler accepts a function, which will be called when the event is triggered

A callback is a simple function that's passed as a value to another function, and will only be executed when the event happens. We can do this because JavaScript has first-class functions (when functions in that language are treated like any other variable.), which can be assigned to variables and passed around to other functions (called higher-order functions)

**Promises and async & await:**

Promise is used to keep track of whether the asynchronous event has been executed or not and determines what happens after the event has occurred. It is an object having 3 states namely pending, resolved, rejected.

Async/Await is used to work with promises in asynchronous functions. It is just a wrapper to restyle code and make promises easier to read and use. It makes asynchronous code look more like synchronous/procedural code, which is easier to understand. Await can only be used in async functions. It is used for calling an async function and waits for it to resolve or reject. Await blocks the execution of the code within the async function in which it is located.

**Event-loop**

The event loop allows Node.js to perform non-blocking I/O operations even though JavaScript is single-threaded. It is done by assigning operations to the operating system whenever and wherever possible. Most operating systems are multi-threaded. When one of these operations is completed, the kernel tells Node.js, and the respective callback assigned to that operation is added to the event queue which will eventually be executed. When using Node.js, a special library module called ***libuv*** is used to perform async operations.

* Event loop is an endless loop, which waits for tasks, executes them, and then sleeps until it receives more tasks.
* The event loop executes tasks from the event queue only when the call stack is empty i.e. there is no ongoing task.
* The event loop allows us to use callbacks and promises.
* The event loop executes the tasks starting from the oldest first.

**Timer in JS**

In JavaScript, a timer is created to execute a task or any function at a particular time i.e. to delay the execution of the program or to execute the JavaScript code in a regular time interval. setTimeout(function, milliseconds) , setInterval(function, milliseconds) (executes the specified function repeatedly after a time interval.)

**Understanding Node modules, exports, and require**

In Node.js, Modules are the blocks of encapsulated code that communicates with an external application based on their related functionality. Modules can be a single file or a collection of multiples files/folders. The reason programmers are heavily reliant on modules is because of their re-usability as well as the ability to break down a complex piece of code into manageable chunks. Three types:

1. **Core Modules:** in-built modules like, http, url, os, fs (file system). These modules can be loaded into the program by using the ***require function.***
2. **Local Modules**: created by user. The functions inside the file should be marked as ***exports*** eg. *exports.add=function(int x. int y){return x+y;};*  Since this file provides attributes to the outer world via exports, another file can use its exported functionality using the *require(./file\_name)*
3. **Third-party Modules:** are available online using the ***Node Package Manager(NPM).*** These modules can be installed in the project folder or globally. Eg express, react, etc.

**Introduction to npm (Node Package Manager)**

* npm is the world's largest Software Registry. npm is free to use.
* The registry contains over 800,000 code packages.
* Open-source developers use npm to share software.
* Many organizations also use npm to manage private development.
* npm is installed with Node.js
* can manage dependencies. (in one command line) install all the dependencies of a project.

**package.json and package-lock.json files**

As you initialize your node application *(>npm init),* you will see three files installed in your app that is node\_modules, package.json, and package.lock.json.

* **package.json file** contains metadata about the project and also the functional dependencies that is required by the application.
* **package.lock.json** is created for locking the dependency with the installed version. It will install the exact latest version of that package in your application and save it in package.json. Without it, there might be some differences in installed versions in different environments.

**Install, update, and manage package dependencies**

**Install:** *>npm install package\_name*  NPM creates a folder named "node\_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

**Update:** *npm update -g <package\_name>*  To update a single global package (npm update for all)

**Managing package dependencies**: You can check which packages are out of date with the command *yarn outdated* or *npm outdated*. The npm modules defined in a package.json file are downloaded and installed into the application’s node\_modules folder when you run the npm install command.

**Local and global packages:**

**Local packages** are installed in the directory where you run *npm install <package-name>*, and they are put in the node\_modules folder under this directory. Different applications can have different versions of the package if installed locally.

**Global packages** are all put in a single place in your system (exactly where depends on your setup), regardless of where you run *npm install -g <package-name>* Updating a global package would make all your projects use the new release.

**Node.js Modules – fs and http:**

**File I/O – Sync & Async Methods**

The fs module provides a lot of very useful functionality to access and interact with the file system. One peculiar thing about the fs module is that all the methods are asynchronous by default, but they can also work synchronously by appending Sync. Eg. *fs.rename(), fs.renameSync()*.

* Read Files: fs.readFile()
* Create Files: fs.appendFile(), fs.open(), fs.writeFile()
* Update Files: fs.appendFile(), fs.writeFile()
* Delete Files: fs.unlink()
* fs.copyFile(), fs.createReadStream(), fs.createWriteStream(), fs.mkdir(), fs.rename()

**HTTP Module – Building an HTTP server**

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the HTTP. The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client using createServer() method

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

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

*res.write('Hello World!'+req.url); //write a response to the client*

*res.end(); //end the response*

*}).listen(8080);*

**Developing a Node web application**

***Step 1 - Import Required Module:*** *var http = require("http");*

***Step 2 - Create Server:*** We use the created http instance and call http.createServer() method to create a server instance and then we bind it at port 8081 using the listen method associated with the server instance. Pass it a function with parameters request and response.

***Step 3 - Testing Request & Response:*** Let's put step 1 and 2 together in a file called main.js and start our HTTP server. Now execute the main.js to start the server *$ node main.js*

***Make a Request to the Node.js Server:*** Open http://localhost:8080/ in any browser