***- Backend is all about serving files to the clients from server***

**Node.js**

**Introduction**

- Until Node.js introduced , javascript was only a client-side language and it was executed only on Browsers

- Node.js is a javascript Runtime Environment built on **Chrome’s V8 Javascript engine** , which lets Javascript to run on server-side as well .

- V8 is Google’s open source high-performance JS engine written in C++ , used in chrome and node.js

- V8 is the thing that takes out JS and executes on Chrome or It provides the run-time environment for JS

- Runtime Environment means , the environment in which a program or application is executed . Ex:- For JS back then browsers were only the runtime , but now local system too has become runtime for JS .

- Node.js is designed to built scalable network applications

- It is free and open source server environment

- It can run on multiple OS

- It allows us to run JS on the server or outside of the browser .

**Why use Node.js**

- By Node.js , we can use JS in the entire stack

- Many famous companies use it as their backend

- It comes with lots of useful built-in modules

- It is very fast, lightweight and efficient

**REPL(Read-Evaluate-Print-Loop)**

- REPL is an interactive shell that processes Node.js expressions and it is activated in terminal on writing ‘node’ command

- In REPL underscore or ‘**\_**’ holds the previous variable

- On hitting tab bar twice in REPL all the global variables/modules will be printed and these variables are always available to use and need not to be imported in a node.js file

**Modules/Packages**

- Basically , module is blocks of encapsulated codes written by other coders and can be used by any user simply by Importing them in our file , in order to achieve any functionality .

- In node.js , Modules are the blocks of encapsulated code that communicates with an external application on the basis of their functionality .

- Modules can be a single file or can be a collection of multiple files , these files consist methods , classes and other relevant stuffs .

**Types of modules in Node.js**

1 . Core modules / built-in modules

- These are the built-in modules that are the part of the platform and comes with Node.js installation .

- These modules can be loaded into the program by using th require () method .

ex: - http - creates an HTTP server in Node.js

fs - used to handle file-system

process - provides information and control about the current Node.js process .

os - provides information about the operating system. etc

2 . Local modules/User-defined modules

- Unlike built-in and external modules , local modules are creates locally by the user in the Node.js application . One can create his own local module . Ex:- Let’s create our own calc.js module

Filename: calc.js

exports.add = function (x, y) {

return x + y; };

exports.sub = function (x, y) {

return x - y; };

exports.mult = function (x, y) {

return x \* y; };

exports.div = function (x, y) {

return x / y; };

3 . Third-party Modules

- Third-party modules are modules that are available online and one can access it using the Node Package Manager(NPM)

- These modules can be installed in the project folder of globally

ex:- mongoose(npm install mongoose) , express(npm install express) , angular(npm install -g@angular/cli) and react

**const fs = require("fs");**

- It returns an object

- require(“module\_name”) is an **include-module** used to include or load JS-modules .

- require() module reads a JS-file , executes the file , and then proceeds to return the **exports object**.

- **fs** module stands for **file-system** module which allows you to work with the file system on our computer .

**const text = fs.readFileSync(“text\_file\_name” , “encoding”) -** This method is an inbuilt application programming interface of **fs** module which is used to read the file and return its content .

- utf-8(Unicode Transformation Format) is preferred encoding for e-mail and web-pages

**text = text.replace("fun","run");** -This method replace run with fun , the text file

**fs.writeFileSync("text\_new.txt",text);** - This will write the content of text into text\_new.text file , and here we don’t need to store the statement in a variable

**Synchronous(Blocking) and Asynchronou(non-blocking) code/calls**

|  |  |
| --- | --- |
| **Blocking code(Synchronous)** | **Non-Blocking code(Asynchronous)** |
| As name suggests , it means to be in a sequence ,  Every statement of code gets executed line by line | These methods execute asynchronously i.e It gets  executed line by line but not in every case . |
| Every line of code waits for its previous one to get executed first and then it gets executed | If any error occurs , these codes allow the program  to be executed further immediately without any  blocking . |
| Hence , If an error occurs in a line , the whole code  will be blocked until that error is fixed | The asynchronous functions requires a callback  method inside it as an argument to deal with error  in background letting rest of the code to get  executed. The callback will fire only if the job of the  non-blocking function is done . |
| Ex:- readFileSync(“file\_name” ,”encoding”) | Ex:- readFile(“file\_name” , “encoding” ,  callback-function) |
| The blocking counterparts of non-blocking  methods have names ending with **Sync** | All of the I/O methods in Node.js standard library  provide asynchronous versions , which are  non-blocking and accept callback functions |

Blocking code:-

const fs = require('fs');

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

console.log(data);

moreWork(); **// will run after console.log**

Non-Blocking code:-

const fs = require('fs');

fs.readFile('/file.md', **(err, data) => { //callback method**

**if (err) throw err;**

**console.log(data);**

}); **// this won’t block the program and will be running in the background until the file is read or until the error is detected**

moreWork(); **// will run before console.log**

**- Node.js works on Non-Blocking I/O model**

**Creation of simple sever which shows nav.html on server**

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

**const fs = require('fs');**

**const file = fs.readFileSync('nav.html');**

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

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

**res.end(file)**

**})**

**server.listen(80 , '127.0.0.1',()=>{**

**console.log("Listen on port 80")**

**})**

- **‘http’** is a buit-in module which allows Node.js to transfer data over the **Hyper Text Transfer Protocol(HTTP)**

- HTTP is a communications protocol which is used to send and receive webpages and files on the internet

- It is a standatrd application-level protocol used for exchanging files on the World Wide Web

- It is a client-server protocol , which means requests are initiated by the recipient , which is usually aWEB- Browsers

- Clients and Servers communicate by exchanging individual messages .

- The messages sent by the client (WEB-BROWSER) are called **HTTP-requests** and the messages sent by the server as an answer are called **HTTP-responses** .

http.**createServer(requestListener)**

- createServer method used to create a server on your computer

- It turns your computer into a HTTP server

- **http.createServer()** creates an **HTTP Server object**

- the HTTP server object can listen to the ports on your computer and execute a function **, requestlistener** , each time a request is made .

- **requestListener** : It specifies a function to be executed every time the server gets request . This function is called a requestListener , and handles request from the user , as well as reponse back to the user

- **req.url** gives the requested url from client side.

- **response.writeHead(status code , {‘Content-Type’:’text/plain’})** is an inbuilt property of the ‘http’ module which sends a response header to the request

- **response.setHeader()** allows you to set only a singular header and won’t take status code as its argument unlike writeHead() .

- **status code 200** means that all is OK , 404 is page not found

- **port** is a number used to uniquely identify a transaction over a network by specifying both the host and the services . Port is the address of the service/transaction between the client and server . POrt 80 and 443 are the two ports used by web-servers to communicate with the web-clients

- **res.end(data)** : It is used to end the response process .

- **server.listen(port , hostname , backlog, callback)** : It creates a listener(which listens the client request) on the specified port or path

- **hostname** specifies the IP address we want to listen to

- Specifies a function to be executed when the listener has been added

**Creating our own module**

index.js

const mod = require('./mod');

console.log**(mod([4,5])**); **// mod.average([4,5]) won’t work here**

mod.js (module-file)

console.log("This is my custom module");

function average(arr){

sum=0;

arr.forEach(element => {

sum += element;

});

return sum/arr.length;

}

**//** **In node.js we have to explicitly mention the methods or classes that we want to export to other files**

module.exports = average**; // we must mention this line to every method to be export**

**NPM (Node Package Manager)**

- It is the default package manager for the Node.js

- It is used to install modules , packages/modules in our project

- **npm-init** : It initializes your project-folder as a node-package and it installs package.json . By this whenever we install anything like express , mongoose , slugify etc to the project , node will include them in dependencies section of package.json.

- **package-json** : This file holds 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.

- **dependencies** : are the packages/modules on which packages.json/our-project is depenedent.

- One package is dependent on its dependencies and these dependencies can be dependent further on other multiple packages/dependencies and so on . ex: express , mongoose etc

- **devDependencies** : These are the modules which are only required during the development and not on runtime or production , whereas dependencies are required at runtime as well. Ex:- Nodemon , Babel , Chai etc

- Hence , the dependencies needs to be installed in the project while deploying it online without any devdependencies.

- **node-modules** folder is the folder where all the projects related modules/packages are kept

- we can install any specific version of a module we use **npm install** [**slugify@1.3.5**](mailto:slugify@1.3.5)

here in **1.3.5** , 1 - major version(major-changes) , 3 - minor version(small-changes) , 5-patch (bug-fixes)

“dependencies”:{ “package1”: “**^**16.5.0 “, “package2”:”**~**1.3.3” , “package”: ”**>**1.0.1”}

if any package is deleted from node-modules folder , then they can be retrieved by npm i and ,

**^** : this means exactly the given version will be installed by npm

**~** : This means npm will install the version with **newer-patch** if any available

**>** : this means if any new **major-version** is available then that will be installed / or simply the latest version will be installed

**Nodemon**

- It is a tool and a command-line interface(CLI) utility that helps to develop node.js based applications by automatically restarting the node application when the file changes in the dirtectory are detected

**npm i/install nodmon** : To install nodemon

**npm i/install -g nodemon** : To install nodemon globally on our machine(we can use nodemon in any project on our system)

**npm i/install nodemon** --save-dev : Install nodemon on your project as dev-dependency