NODE JS

1. What is NODE JS?

: a. it is an few open source server environment

b. JavaScript can be used on the server.

c. A single language for both front end and backend.

2. How to Install nodejs?

- install node js from the node js official website

Check version of node by node – version

Check version of npm by npm –version

Npm stands for node package manager

To initialize modules we will use the command npm init

It will intilize all the files from the modules

Now, how to install packages in node js ?

Packages are set of codes which someone else has written for us and we can download the same and use it in our program for example we have express its a framework.

The keyword used to install any node package is npm I followed by the package name

We don’t push the node module folder into github or any other git repo because the following folder is one heavy folder with a lot of files. We just forward the file without node modules and when someone else clones the file into there computer they can just run the command npm I and all the required dependencies which was present in my computer will be downloaded in there pc.

Now to install a package globally into a computer we can use the command npm I -g we can globally install a package into my computer so if I use one package very frequently in my computer I can globally install that package so I don’t have to install it every time I want to use that package.

Now to use that package write the name of the package followed by the file name.

What are dev dependencies?

There are some dependencies which we only want to use during the development phase of a project like nodemon so we install that dependencies as a dev dependency to do that the keyword we use is npm Install –save-dev nodemon

Now how to uninstall node modules ?

To uninstall any package the keyword is fairly simple

Its npm uninstall nodemon (followed by the name of the package )

Now we can export one module from one file to another ( create another js file and create an object in it with a string value , int value and a Boolean value )

Now we will export the file from the second.js (the second file using the keyword) module.export = user; (name of the object we want to export.

And in the first file where we want this second module we will made a constant in which we will store the value of this object

Const details = require(“./second”) in this format

Wrapper function of node is (function(exports,require, module, \_\_filename, \_\_dirname )

This is automatically implemented on a node module

If you console log the parameters of the function you will get all the details of that particular module

**Node JS documentation :-**

go to the third link after the search of node js and then go to os then use the command

const os = require('os');

to import os module it can be imported without being exported like the previous export because these are common js modules and its already present in ES6 unlike the module that we created. That is the reason (./) is not used before the import file directory.

File is a CommonJS module; it may be converted to an ES6 module.

We will go through some of the common os modules such as

console.log(os.freemem()); *// tells you the free memory in your computer*

console.log(os.homedir()); *// tells you your home directory*

console.log(os.hostname()) ; *// tells you the host name of your computer*

and another module which we will test out is going to be path

importing path will be done like

const path = require('path');

encourage students to use node documentations as a reference site because it wont be very helpful or convenient to remember all of the upcoming commands or keywords it will be tedious, time-consuming and will increase the chances of errors.

const a1 = path.posix.basename('/tmp/myfile.html');*// Returns: 'myfile.html'*

const a2 = path.posix.dirname('/tmp/myfile.html'); *// will return the directory of the file path given*

const a3 = path.posix.extname(\_\_filename); *// this command will tell me the extention name of the file i mention inside the ()*

these are the commonly used path modules but encourage students to search for more and discover them on their own.

FileSystem module :-

Another commonly used module in node js is

FileSystem module it has a lot of different modules inside of it but we will try the most common ones

To initlize the file system in the document

const fs = require('fs');

use this command

now we will use the read command to read a given txt file

fs.readFile("file.txt", 'utf8',(err,data)=> { *// i have used fs.read file so it will search the file and read its content in th console window*

    console.log(err, data);

})

In the following code I have created a file.txt file and used this command to read it now if I play with the name file.txt and use a name which dosen’t exist it will give me an error but when I have the correct the file name it will give me null as error and data will give me the content of that text file

A point to know about javascript is that when I print a file like this if I have line of codes after this like

console.log(err, data);

    console.log("all the data in the file has been read");

the output of the second console will come before the first one

the reason for this is that the first console requires the data to be loaded from another file which might require time depending on the size of the file so it dosent block the thread it lets it execute the next like and come back to it when its loaded and ready.

But some times we want a console to block the following and run before letting the next console run what would we do in that case ?

In that case we will use readFileSync

const a = fs.readFileSync("file.txt") *// this is the way we will use fs is we want it to execute it before jumping gto the next line*

console.log(a.toString());

console.log(" ok now lets give sync a try ")

this will block the code from jumping to the next line and run it before and then run the next console window.

Now lets learn about the writefile

fs.writeFile("file2.txt", "this is where i want to try the write feature of fs ", () =>{

    console.log("i have written the data inside the new file");

})

console.log("job done");

this will create a file2.txt and will add the content I have given inside but again we will face the same issue the second console will be executed before the first one so again we will use sync inside the write file module

const b = fs.writeFileSync("file2.txt", "this is where i want to try the write feature of fs using sync ") *// now it has written using the sync method in the file . txt*

console.log(b);

console.log("so now we have used sync in write method aswell");

**Common JS modules vs ECMA script modules :-**

ECMAScript modules are [the official standard format](https://tc39.github.io/ecma262/#sec-modules) to package JavaScript code for reuse. Modules are defined using a variety of [import](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import) and [export](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export) statements.

To import using common js the syntax is

const addTwo = require("./secondmodule");

*// this is the format we use to import in common javasccript*

addTwo(); *// calling the function we created in the other second module*

**to export using common js the syntax is**

function addTwo() {

    console.log(2+2);

  } *// we created a javascript function*

 module.exports = addTwo; *// this is how we export using common javascript*

now to use ES6 script module fisrt the file exporting should have an extention as .mjs instead of .js

secondly we need to add

 "type": "module",

In the file package.json

Then the syntax for import is

import  {addTwo}  from "./secondmodule.mjs";

*// this is the normal way of importing in esma script*

And the syntax to export the same module is

export function addTwo() {

    console.log(2+2);

  }; *// we created a javascript function and exported the same*

Now we can import one function as another function and the content of the first function will be stored in the next one

import  {addTwo as addNum}  from "./secondmodule.mjs"

*// this is used to store call a function as another functions*

addNum();

when the content of second modules are

export function addTwo() {

    console.log(2+2);

  }; *// we created a javascript function*

  export function addNum() {

    console.log(85+65+96);

  };

The output of the add num will be 4

We can use default export with the syntax if we are importing only one entity with a non declared name the default value will be assigned to the following for that we will need to remove the curly braces

import  addTwoThousand  from "./secondmodule.mjs"; // we are calling something that hasn’t been declared in the second file but if we call anyfunction as default like

export default function addNum() {

    console.log(85+65+96);

  };

The value of addNum will be called when we import addTwoThousand

We can also import multiple functions at once but we have to use declared names

import  {addTwo, addNum}  from "./secondmodule.mjs"

now what is import \*

import  \* as add  from "./secondmodule.mjs" *//this is used to import all the functions from the export group and store it in the function add*

**URL module:**

to import a url module we can use

const url = require('url'); *// importing the url*

and then we are going to create a constant in which we are going to store a dummy url to do that

const myURL = new URL('https://example.org:5649');

*// using this as a dummy url*

Inside the dummy url we can set paths and then we can set search status as well as hash methods to that

myURL.pathname = '/a/b/c'; *// declares the path of the url*

myURL.search = '?d=e'; *// gives me the search result*

myURL.hash = '#fgh'; *// to navigate to a section of the page*

to see the constructed url we can console . log the constant myURL.href

console.log(myURL.href); *// this will tell me excatly how my url is being cooked after adding all the required sections*

which should give me the output of

https://example.org:5649/a/b/c?d=e#fgh

**EventEmitter:-**

Event Emitter is used to create an event which is designed to perform a task but only when its being emitted it keeps on listening and dose not block the code, the code runs forward. It works synchronous when its being emitted. To import EventEmitter

const EventEmitter = require('events'); *// to import the event module*

and then we would need to create a class in which we would use event emitter module

class MyEmitter extends EventEmitter {}

*// we have created a class named MyEmitter*

The class is then stored inside a constant and turned into an object.

const myEmitter = new MyEmitter();

*//we created a constant named myEmitter and assigned the class MyEmitter to the same*

Now to create an event we will use

myEmitter.on('CookedChicken', () => {

*// we have created an event which if triggered the task present inside the*

*curly braces will happen*

  console.log('Turn off the oven the Chicken has been cooked ');

  setTimeout(() => {

      console.log("The chicken is ready pleas turn off the oven ")

  }, 3000);

  setTimeout(() => {

      console.log("Your food is going to be overcooked tonight!")

  }, 6000);

Inside the curly braces we will write the code the event should perform incase the event is emitted

And but if we run the code as of now the code will not run and not show anything as the code is asynchronous the code will move forward not executing the event but once we use

myEmitter.emit('CookedChicken');

*// to emit the event which is waiting to be listened*

The event will run synchronously and will perform the code written inside.

**HTTPServer:-**

We are creating a https server where we will create a server and push the server into a local host port .

To use https

const http = require("http");

to set the port on which the server will be sent using the network

const port = process.env.PORT || 3000;

*// we are creating the port variable and storing the port value to be 3000*

To create a server we will

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

    res.statusCode = 200;

    res.setHeader('Content-Type', 'text/html')

    res.end('<h1> This is Nishant Gupta </h1> <p> we are learning https server <p>');

}) *// creating a server in which we are giving two parameters req and responce and we have aheader as well as a paragraph*

And to listen to the server

server.listen(port, () => {

console.log(`Server is listening on port ${port}`);

}) *// we are pushing the server on the desired port*

This is how we are using the https request protocol

Now we can run the program and then go to google chrome and open the local host 3000 or whatever the port you are going to use to see the content that you wrote in the page.

We can inspect the page and go to network to see to see the status code of the transfer

We can also check out the method type which is get here

To print the req in the console window inside the create server function console .log this

 console.log(req); *// we can use this to print request type in the console window*

to see the req type and details in console window every time we run the program

now create a branches of the website by using pathways we can do that using if else conditioning

if (req.url == '/') {

        res.statusCode = 200;

        res.end('<h1> This is Nishant Gupta </h1> <p> we are learning https server <p>');

    }

In req.url we need to specify the path in which we want to print whats inside the block

Just like this we can create multiple paths using else if

And in the final else we can say

  else {

        res.statusCode = 404;

        res.end("<h1> Error 404 page not found </h1> <p> this page was not found <p/>")

    }

To like any of this if or else if a html created web page we can use fs.readfilesync which we discussed earlier

 else if (req.url == '/home')

    {   res.statusCode = 200;

        const read = fs.readFileSync("./index.html")

        res.end(read.toString());

    }

Where /index.html is the path of the html webpage that we have created

**Express JS :-**