Server Side Programming using Js

****

1. dotenv module/package
2. fs module/package (core module of node)
3. http module (core module of node)
4. mongoose module (connection mongo DB)
5. express module (Framework package for Node)
6. express router module (for routing)
7. jwt module (json web token Api-Secure)
8. multer module (image uploading)
9. body-parser package (to handle Post data send from form)

10. Template Package of Node Js.

Note :: Node is Purely

server side, programming Js Runtime Environment.

PL => JS.

Templating Means :-

laravel => Blade Template Engine

Django => gingerTechnique {% static %}

React => JSX Template

but in php we can directly implement code

in html script.

Node itself is not capable of writing the,

html script and there is not way you can interact

with dom.

hence we template Engine for Node

Node will take Js file with html code and parse

them and re-render the code, in browser.

View : React-front-End/AngularJs/Angular/VueJs/core html or Template Engine

Most Popular Template Engines:-

****

1. pug
2. HandleBars
3. Mustache
4. Ejs (Embedded Javascript)

...

...

etc

Q1: IQ => Difference EJS and Jsx

Both are using html + Js but Jsx in used in front End

and Ejs is used in Backend

Jsx => React/Angular/Vuejs

Ejs => Node/Express/Fastify

Framework of Node

****

1. Express 2. Fastify 3. Hapi
4. Koa

Model (data or schema or Api or Query)

******

let data = {

name : "Awnish",

class : "MBA",

isBack : "yes",

noOfBack: 4

}

for example:-

<h1> name = {data.name} </h1> =====> input => js code will replace => replace Output return to Browser

controller :-

****

All the Bussiness Logic will be written in,

Node Js in Controller.

controller => Js file where most important

code of Backend is Written

controller main role is get the data from model and pass to the view or vice versa.

hence Node Follows MVC Archiecture ?

IQ2 : what is MVC?

model : for database Operations.

view : all the front end Part or UI

Template (pug,ejs...)

controller : data transfer B/w M to V

Date : 10-02-2023

****

- Brief Introduction of GO4 (gang of four)
    - As your experience increase, you would see multipe way, of writing the code.

For Example:-

$() => Jquery

1. $(document).ready()

```
2. (function(){
        $('#btn').click();
});
```

3. jQuery(document).ready()

for Example:-

```
$(document).ready(function(){

})
$(document).ready(doSomething)

function doSomething(){

}

$(document).ready(()=>{



});

$(document).ready((event)=>{
```

});

$(document).ready(event=>event.preventDefault());

Programming Paradigm

****

we know as Programming paradigm(no of way you can write the code in PL,)

1. procedural Programming => limitation

2. Object Oriented Programming => limitation.

These limitation of Object Oriented Programming system

was Redesigned by 4 cs developer (Team,gang)

1990's most of IT firms were using, OOps for Software Developement and Research.

When GO4 Research Books(thesis), OOps is not the best way of Making S/W.

GO4 they re-design the oops into a set of 23 codes

"Re-usable component of Object Oriented Programming system".

There now called Design Patterns.

These are divided into categories

1. Structural DP.

2. Creational DP.

3. Behavioural DP.

These Design Pattern are there in Every Programming Langauge.

For Example:-

****

Android => ListView => ArrayAdapter, BaseAdapter,CustomAdapter ...

adapter is one of Design Pattern

****

https://github.com/EvilAnshu/design-patterns-JS/blob/master/docs.md#singletonjs

---

Nature of Node Js

****

1. Asynchronous.
2. Single Threaded.
3. Server Side Runtime Environment.
4. V8-Engine Architecture.
5. Event Driven
6. Non Blocking
7. Background I/O Mechanism
8. highly effiecent.
9. mvc Archiecture or Mvc Design pattern.

HW :- Date/10/02/2023

****

1. watch vedios on Yotube for each topic.

2. make a Notes on it with Diagrams:-

3. sunday :- React Js Theory + Practicle.

For More documentation please visit :-

****

--> https://nodejs.org/docs/latest-v17.x/api/

How to Run node Environment

****

1. console mode (output will be in cmd)

Node In console Mode :- Returns Json Output.

console of node supports :-

1. two type of command line : node chevron or node prompt, node REPL or Interactive Mode. or interactive shell.

Interactive mode is basically, interpreted in nature.

and allows only one line at a time.

REPL itself is interpreter also it is module in node.

it gives output for each input statement provided.

How to start interactive mode :-

1. open cmd

2. type node <- Hit Enter

3. > : this single greater than this is called, as Node prompt, or Node Chevron

standard Name is : Node Repl.

what is Repl?

Node : >

cmd : :/>

gitbash : $

mathlab : >>

python : >>>

laragon : lambda

xampp : #

mysql : [dbname]>

common : these all running in cmd

difference : symbol

Explaination : different symbol means different command.

Example :-

>>> git push

Error

$ print('Python is good Langauge');

Error

> echo $x;

Error

:/> console.log('hello world in node Js');

Error.

C:\>console.log('hello from node')

'console.log' is not recognized as an internal or external command,

operable program or batch file.

cmd:/> ver

Microsoft Windows [Version 10.0.19045.2486] valid Command.

These all Above Repl shows error because of invalid command is written in front

of them.

each symbol of Repl represent a unique interprator related to, PL`s (programming Langauge)

REPL : READ EVALUATE PRINT LOOP.

Read : Read input from the user

Evaluate : Solve the expression

print : generate output

Loop : we can use looping statement like, for loop while loop.

working with Node Interactive Mode

****

start Node

cmd:/> node <-hit Enter

> //node command

Keywords in node Repl

1. syntax Js

2. repl keyword :- .exit(), .help() ...

.break Sometimes you get stuck, this gets you out

.clear Alias for .break

.editor Enter editor mode

.exit Exit the REPL

.help Print this help message

.load Load JS from a file into the REPL session

.save Save all evaluated commands in this REPL session to a file

---

Process Object in Node JS

****

IQ - can you prove that node js is a runtime Environment.

Node Js contains :- process global object which holds all the environmets, path,os related information.

process.env : it is used to access Environment variable.

---

2. Script *mode or Batch Mode/shell*

---

here we want to execute bulk of lines of code we go for, script mode.

what is script here :-

1.likewise every Js file is a module itself.

2. similarly every Js file with some line of code within is a script.

p1.js

....line-1

....line-2

....line-3

....line-4

....line-5

....line-6

....line-n

js script => p1.js

execute

terminal cmd:/> node p1.js

node <filename.js>

Note :: Terminal (shell/cmd).

This script can be also, executed using npm

what is npm ? it is node package manager, and used to add dependencies on the project.

using npm script execution

****

p1.js

****

1. npm init -y
2. package.json => Edit => main => p1.js <----Entry Script.
3. scripts => key => {"start":"node p1.js"}
4. npm run start.

2. Server Mode (output will be rendered in, Browser).

in server mode Browser will work as client.

Note :: console cannot accept, Request and Response.

it does not support http protocal.

we know, in client and server model we have some algorithm like hande-shake alogrithm,

sliding window protocal, etc such type of alogrithm are not supported

by console, hence, we need a client which can support request/response

cycle, such client are, like postman, thunderbold, Browser.

=> node here will work as, Server.

node console will not as of client rather than a server.

(postman/thunderbold)[client]

Browser ---------> Request ----------> Node Server (server.js or index.js).

^

|

|

|

Response

|

|

1. to the client.

2. to the console.

Note :: console can be used as server and client both

console => client => curl

curl stands for content url.

How to create node-server

****

1. -Request Object : handle client.

2. -Response Object : handle server.

steps to create node server:-

****

server.js : most important which intialises the server.

or which set-ups server.

No Application is possible without node-server file:-

step1 :-

******

create const reference for http module:-

const http = require('http');

step2 :-

******

using http object createServer Interface.

http.createServer((request,response)=>{

});

step3:-

****

Now set a port where you want your server launch.

Never use following

port => 80 => Apache

port => 3000 => react

port => 5000 => Django

Note ::Never use reserved Port

8080 => by-default port.

7080

7000

```
const PORT=8080;

http.listen(PORT,function()=>{

  console.log('Server Started Sucessfully at port'+PORT)

});
```

How to send JSON Response to the Browser

****

1. You should have a Array of Json Object which can be

send as response to server.

2. set the content-type : application/json

3. use JSON.stringify() to encode the Json to String : Serialisation.

Note :: Why we are using JSON.stringify,

becuase of two reason

1. response.write() only takes string Input

2. Browser only Understand text or tag.

How to send html Response to the Browser

****

1. response.writeHead(200,{"Content-Type":"text/html"});

2. response.write("<h1> This is Heading </h1>");

Note here is problem :-

You cannot write lot of html code.

You can use template string and Write the web-page code

and you can then pass that variable as response to write();

Problem:-

****

Right now we are writing all data level code and design level code

in server.js

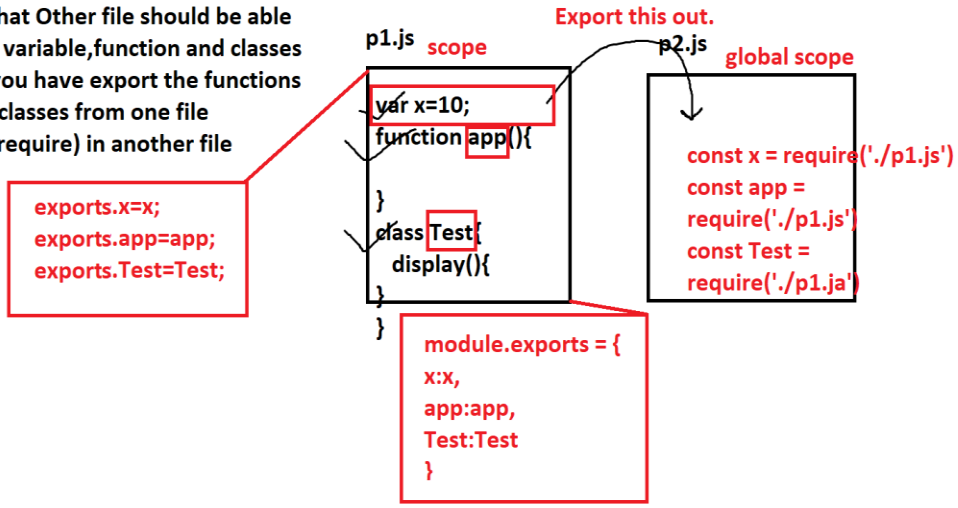
data level => model

design level => view

hence we must organise the data in mvc design pattern to follow modular approach.

Project structure of MVC Node Application

concept of modularity in Node Js
we know that,
a js file itself is a module.
if you want that Other file should be able to access the variable,function and classes access then you have export the functions variable and classes from one file and capture(require) in another file



controller

model

view

index.js or server.js

.env => configuration or Environment varaible.

.gitIgnore

package.json

package-lock.json

This project structure is common

for all, projects.

controller => folder mkdir

model => folder mkdir

view => folder mkdir

index.js => touch command

package.json => npm init -y

package-lock.json => npm install

Note if package.json

****

will install node_modules folder

if any dependencies are added in package.json

Note :: please make use of git bash

terminal.

Making a Node Module:-

****

1. module.exports = {}
2. exports.var = var;

Note :: module.exports/exports both referes to global empty Object

this => {} => global empty Object.

module = {exports : {x:10}}

module.exports = {x:10}

module.exports.x=10

var x=10;

module = {exports:x}

module.exports = x;

In flexible there is no difference B/w module.exports and exports.

but in strict mode we cannot use exports directly.

it is because module is a mendatory, Object in strict mode.

but since module refer to this Object.

you can pass varaible in following

1. module.exports = x;
2. this.exports = x;
3. exports.x=x;

|

this => module.

exports === module.exports : strict mode : off

exports != module.exports : strict mode : on

Implementation modularity in MVC

****

we know that,

m => model

v => view.

c => controller.

it is always better approach, to keep the different Js file in different associated folder.

such that modularity of the project can be maintained.

StudentModel.js => Model Suffix => pascel case.

StudentController.js => Controller suffix => pascel case.

students.js => view lowercase suffix.

Note :: Template Engine

pug : students.pug

EJS : student.ejs

jade : student.jd

handlebars : student.hbs

mustaches : mts

These template files on views are called as partials.

views

|partials

....template Engines.

|layouts

index.html, index.js

StudentModel.js

****

data of the Student model =>

Api call => student data.

```
var studentModel = {
students:[
        {id:1001,name:"Awnish", class:"Btech", stream:"cs"},
        {id:1002,name:"mohit", class:"Btech", stream:"ec"},
        {id:1003,name:"Amit", class:"BA", stream:"Hindi"},
        {id:1004,name:"Ritik", class:"Mba", stream:"it"},
        ]
```

}

module.exports = studentModel

StudentController.js

****

const studentModel = require("./StudentModel");

1. data from model to controller

Response Object it must contain

1. code : 201

2. data : it can be array or object [], {}

3. status : true or false

4. message or comment : "Login successfull","Oops something, went wrong".

5. error : by default it will be false, if any error error message will be raised

```
let response = {};

try and catch.
let promise = fetch(url).then().then.catch((error)=>{
  let responseError = error
});

JsonResponse={
  code : 404,
  message : "Runtime Exception cannot Post data",
  data : [],
  status : false,
  error : responseError
```

}

response.writeHead(200,{"Content-Type":"application/json"});

or

response.writeHead(200,{"Content-Type":"application/json;Charset=utf-8"});

response.write(JSON.stringify(JsonResponse));

Generating Pretty Json Response :-

****

by default when you will be using express module.

you will get json() method, to print the output in pretty mode.

but we are not using "express" module hence, we need to use JSON.stringify() to print the output in pretty mode.

pretty mode in JSON.stringify() :-

****

JSON.stringify({name:"awnish",class:"Btech"})

output :

{name:"awnish",class:"Btech"}

pretty Output :-

****

you need to increase padding width or pretty width

JSON.stringify(object,null,Pwidth)

Pwidth = 1,2,.....n

standard : 4

JSON.stringify({name:"awnish",class:"Btech"})

output :-

{name:"awnish", class:"Btech"}

{

name:"awnish",

class:"Btech"

}

---

fs module in Detail

****

Introduction about Fs Module

fs : fs means file system its is module and sometimes call as ready made Api in Node Js

1. module or package
2. Api

Fs module is built in module which comes ready made in node Js out of the box. (as a Additional Functionality or Add On Feature) it is core module of node Js

why core module :-
fs module is called as Api because it can used by other modules.
Note ::
its is compulsory module for some packages because they directly or indirectly use fs module as a dependency.

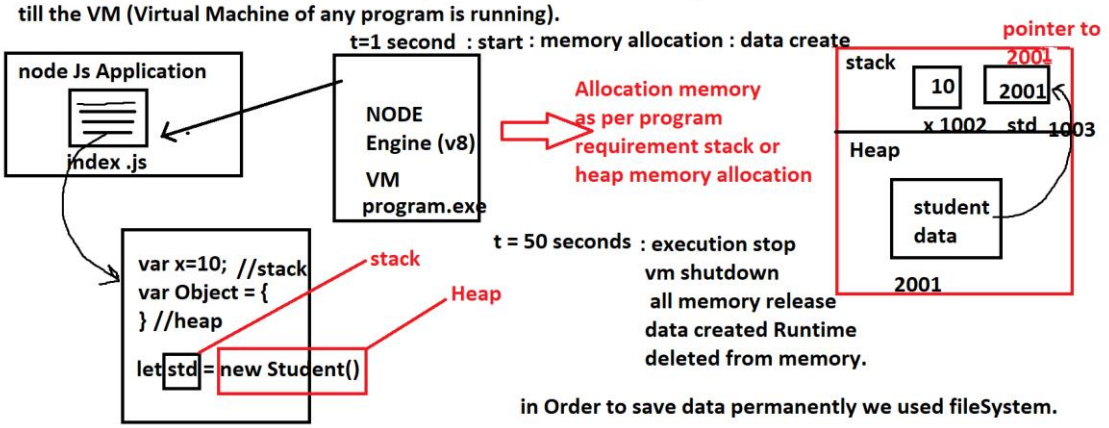
Is JS supports file handling :-
1. client side or pure Js or Vanilla Js not applicable or not supported because of security Issue.
1. using dataBuffer : data:/ or blob Object
2. FileReader
3. using fakePath from same Folder.
2. in Server side : Node Js -> Fs module

const fs = require("fs"); — **when you want some method/function of fs module**

or
require("fs") — **same when you just need to require()**

File Handling concept :-

Any Program you run, it will have memory allocation or life of the Object created will remain till the VM (Virtual Machine of any program is running).

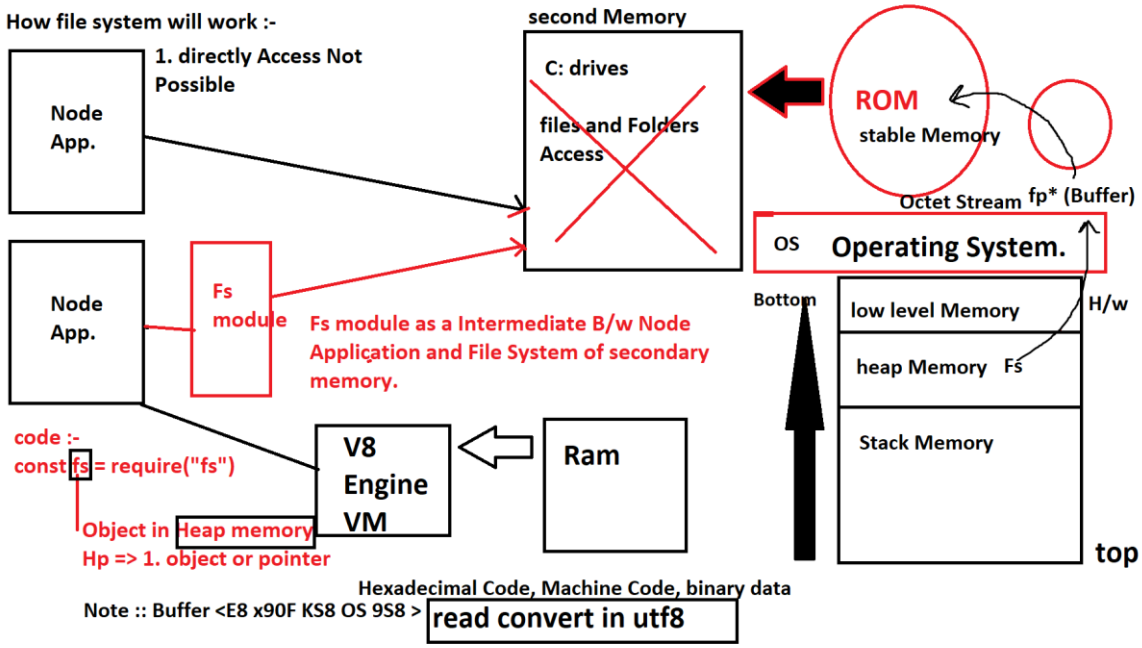


in Order to save data permanently we used fileSystem.

1. in order to save Runtime data or writing the runtime data which user creates or interacts during program execution is called as, FileHandling

why ?

for later use.



1. filesystem supports
   1. synchronous : without callback
      eg : readFileSync, writeFileSync...
   2. Asynchronous : with callback
      readFile,writeFile()...

what is difference B/w readFileSync and readFileAsync

readFileSync : function exist

readFileAsync: reason, Node Js is Asynchronous hence all module will be asynchronous in nature and we want to use asynchrounous we can use "async" keyword before any function. or callback.

2. How to read a html file asynchronously or synchronous.
   file type : .txt,html,css,json any type of file.

Important Methods in fs module.

1. readFileSync
2. readFile
3. writeFileSync
4. writeFile
5. copy method
6. unlink (delete)
7. stat
8. _dirname()
9. readdir()
10. rename()

Note :: I have taught but student did not do, tommorrow dont argue with me.

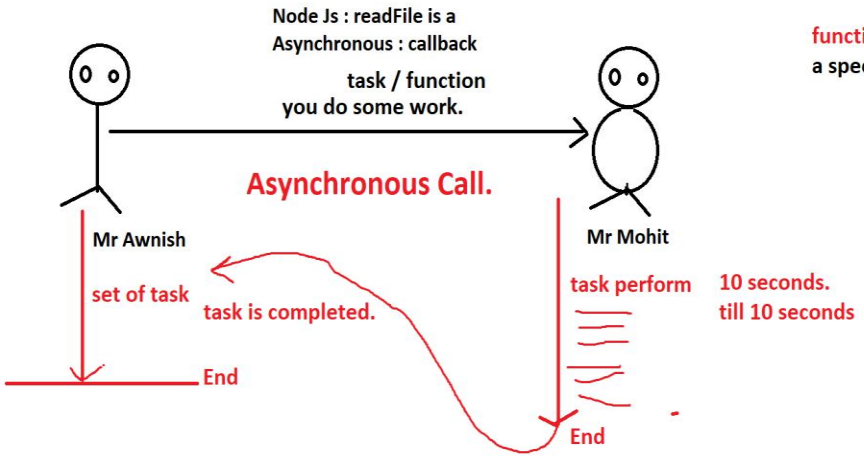
(modes of file Handling)

1. readFileASync() (readFile()) : used to read the content of the file. in binary format, or

utf8 format, by default readFileSync invokes filePointer which internally returned buffer,hence we need to encode it.

Asynchronous Reading of the file => using callback.

readFile in fs Module



function : a block of code performing a specific task is called as function.

Router Core Node.js

By default Router is core component of Express module

Router : check for endPoint Url.

url = BaseUrl + endPoint

BaseURL of Node App : http://localhost:8080 endPoint => /

http://localhost:8080/home endPoint => /home

http://localhost:8080/about endPoint => /about

http://localhost:8080/xyz endPoint =>/xyz

default case: page Not Found

Route Using Node

server.js

```
const http = require('http');
const fs = require('fs');
const PORT = 8080;


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

    let filename = '';

    switch(req.url){
      case '/':
         filename = 'index';
      break;
      case '/about':
         filename = 'about';
      break;
      default:
        filename = 'error';
        break;
    }

    res.writeHead(200,{
        "Content-Type":"text/html"
    });

    fs.readFile(`${filename}.html`,"utf-8",function(err,data){

        if(err==null){
            res.write(data);
            res.end();
        }else{
            console.log(err+'Something went wrong');
        }

    });


})

server.listen(PORT,function(){
    console.log("server Started!")
})
```

Copy using fs module

copy.js

```
const fs = require("fs");
fs.readFile("A.txt","utf8",(err,data)=>{
   if(err == null){
      //console.log(data);
      fs.writeFile("B.txt",data,"utf8",(error)=>{
        if(error == null){
            console.log('data copied from A to B');
        }
      })
   }
});
```

Delete using fs module (unlink)

unlink.js

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

fs.unlink("B.txt",(error)=>{
   console.log("File Deleted");
});
```

Move file fs module

move.js

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

let source = "A.txt";
let target = "B.txt";

fs.readFile(source,"utf8",(err1,data)=>{
   if(err1 == null){
      fs.writeFile(target,data,"utf8",(err2)=>{
        if(err2 == null){
            fs.unlink(source,(err3)=>{
               if(err3==null){
                  console.log('File Move Successfully.');
               }
            })
        }
      })
   }
});
```

Stats in fs module

stats.js

```
// stats is used to see status or statistics about file
const fs = require("fs");

fs.stat("C:/Users/anshu/Desktop/node/fs-module/copy.js",(error,fileInfo)=>{
    if(error == null){
      let sizeKb = fileInfo.size / 1024; //size convert in Bytes to KB

          fileInfo['size'] = sizeKb; //OverRide the FileInfor
          console.log('Size is '+fileInfo.size+' KB.');
          //console.log(fileInfo);
    }
})
```

Create Logger in Node

logger.js

```
const http = require("http");
const fs = require("fs");
const PORT = 8080;

const server = http.createServer((req,res)=>{
    let filename = '';
     writeLog(req.url);
     switch(req.url){
        case '/':
            filename = 'index';
        break;

        case '/home':
            filename = 'home';
        break;

        case '/about':
            filename = 'about';
        break;

        default:
            filename = 'error';
        break;
     }

     res.writeHead(200,{
        "Content-Type":"text/html"
     });

     fs.readFile(`${filename}.html`,"utf8",(err,data)=>{
          if(err ==  null){

            res.write(data);
            res.end();

          }
     })


});

server.listen(PORT,()=>{
```

```
    //console.log('Server Started in port,'+PORT);
    writeLog('Server was started at'+PORT);
})

function writeLog(action){

    let content = `[logged at,${new Date()}] = user performed ${action}.`;
    fs.appendFile("logger.log",content+"\n","utf8",(err)=>{
        if(err ==  null){
            console.log('Written data to log');
        }
    })
}
```

Add two number (cgi)

p1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="/addNumber">
        No 1 : <input type="number" name="a"> <br/>
        No 2 : <input type="number" name="b"> <br/>
        <input type="submit" value="Add">
    </form>
</body>
</html>
```

index.js

```
const http = require('http');
const PORT = 8080;
const fs = require('fs');

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

    res.writeHead(200,{"Content-Type":"text/html"});

    if(req.url == "/"){

        fs.readFile("p1.html", "utf-8", (err, data) => {
            if (err == null) {
                res.write(data);
                res.end();

            }
        });
    }else if(req.url.startsWith("/addNumber")){

        let query = req.url.split("?"); //addNumber?a=10&b=30
        let endPoint = query[0]; //a=10&b=30
        let data = query[1];
        let param1 = data.split("&"); //a=10 and b=30
        let a = param1[0].split("=")[1]; //a and 10
        let b = param1[1].split("=")[1]; //b and 30

        res.write(JSON.stringify(Number(a)+Number(b)));
        res.end();

    }

})

server.listen(PORT, () => {
    console.log("server started!")
})
```

Syntax:-

readFile("filename.txt","encoding",callbackFunction)

1. filename.txt can be any file which can be as a input source file for reading.

it can be txt,log, file or even html file.

2. encoding : By default, FileReader in Fs module, invokes or call internal File Pointer by system call :fopen inside Primary Memory, and File Pointer return the Buffer data.

hence while reading the file Content buffer data must be converted into, proper formating like utf-8, utf8, or binary format.

what is difference B/w binary encoding and utf-8 encoding:-

****

binary will not support unicode character : chinese,hindi langauge.

not read by binary langauge.

language data will only support Ascii character.

for eg : kruti dev, emoji, special symbol character, these are unicode character. hence utf-8 or utf8 both will allow any unicode character.

3. **callbackFunction** : callback tells, that function itself is Asynchronous.

syntax :-

```
function onReadingCompleted(error,data){
   if(error == null){
         let content = data;

               console.log(content); //console output.
```

```
                res.write(content); //Browser output
            }
    }
```

b) **readFileSync**:- it is a used to read File Content synchronously.
or in synchronouse mode.

syntax :-

```
const fs = require("fs");
const data = fs.readFileSync("filename.txt","encoding");
```

for Eg:-

```
const data = fs.readFileSync("input.txt","utf8");
```

Note :: Since it synchronouse it does not require any callback function.

c) **writeFileSync** : it is used to write data to the file in synchronouse mode

Syntax :-

```
const content = "This is sample content";
fs.writeFileSync("filename.txt",content,"encoding")
```

1. filename.txt : is a output file where you want content to write.
2. content : what content you want to write.
3. encoding : utf-8,utf8,binary...

Note :: if file does not exist it will create the file, since
the mode of the file is w:- (write mode)
a) since everytime file is going to generate there is less chance of
any execption.

b) write mode of the file is in overwrite mode.

Any time of times you run the code, recent data will be lossed from the file and new data will be added.

Solution :

1. readthe file content, before writing

Asynchronous reading

****

```
const content = "This is My New Data";
fs.readFile("output.txt","utf8",(error,data)=>{
  if(error == null){

                content = data+content; //oldData + newData.

                fs.writeFileSync("output.txt",content,"utf8");
  }

});
```

2. **appendFileSync**() : used to write the File in Append Mode.

what is EOF and EOL?

By default data write in a file, the it keeps on adding data in same line, "\n" and End of Line (EOL).

by default file Pointer will keep on reading the data,

till, -1 is reached. => EOF.(or any -ve Integer constant)

----------Output.txt----------

->0

->1

->2

->3

...

...

...

->n

-1

writeFile() : it is used to write File Asynchronously.

Syntax :-

```
fs.writeFile("output.txt",content,"encoding",(error)=>{
   if(error == null){
            console.log("data written completed..");
          }
});
```

a) output.txt : file where data is written

mode : write

write mode : overwrite

b) content : Any data written in the content will be written inside file.

c) encoding : converting the buffer to standard encoding

utf-8, utf8 or binary.

d) callback function : to write the back in asynchronous mode,and

it has argument 1st => error

```
if(error == null){
  //any action file writing completed.
}
```

appendFileSync: same as writeFileSync, working and syntax is same

****

writeMode : append mode

data wont be lossed.

mode : synchronous

appendFile: same as witeFile

****

writeMode : append mode

data wont be lossed.

mode : Asynchronous.

unlink : this is also, asynchronous function used to, delete the file,

****

from, fileSystem.

mode : Asynchronous

No of arguments : 2

1. filename : relative url file, or complete url for filename.

2. callback. : its function which holds error as argument,

which tells if any error, occurred during deleting a file.

syntax :-

****

complete Url :-

```
fs.unlink("c:/path/to/filename.txt",(error)=>{

  if(error){
        //any action
        console.log('file deleted successfully');
  }

})
```

Relative Url:-

```
fs.unlink("./filename.txt",(error)=>{

  if(error){
        //any action
        console.log('file deleted successfully');
  }

})
```

or

```
fs.unlink("../public/filename.txt",(error)=>{

  if(error){
        //any action
        console.log('file deleted successfully');
  }
```

});

stat:- it is used see file related information.

mode : asynchronous

argument : 2

1. path or filename : relative or absolute(complete path)

2. callBack Function : it is used to capture any error, and fileInformation about the file.

syntax:-

```
fs.stat("path/to/file",(eror,fileInfo)=>{
  if(error == null){
        console.log(fileInfo);
   }
})
```

---

task : convert the size of file to KB/Mb

****

Kb => fileInfo.size/1024

MB => (fileInfo.size/1024)/1024

or

mb => Kb/1024.

copy(): this is not in fs module

1. readFile the from source

2. writeFile the to target.

move():

1. readFile from source

2. writeFile to the destination

3.delete the source file. (unlink)

logger(): it is custom function used to log the activity of the user/error for this we use, writeFile() function, to keep the log with date and time. so that any actions or error occurred can be seen, later.

importance :-

in server, we are not allowed to console the output or error.

so in order to save the error at server level, and security purpose.

we generate the log file.

rename(): it is a function inside fs module used to, rename the file from old name to new Name

mode : asynchronous

arguments : 3

syntax:-

```
fs.rename("path/to/oldfilename","NewFilename",(error)=>{
   if(error == null){
            console.log("file renamed successfully");
          }
});
```

readdir() : it is function which executes ls command in cmd.

$ ls : show files and folder list.

shows list of files and folder.

mode : asynchronous

arguments : 2

syntax:-

```
fs.readdir("path/to/folder",(error,list)=>{
        if(error == null){
              console.log(list); //Object or array showing list of files and folders.
            }
    });
```

_dirname() : it is magic constant (mangeled)

1. global scope
2. donot require any module.
3. use : to show current directly path.

_filename() : it is also magic constant (manageled)

1. global scope
2. donot require any module.
3. use : to show current filename of the script

node index.js

_filename : index.js.

Note :: Name mangeling :

it is varible or constant naming convention to tell that it is special

function or do something special.

this : global scope constant object. [Non-mangeled]

window : global scope constant object. [Non-mangeled]

document : global scope constant object. [Non-mangeled]

__proto__: it is global scope [mangeled]

__dirname : global scope [mangeled]

__filename : global scope [mangeled]

it is very common technique in all langauge.

for eg : php,python,js, node Js.

Misconception : mostly __dirname and __filename are going be used, with fs module.

so people think __dirname and __filename are of fs module.

but due to mangling, they donot require fs module.

---

wrapper functions

****

wrapper : its is container or cover which covers any thing, for safe use.

in python :-

```
print('Hello world');
```

as per python standard every program must wrapped inside a wrapper function

```
def main():
  print('Hello world')
```

wrapper function : is not only the code of node

but it is available in c, c++, Java.

Every code you execute in Node Js is automatically covered with a wrapper function.

index.js

```
console.log('Hello world');
```

node index.js

Hello world

Internally

****

index.js <----------Fs module --------| internally source code read

*fp (file pointer) ----------> source code --------> V8-Engine

|

|

```
(function(exports,require,module,__filename,__dirname){
    Hello world ;
});
```

node index.js

Es6 standard in Node Js

we know that, React uses import statement which is run in V8 Engine.

why node does not support es6 import statement.

because of wrapper function, overrides behaviour of importing by require.

How to use import statement in Node Js

****

there are 2 ways

1. package.json => declare "type":"module" (standard)

2. rename file from .js to .mjs

react

 src

  index.js

node

 server

  index.js

first method :

****

this is standard way, this is how, you cannot front with backend

with same server file => index.js

Explain Error :

ReferenceError: require is not defined in ES module scope, you can use import instead

How to use Both :-

****

for this we have write 2 extra lines of code.

automatically you can use import and require together.

```
import {createRequire} from "module"
const require = createRequire(import.meta.url);

import fs from "fs"; //valid
const fs = require("fs"); //valid
```

Note :: when we use both import and require together due to ES6 Standard, specification

the wrapper function is overriden and all the constants.

like,

module,exports,__filename,__dirname,require

every constant is out of the scope.

Node Js

****

p1.js p2.js

a, -------> module.exports X a //out of scope

b, -------> module.exports X b //out of scope

c -------> module.exports X c //out of scope

in import syntax :-

****

p1.js p2.js

a, -------> export ✓ a //global scope

b, -------> export ✓ b //global scope

c -------> export ✓ c //global scope

Working With querystring module

****

it used to parse QueryString in get and post request.

querystring : parse method which converts key and value pairs.

converts in Object

var q = ?x=10&b=20&name=ravi&class=btech

query.parse(q);

```
{
 x:10,
 b:20,
 name:"ravi",
 class:"btech"
}
```

==========

25-02-2023

==========

Working with OS Module in Node Js

Os Module : is one most, popular core built module. which is directly or indirectly used by many, modules as dependency.

This is read only and unmutable module.

it has some predefined, functions and constant variables for getting information.

Syntax :-

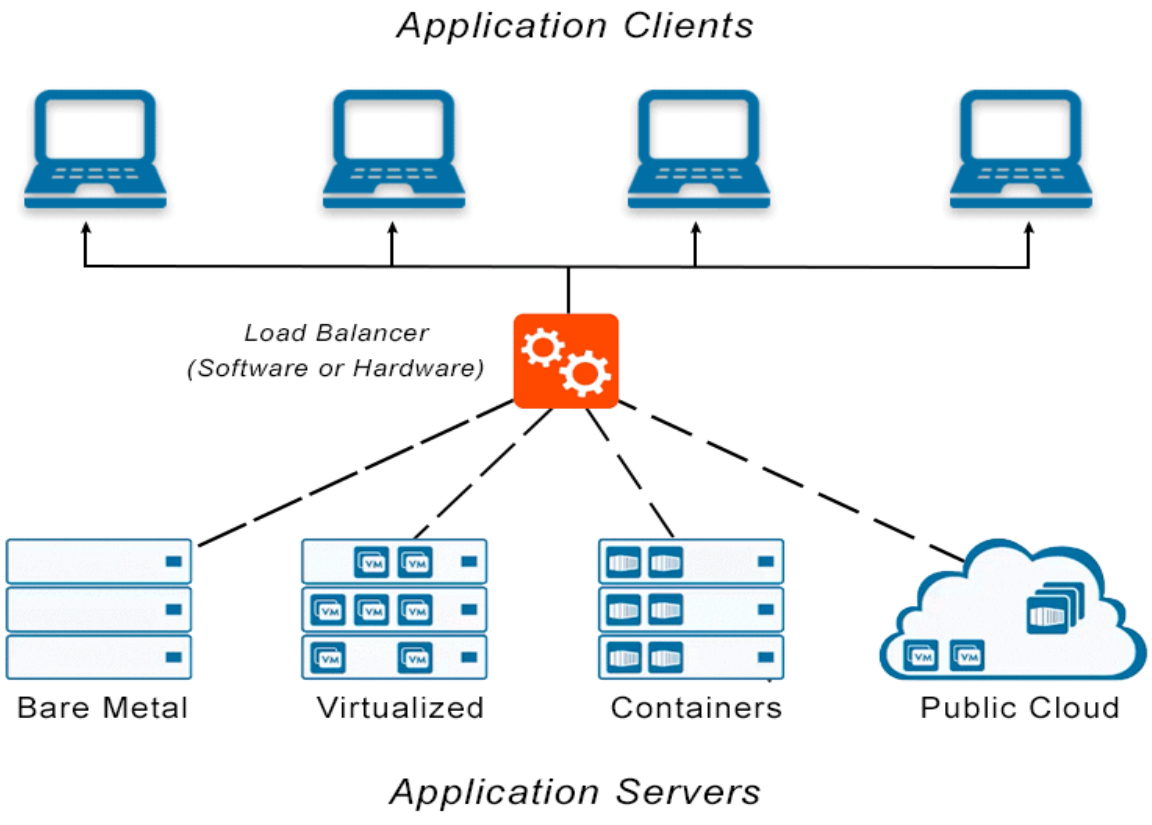
****

```
const os = require('os');

console.log(os)
```

```
const os = require('os');

//console.log(os);


console.log('End of Line = ',JSON.stringify(os.EOL));

//console.log('Constant Object = ',os.constants);

console.log('Free Memory = ',os.freemem());


os['constant'] = null;

console.log(os);


console.log('Architecture of Os : ',os.arch());

console.log('No of Cpus : ',os.cpus());

console.log('Hostname : ',os.hostname());

console.log('Avg Load of Machine : ',os.loadavg());

console.log('Total Memory : ',os.totalmem());

console.log('Os Type : ',os.type());

console.log('Release : ',os.release());
```

```
console.log('Os Platform : ',os.platform());
console.log('Temporary Path : ',os.tmpdir());
console.log('Home Path : ',os.homedir());
console.log('Priority : ',os.getPriority());

let consumedMemory = os.totalmem() - os.freemem();
let result = consumedMemory / os.totalmem();
console.log('Effeciency/performance/tolerance/latency : ',result);
console.log('slow rate : ',(1-result));
```

Load Balancer



Q. Why Os module is important ?

1.For getting system information, platform information.

2.For checking the Performance for managment of load Balancer in aws.

3.Used in pipelines to automatically reserve sockets, port according to available memory and Priority.

4.To detect EOL in file Handling by fs Module.

os Module is > dependency > fs Module.