Notes on Node.js - THENETNINJA

About Node.js

- Node.js is not a framework nor a library, BUT it is a Runtime Environment based on Chrome's V8 JavaScript Engine which is an open source and high performance JavaScript Engine.
- node on V8 engine in which both are written in C++.
- V8 engine is Standalone and can be embedded into any C++ application.
- It allows us to build scalable network applications using an Event driven, NON-blocking IO, which makes it fast and light on resources.
- by Node.js, now we can use JavaScript as server side language.
- JavaScript passed into V8 engine and it converts it into Machine Code.

Inside Node.js

- Global Object - which is similar to window object in client side providing all basic functions like console, setInterval, clearInterval etc. It provides extra objects like module, exports etc.
- Function Expressions - Assigning an Anonymous function to a variable which can be used to call it or pass as arguments to another function.
  - Ex: var counter = function (){};
    "function (){}" -> called as anonymous function because the function as no name.

Node.js Exporting Modules

- to reuse the modules, export it and import that module using "require" function. This makes function to be accessed outside the module.
- Ex: In count.js:-

```
var counter = (arr) => { return "There are " + arr.length + " elements in the given
                          array";}
module.exports = counter
                                     or

module.exports.counter = (arr) => { return "There are " + arr.length + " elements
                                     in the given array";}
```

```
//importing module
var counter = require(“. /count")
console.log(counter([1,2,3]))
```

- Module patterns are:-
  1. module.exports.counter = funtion(){}
     or
  2. var counter = funtion (){}
     module.exports = counter
     or
  3. var counter = funtion (){}
     var add = funtion (){}
     module.exports = { counter: counter, add: add }

Server, Stream, Buffer and Routing

- Using 'http' module, we can create server.
  Ex:

```
var http = require('http')
var server = http.createServer((req, res)=>{ //remaining code })
server.listen(PORT_NUMBER, () => { //call back funtion })
```

  In response, make sure this line *“res.writeHead(200, {'Content-type': 'text/html'})”*

Content-type:-
- text/plain - returns a file as plain text
- text/html - recognises a file as html
- application/json – returns contents in json format and can be convert into string using JSON.stringfy()

➢ Using 'fs' module, we can read and write file from server at a time or through stream or buffering.
Ex:
Read Stream:-
```
let fs = require("fs");
readStream = fs.createReadStream(__dirname + "/notes.txt", "utf-8");
readStream.on("data", (chunk) => {
      console.log("Read chuck of data");
});
```
Write Stream:-
```
let fs = require("fs");
let readStream = fs.createReadStream(__dirname + "/notes.txt", "utf-8");
let writeStream = fs.createWriteStream(__dirname + "/readme.txt");
readStream.on("data", (chunk) => {
      console.log("Read chuck of data", chunk.length);
      writeStream.write(chunk, () => {
            console.log("Written chunk of data successfully");
      });
});
```
➢ *Using readStream.pipe(writeStream) can work even faster and reduce no of lines of code.
Instead of listening readStream, the pipe() write data directly from readStream to writeStream.
Ex:
```
let fs = require("fs");
let readStream = fs.createReadStream(__dirname + "/notes_copy.txt", "utf-8");
let writeStream = fs.createWriteStream(__dirname + "/readme.txt");
readStream.pipe(writeStream);
                                        or
fs.createReadStream(__dirname + "/notes_copy.txt", "utf-8").pipe(res)
//write directly to request.
```

Implement routing using if else statements and comparing "req.url" with different link
```
Ex:     if(req.url === "/home" or req.url === "/"){
              req.writeHead(200,{'Content-type':'text/html'});
              req.end('index.html');
        }
        else{
              req.writeHead(200,{'Content-type':'text/html'});
              req.end('404.html');
        }
```

<u>Usage of Express, Nodemon, Templating Engine and Query url</u>

➢ Installation:-
- create package.json for maintain dependencies using cmd: npm init
- install express package and save the dependency: npm install express -g –save
- install nodemon package and save the dependency: npm install nodemon -g -save

Explanation:-
- Nodemon is used to as live server and any changes are made to files and when we save, it will automatically render the changes.
- Express is easy and flexible routing system and it integrates with many templating system and contains a middleware framework.

- Types of HTTP request: GET, POST, DEL and PUT

- Templating Engine:- EJS
- Usage of templating engines makes rendering faster. I used "ejs". So install and save it.
- Now ejs by default look for .ejs file under view folder from current folder of app.js or main file.
  So, create view folder and .ejs file inside it.
  Working:-
  1. set view engine i.e.,
     ```
     let app = express()
     app.set('view engine', 'ejs')
     ```
  2. render to respose object i.e., res.render(NAME_OF_EJS_FILE, params as object)
     Ex:-for "views/profile.ejs" - res.render('profile', {person: req.params.name})
     Inside ejs file, destructure passed arguments i.e.,
     `<p> Name: <%= person %></p>`

     Send custom data.
     Ex:-
     ```
     let data = {age: 30, name: 'New'};
     res.render('profile', {data: data});
     ```
     Inside ejs file, `<p><%= data.age %><%= data.name %></p>`

  - Query Url:- ?name=value
    Ex:-
    1. Query for single attribute:
       mysite.com/blog/news?page=2 - Query for page 2 of news
    2. Query for multiple attributes: Here multiple attributes are combined using '&' symbol and multiple attributes can be any order.
       mysite.com/contact?person=uru&dept=marketing
       or
       mysite.com/contact?dept=marketing&person=uru
       - Query contact of 'person' named "uru" in "marketing" 'department'
    3. We can get attributes using 'req' argument i.e., req.query() -> it returns query as object in {name: value} format.
    4. POST requests:- It is a request method which ask to store/retrieve data enclosed in the body of the request. Often used when submitting forms.
       Usage:-
       1. add an attribute called 'method' and 'action' to 'form' tag.
          method attribute specifies type of a request and action specifies where to send the form data
          ex:- `<form method="POST" action="/contact"></form>`
       Note:- input inside form tag should contain the name attribute.
       2. Install a middleware called 'body-parser', which is used to parse incoming request.
          ex:-
          ```
          let bodyParser = require('body-parser');
          urlencodedparser = bodyParser.urlencoded({extended: false})
          app.post ("/contact", urlencodedparser, (req, res) => {
                console.log (req. body)
          })
          ```

------------------------------------- ***THE END*** -------------------------------------

For More details, See official documentation of Node.js and other packages.